Image Compression by Learning to Minimize the Total Error

Chiyuan Zhang and Xiaofei He, *Senior Member, IEEE*

***Abstract*—In this paper, we consider the problem of lossy image compression. Recently, machine learning techniques have been introduced as effective mechanisms for image compression. The compression involves storing only the grayscale image and a few carefully selected color pixel seeds. For decompression, regression models are learned with the stored data to predict the missing colors. This reduces image compression to standard active learning and semisupervised learning problems. In this paper, we propose a novel algorithm that makes use of all the colors (instead of only the colors of the selected seeds) available during the encoding stage. By minimizing the total color prediction error, our method can achieve a better compression ratio and better colorization quality than previous methods. The experimental results demonstrate the effectiveness of our proposed algorithm.**

***Index Terms*—Active learning, image compression, semisupervised learning.**

I. Introduction

With the arrival of the internet and the multimedia age, the number of images available online has grown rapidly, and there is an increasing demand for better image compression techniques [1]–[6]. Typically, there is a lot of redundancy in the image data (e.g., the nearby pixels are usually correlated). With a sophisticated compression algorithm, we can greatly reduce the space required to store an image. Classical image compression algorithms (e.g., JPEG) transform the image from a spatial domain representation to a frequency domain representation and use a carefully designed encoding scheme to store the coefficients of the frequency domain. At the decoding stage, the frequency coefficients are recovered and transformed back into the spatial domain to get the normal image pixel data.

Recently, Cheng *et al.* [7] proposed to solve the image compression problem with machine learning techniques and achieved impressive results. Instead of performing a frequency transformation, they transformed the image into a grayscale version. At the encoding (compression) stage, they stored only the grayscale image plus a few representative color pixels. During the stage of decoding (decompression), the stored information is used in the learning of a model to colorize the remaining grayscale pixels [8].

There are two key steps in the machine-learning-based image compression framework: selecting the most informative color pixels and learning to predict the color values. They are modeled and solved with two standard machine learning techniques: active learning and semisupervised learning, respectively. We will give a more detailed explanation of machine-learning-based image compression technique in the next section.

This idea, based on machine learning techniques, sheds new light on image compression problems. The potential importance of learning methods for image compression is mainly in two aspects. First, the formulation of the image compression problem as a standard machine learning problem makes it possible to utilize techniques and incorporate new advances from the machine learning community. For example, new techniques in semisupervised learning like Hessian regularized regression [9] and parallel vector field regularized regression [10] can be adopted here to improve the colorization quality. Second, although currently machine learning-based algorithms are still costly concerning both memory and time consumption, the quality of the compressed image and the compression ratio are already competitive with those of the standard techniques; there are still other ways to improve the current prototype (e.g., the selected color pixels can also be compressed when storing). Actually, the classical image compression techniques focus mainly on the grayscale component, while the machine learning-based methods focus mainly on the color components. So it would also be natural to combine the advantages of the two kinds of methods.

In the work of Cheng and Vishwanathan [7] and a follow-up by He *et al.* [11], general-purpose machine learning algorithms were applied. But we note that the image compression problem is fundamentally different from general machine learning problems in that the ground truth colors are all known at the encoding stage. By utilizing this, we can greatly improve the performance. Specifically, with all the ground truth colors available, we can select the color pixel seeds to minimize the total error (i.e., the prediction error for all pixels). But He *et al.* [11] simply minimized the expected error and the method in [7] is completely heuristic. Besides, we also store the difference-image between the true colors and the predicted colors. At the decoding stage, we first colorize the grayscale image with our

Manuscript received January 9, 2012; revised May 15, 2012; accepted July 1, 2012. Date of publication August 3, 2012; date of current version April 1, 2013. This work was supported in part by the National Natural Science Foundation of China under Grants 61125203 and 90920303 and the National Basic Research Program of China (973 Program) under Grant 2009CB320801. This paper was recommended by Associate Editor L.-P. Chau.

The authors are with the Department of Computer Science, Zhejiang University, Hangzhou, Zhejiang 310058, China (e-mail: pluskid@gmail.com; xiaofeihe@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2210803

1051-8215/$31.00 © 2012 IEEE

learned model and then add the difference image to the predicted colors to correct the prediction error. The experimental results show that our algorithm achieves a better compression ratio and greatly improves the colorization quality.

The remainder of this paper is organized as follows. In Section II, we give an introduction of how the machine learning-based methods work. A review of the previous work is given in Section III. Section IV introduces our total error minimization (TEM) algorithm for labeled point selection. TEM-Compensate (TEM-C), the improved version of our algorithm that uses the difference-image for better colorization quality, is introduced in Section V. A variety of experimental results are depicted in Section VI and, finally, we conclude in Section VII.

II. Machine Learning-Based Method for Image Compression

Machine learning-based method works in the YCbCr space, where Y is the luminance channel (i.e., the grayscale image), and Cb and Cr are the chrominance channels containing color information. Basically, the luminance channel is stored and the chrominance channels are predicted from models.

The methods for color prediction of the Cb and Cr channels are virtually the same, so we only describe a generic model here. The prediction function f gives the predicted colors when given the grayscale value I and the location (X, Y) of a pixel. The location of a pixel is considered because there might be many pixels with the same grayscale value but different chrominance values in an image.

In the standard machine learning terms, we denote the vector $\mathbf{x} = (I, X, Y)$ the feature vector for each pixel. We also call the colors (values to be predicted) as *labels*, denoted by y for each pixel.

To train our model f at the decoding stage, we need a set of labeled pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ as the training data. So we select and store l color pixel seeds. Since we actually store the whole luminance channel, we also have a bunch of unlabeled pixels $\{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n\}$, where n is the total number of pixels in the image. The standard learning paradigm that can make use of both labeled and unlabeled training data is semisupervised learning. In this paper, we use Laplacian regularized least square (LapRLS) [12], which is a state-of-the-art semisupervised learning algorithm. LapRLS is formulated on the graph assumption, which has become popular in recent literature of manifold learning [13]–[17]. Specifically, it assumes that when two data points are close, their corresponding labels should also be similar. In our problem, this means that when too many pixels are spatially close and have similar grayscale values, their colors should also be similar. This is a reasonable assumption for images and this algorithm has been shown to perform well in previous studies [7], [11].

Traditional machine learning algorithms simply learn from pregiven training data and assume that the data points are independently and identically sampled from the underlying (unknown) probability distribution. However, the selection of the color pixel seeds is very important here as different color pixel seeds may lead to very different color predictions. Active learning is the standard learning paradigm for the this situation. So instead of random sampling, an active learning algorithm is applied here to select color seeds that can achieve good prediction accuracy.

In summary, at the encoding stage, an active learning algorithm is applied to select a subset of the pixels as seeds. Then the whole luminance channel and the colors for the seeds are stored. At the decoding stage, the stored luminance channel and the color seeds are used to train a prediction model with a semisupervised learning algorithm. The model is then applied to all the pixels to get the predicted colors.

III. Previous Work

The idea of using a machine learning-based method for image compression was initially proposed by Cheng and Vishwanathan [7]. He *et al.* [11] largely followed their framework. As described in the previous section, the feature vector for each pixel contains its luminance value and location. Cheng and Vishwanathan also included local texture in their feature vector. From now on, we will uniformly denote the space of the feature vectors by $\mathcal{X}$.[1]

Let $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\} \subset \mathcal{X}$ be the feature vectors for all the pixels in an image. At the encoding stage, the whole grayscale channel plus the color labels for a selected subset $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\} \subset U$ are stored.

To select the color seeds, Cheng and Vishwanathan [7] used a method that iteratively picks random points from regions with a high prediction error. However, their method is purely heuristic and does not have any guarantee to get good results. Actually, it is shown in [11] that Cheng's method cannot perform well when the number of sampled points is limited. Moreover, they rely on normalized cut [18] to segment regions for random sampling, making the computational burden heavier.

He *et al.* proposed an active learning algorithm named graph regularized experimental design (GRED) in [11] and showed that it is more effective than Cheng's heuristic procedure. The GRED algorithm is based on the statistical theories of optimal experimental design [19], which has recently attracted much interest from the active learning community [20], [21]. The key observation here is that the variance of the parameters of the regression function can be estimated without knowing the labels in advance. So the objective function of GRED is to select color seeds by minimizing the variance, in order to obtain a more stable model. However, for the problem of image compression, color seeds selection is done at the encoding stage, in which colors for all pixels are available for investigation. This important fact is different from the classical active learning setting and is ignored by GRED. As we will show in the remainder of this paper, by making use of this fact, we can improve the image compression performance significantly.

Both Cheng's method and GRED use LapRLS [12] to learn the regression model for color prediction at the decoding stage.

[1] $\mathcal{X} = \mathbb{R}^3$ for He's and our methods. For Cheng's method, $\mathcal{X} = \mathbb{R}^6$ because they use extra local texture information in the feature vectors.

IV. Point Selection by TEM

A. Motivations

Classical active learning algorithms are proposed in the situation in which the label is expensive to acquire and the algorithms do not know the corresponding label before choosing the data point. However, in image compression, all the labels are available at hand—they are not expensive to acquire but to store.

In this situation, for any given set of selected pixels and the resultant regression function trained with them, we can actually evaluate the prediction error over all the pixels, i.e., the total error. So we design a point selection algorithm that explicitly minimizes the total error. Moreover, with all the labels available, the predicted result of the regression function can be corrected and further improved.

Compared with previous methods, it is worthwhile to highlight several aspects of the algorithm proposed in this paper.

1) Previous methods for point selection are either heuristic or aiming at minimizing the variance of the parameters. Therefore, they cannot directly guarantee the quality of predictions. On the contrary, our approach uses the full label set to minimize the total prediction error explicitly.
2) To further improve the colorization quality, we introduce the difference image, which is the difference between the original image and the one recovered by the regression model. It can be stored with very little space consumption. With the difference image, we can compensate the recovered image by the regression model to get much better colorization quality.
3) Efficient iterative optimization procedures are developed to optimize the objective function. Besides, by making an explicit nonlinear map, we use linear LapRLS to significantly reduce the computation burden.

We will first describe the semisupervised regression algorithm to predict the missing colors and then present the objective function of our algorithm for point selection.

In order to predict the colors of an image, we first construct the feature vector of each point from the grayscale image. Then we obtain a few values for each of the Cb and Cr color channel as training labels. Finally, we learn two prediction functions f_{Cb} and f_{Cr} from the given data so that they can give color predictions from the grayscale image.

Before considering how to select a good subset of labeled points, we first formulate how to train the regression model when this subset is already given. Denote the set of all points by $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$, the subset of l labeled points by $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\} \subset U$, and their corresponding color labels by $\{y_1, \ldots, y_l\}$. We learn the prediction function by minimizing the prediction error on the set of the labeled training points

$$\begin{aligned} \arg\min_{f \in \mathcal{H}_\kappa} J(f) &= \arg\min_{f \in \mathcal{H}_\kappa} \sum_{i=1}^{l} (y_i - f(\mathbf{x}_i))^2 \\ &= \arg\min_{f \in \mathcal{H}_\kappa} \|Y_X - F_X\|^2 \end{aligned} \tag{1}$$

where $Y_X = [y_1, \ldots, y_l]^\top$, $F_X = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_l)]^\top$, and $\mathcal{H}_\kappa$ is the reproducing kernel Hilbert space (RKHS) associated with the kernel κ (to be defined shortly).

In order to make use of the unlabeled data points, we make an assumption that in an image, if two points are spatially close and have similar grayscale values, then their colors should also be similar. This is usually known as the local consistency assumption or manifold assumption [12]. Since the feature vector for each point consists of the coordinate and the grayscale value, the assumption translates into that if $\|\mathbf{u}_i - \mathbf{u}_j\|$ is small, $|f(\mathbf{u}_i) - f(\mathbf{u}_j)|^2$ should also be small.

This assumption can be implemented by a graph regularizer. Specifically, we construct a graph to encode the similarity information. The vertices of the graph are all the data points. If $\mathbf{u}_i$ and $\mathbf{u}_j$ are close in Euclidean distance (i.e., the two pixels are spatially close and have similar luminance values), they will be connected by an edge. Optionally, we can assign a weight to measure the similarity between $\mathbf{u}_i$ and $\mathbf{u}_j$. There are two popular methods to construct similarity graphs from feature vectors: K-nearest neighbors (KNN) and ε-similarity. The KNN graph is constructed by connecting $\mathbf{u}_i$ and $\mathbf{u}_j$ if $\mathbf{u}_i$ is within the k nearest neighbors of $\mathbf{u}_j$ *or* $\mathbf{u}_j$ is within the k nearest neighbors of $\mathbf{u}_i$. The ε-graph is constructed by connecting $\mathbf{u}_i$ and $\mathbf{u}_j$ if $\|\mathbf{u}_i - \mathbf{u}_j\| < \varepsilon$. In this paper, we use KNN graph and all the weights are assigned to the constant 1. We represent the graph with a matrix $W \in \mathbb{R}^{n \times n}$, where

$$W_{ij} = \begin{cases} 1, & \mathbf{u}_i \text{ and } \mathbf{u}_j \text{ are connected} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

With this graph, we can implement the manifold assumption by minimizing the following term:

$$\mathcal{R}(f) = \frac{1}{2} \sum_{i,j=1}^{n} W_{ij} (f(\mathbf{u}_i) - f(\mathbf{u}_j))^2. \tag{3}$$

To understand this term, we note that $W_{ij} = 1$ if and only if $\mathbf{u}_i$ and $\mathbf{u}_j$ are similar. In this case, the term $(f(\mathbf{u}_i) - f(\mathbf{u}_j))^2$ is included in the final summation. This means the difference between the colors for pixel $\mathbf{u}_i$ and $\mathbf{u}_j$ must be made small in order to minimize our objective function. On the other hand, when $W_{ij} = 0$, the term $(f(\mathbf{u}_i) - f(\mathbf{u}_j))^2$ will not be present because of a zero coefficient. This means we do not make constraints on the color values between pixels that are not similar to each other (i.e., not connected in the graph).

Let D be a diagonal matrix whose diagonal elements are the row (or column, since W is symmetric) sums of W, i.e., $D_{ii} = \sum_{j=1}^{n} W_{ij}$. The matrix $L = D - W$ is called graph Laplacian matrix [22]. Then, (3) can be written more compactly as

$$\begin{aligned} \mathcal{R}(f) &= \frac{1}{2} \sum_{i,j=1}^{n} W_{ij} (f(\mathbf{u}_i)^2 + f(\mathbf{u}_j)^2 - 2 f(\mathbf{u}_i) f(\mathbf{u}_j)) \\ &= \sum_{i=1}^{n} f(\mathbf{u}_i)^2 \sum_{j=1}^{n} W_{ij} - \sum_{i,j=1}^{n} W_{ij} f(\mathbf{u}_i) f(\mathbf{u}_j) \\ &= F_U^\top D F_U - F_U^\top W F_U = F_U^\top L F_U \end{aligned} \tag{4}$$

where $F_U = [f(\mathbf{u}_1), \ldots, f(\mathbf{u}_n)]^\top$. Combining this term with (1), we obtain the objective function of LapRLS [12]

$$\arg\min_{f \in \mathcal{H}_\kappa} \left\{ J(f) = \|Y_X - F_X\|^2 + \lambda_1 \|f\|^2 + \lambda_2 F_U^\top L F_U \right\}. \tag{5}$$

Here the norm in the function space $\|f\|^2$ is another regularizer that ensures the numerical stability of the solution. In order to specify the RKHS $\mathcal{H}_\kappa$, we follow the previous work to use the Gaussian kernel $\kappa(\mathbf{u}_i, \mathbf{u}_j) = \exp(\|\mathbf{u}_i - \mathbf{u}_j\|^2/\sigma^2)$, where σ is the bandwidth parameter of the kernel. Then H_κ will be the completion of the linear span given by $\kappa(\cdot, x)$ for all possible $x \in \mathcal{X}$. By representer theorem, the optimal solution of (5) can be represented as

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i \kappa(\cdot, \mathbf{u}_i). \tag{6}$$

With this representation, we have

$$F_U = \begin{bmatrix} f(\mathbf{u}_1) \\ \vdots \\ f(\mathbf{u}_n) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \alpha_i \kappa(\mathbf{u}_1, \mathbf{u}_i) \\ \vdots \\ \sum_{i=1}^{n} \alpha_i \kappa(\mathbf{u}_n, \mathbf{u}_i) \end{bmatrix} = K\boldsymbol{\alpha} \tag{7}$$

where K is the kernel gram matrix with $K_{ij} = \kappa(\mathbf{u}_i, \mathbf{u}_j)$. Denote by K_{XU} the submatrix consisting of rows of K corresponding to those training points in the set X. Similarly, we have $F_X = K_{\mathrm{XU}}\boldsymbol{\alpha}$. Substituting this representation back into (5), the minimization problem becomes a quadratic problem with respect to the coefficient vector $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^\top$

$$\arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ J(\boldsymbol{\alpha}) = \|Y_X - K_{\mathrm{XU}}\boldsymbol{\alpha}\|^2 + \lambda_1 \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + \lambda_2 \boldsymbol{\alpha}^\top KLK\boldsymbol{\alpha} \right\}. \tag{8}$$

Setting the derivative $\partial J(\boldsymbol{\alpha})/\partial \boldsymbol{\alpha} = 0$, we obtain the optimal solution in a closed form as

$$\boldsymbol{\alpha}^* = (K_{\mathrm{UX}} K_{\mathrm{UX}}^\top + \lambda_1 K + \lambda_2 KLK)^{-1} K_{\mathrm{UX}} Y_X. \tag{9}$$

Note that $K_{\mathrm{UX}} = K_{\mathrm{XU}}^\top$. Substituting this optimal coefficient vector into (7), we get the optimal predicted values for all the points. Thus, we can calculate the total prediction error for all the points as

$$\mathcal{E} = \|K\boldsymbol{\alpha}^* - Y_U\|^2. \tag{10}$$

In order to recover the colors as faithfully as possible, we want the total error to be minimized. This is achieved by selecting the subset $X \subset U$ to minimize $\mathcal{E}$. So we define the objective of our point selection algorithm as follows.

Definition 1: TEM:

$$\arg\min_{X \subset U, |X| = l} \left\{ \mathcal{E} = \|K\boldsymbol{\alpha}^* - Y_U\|^2 \right\} \tag{11}$$

where $|X|$ means the cardinality of the set X and $\boldsymbol{\alpha}^*$ is the optimal coefficient vector given by (9).

B. Sequential Optimization of TEM

We develop a sequential algorithm to solve (11). This derivation of the sequential optimization procedure is inspired from [11]. We tested two variants: one that starts with an empty set and iteratively adds new color seeds, and the other that starts with all the pixels and iteratively removes pixels. We observed that on the same image, when the final number of color seeds are the same, the latter variant generally performs better. So the latter variant is adopted in our algorithm.

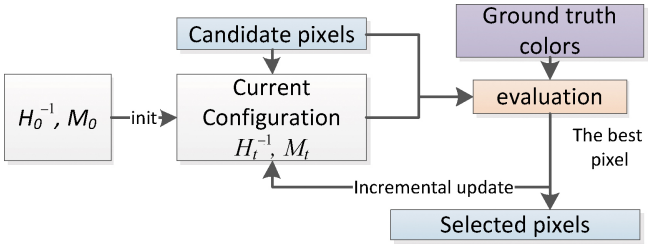


Fig. 1. Flowchart of the point selection process. One point is selected in each iteration. The process stops when the number of selected points arrives at the maximum or the required colorization quality is achieved.

Considering the LapRLS solution for a given labeled set X as shown in (9), we can rewrite it as

$$\boldsymbol{\alpha}^* = \left(\sum_{\mathbf{x} \in X} K_{\mathbf{x}} K_{\mathbf{x}}^\top + \lambda_1 K + \lambda_2 KLK \right)^{-1} \left(\sum_{\mathbf{x} \in X} K_{\mathbf{x}} y_{\mathbf{x}} \right) \tag{12}$$

where $K_{\mathbf{x}} = [\kappa(\mathbf{x}, \mathbf{u}_1), \ldots, \kappa(\mathbf{x}, \mathbf{u}_n)]^\top$ and $y_{\mathbf{x}}$ is the label of the point $\mathbf{x} \in X$. Let $H = K_{\mathrm{UX}} K_{\mathrm{UX}}^\top + \lambda_1 K + \lambda_2 KLK$. Then it is easy to see that if we remove a point $\mathbf{v}$ from X, the new LapRLS solution will be

$$\boldsymbol{\alpha}^* = \left(H - K_{\mathbf{v}} K_{\mathbf{v}}^\top\right)^{-1} (K_{\mathrm{UX}} Y_X - K_{\mathbf{v}} y_{\mathbf{v}}). \tag{13}$$

With the Sherman–Morrison formula [23], we can update the matrix inverse efficiently as

$$\left(H - K_{\mathbf{v}} K_{\mathbf{v}}^\top\right)^{-1} = H^{-1} + \frac{H^{-1} K_{\mathbf{v}} K_{\mathbf{v}}^\top H^{-1}}{1 - K_{\mathbf{v}}^\top H^{-1} K_{\mathbf{v}}}. \tag{14}$$

Once we get the updated LapRLS coefficients, the total error in (10) can be updated accordingly. We call this total error the would-be total error for the point $\mathbf{v}$. By computing the would-be total error for each point in X, we can choose to remove from them the one with the smallest would-be total error.

We summarize the iterative procedure as follows. Initially, we have the full dataset $X = U$. Let $H_0^{-1} = (KK^\top + \lambda_1 K + \lambda_2 KLK)^{-1}$ and $M_0 = KY_U$, so the initial total error is

$$\mathcal{E} = \|KH_0^{-1} M_0 - Y_U\|^2. \tag{15}$$

Consider the situation where we have removed $t (t \geq 0)$ points from X. If $t = n - l$, we can stop and return X. Otherwise, we will continue to select the $(t+1)$th point as

$$\mathbf{v}^* = \arg\min_{\mathbf{v} \in X} \|K(H_t - K_{\mathbf{v}} K_{\mathbf{v}}^\top)^{-1} (M_t - K_{\mathbf{v}} y_{\mathbf{v}}) - Y_U\|^2. \tag{16}$$

After removing $\mathbf{v}^*$ from X, we update $H_{t+1}^{-1} = (H_t - K_{\mathbf{v}^*} K_{\mathbf{v}^*}^\top)^{-1}$, $M_{t+1} = M_t - K_{\mathbf{v}^*} y_{\mathbf{v}^*}$.

With this algorithm, we are guaranteed to minimize the total error at each iteration. Also note that we only need to compute the inverse matrix H_0^{-1} at the initial step and all subsequent inverses can be updated incrementally with (14).

C. Improve Computation Efficiency

Although formulated as an iterative procedure, the computation cost for our algorithm is still high. It can be shown that the complexity for selecting one point is $O(n^3)$. In this section, we make a modification to the algorithm that reduces the complexity to $O(nm^2)$. Specifically, we explicitly map the data points to an m-dimensional feature space and formulate linear LapRLS in that space. By choosing a small m, we can greatly reduce the computation cost.

In order to obtain a nonlinear map with similar capacity to the original kernel feature mapping φ, we use the kernel principal component analysis (PCA) map [24].

Definition 2: Kernel PCA Map [24]: For a given set $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\} \subset \mathcal{X}$, we call $\varphi_Z : \mathcal{X} \to \mathbb{R}^m$, where $\mathbf{x} \mapsto \varphi_Z(\mathbf{x}) = K_{ZZ}^{-1/2}[\kappa(\mathbf{x}, \mathbf{z}_1), \ldots, \kappa(\mathbf{x}, \mathbf{z}_m)]^\top$ the kernel PCA map with respect to Z. Here, K_{ZZ} is the $m \times m$ kernel gram matrix whose (i, j)th element is $\kappa(\mathbf{z}_i, \mathbf{z}_j)$.

Kernel PCA map and its variant without the leading $K_{ZZ}^{-1/2}$ (called the empirical kernel map [25]) is usually used as a feature extraction technique to construct nonlinear features. Here we make an explicit connection between the linear regression in the mapped feature space and the original nonlinear kernel regression problem.

To use a less cumbersome notation, we use a tilde to represent the kernel PCA map, i.e., $\tilde{\mathbf{u}}_i = \varphi_Z(\mathbf{u}_i)$ and $\tilde{\mathbf{x}}_i = \varphi_Z(\mathbf{x}_i)$. By linear LapRLS we mean that a linear kernel function $\kappa(\tilde{\mathbf{u}}_i, \tilde{\mathbf{u}}_j) = \tilde{\mathbf{u}}_i^\top \tilde{\mathbf{u}}_j$ is used. With this special kernel, the representation of f in (6) can be written as

$$f(\tilde{\mathbf{u}}) = \sum_{i=1}^{n} \alpha_i \kappa(\tilde{\mathbf{u}}, \tilde{\mathbf{u}}_i) = \left(\sum_{i=1}^{n} \alpha_i \tilde{\mathbf{u}}_i^\top \right) \tilde{\mathbf{u}} := \mathbf{w}^\top \tilde{\mathbf{u}}. \tag{17}$$

Consequently, f becomes a linear function with the coefficient vector $\mathbf{w} \in \mathbb{R}^m$. Let $\tilde{U} = [\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_n]$ and $\tilde{X} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_l]$. Then we get

$$F_U = \tilde{U}^\top \mathbf{w} \tag{18}$$

$$F_X = \tilde{X}^\top \mathbf{w}. \tag{19}$$

Substituting those into (5), the objective function becomes a quadratic problem with respect to $\mathbf{w}$

$$\arg\min_{\mathbf{w} \in \mathbb{R}^m} \left\{ J(\mathbf{w}) = \|Y_X - \tilde{X}^\top \mathbf{w}\|^2 + \lambda_1 \mathbf{w}^\top \mathbf{w} + \lambda_2 \mathbf{w}^\top \tilde{U} L \tilde{U}^\top \mathbf{w} \right\}. \tag{20}$$

Setting the derivative $\partial J(\mathbf{w})/\partial \mathbf{w} = 0$, we obtain the optimal solution as

$$\mathbf{w}^* = (\tilde{X}\tilde{X}^\top + \lambda_1 I + \lambda_2 \tilde{U} L \tilde{U}^\top)^{-1} \tilde{X} Y_X. \tag{21}$$

Comparing this with (9), we find that to obtain the solution for a general kernel, a matrix of $n \times n$ must be inverted, while for a linear kernel, the inverse for a matrix of only $m \times m$ needs to be calculated. When the dimension of the feature space m is much smaller than the number of data points n, the computation efficiency can be improved significantly.

Combining (18) with (21), the total error becomes

$$\mathcal{E} = \|\tilde{U}^\top \mathbf{w}^* - Y_U\|. \tag{22}$$

Algorithm 1 Sequential point selection

Input: U: Dataset of all n candidate samples.
Input: l: Number of samples to select.
Output: Selected samples X.

```
X ← U
H_0^{-1} ← (Ũ Ũ^⊤ + λ_1 I + λ_2 Ũ L Ũ^⊤)^{-1}
M_0 ← Ũ Y_U
for t = 1 to n − l do
  for v in X do
    E_v = ‖Ũ^⊤ (H_{t−1} − ṽ ṽ^⊤)^{-1} (M_{t−1} − ṽ y_v) − Y_U‖^2
  end for
  v* ← arg min_{v∈X} E_v
  X ← X − {v*}
  H_t^{-1} ← (H_{t−1} − ṽ* ṽ*^⊤)^{-1}
  M_t ← M_{t−1} − ṽ* y_{v*}
end for
return X
```

Similar to the case of a generic kernel function, we develop an iterative procedure to select the subset X to minimize the total error $\mathcal{E}$. First, we rearrange (21) as

$$\mathbf{w}^* = \left(\sum_{\mathbf{x} \in X} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top + \lambda_1 I + \lambda_2 \tilde{U} L \tilde{U}^\top \right)^{-1} \left(\sum_{\mathbf{x} \in X} \tilde{\mathbf{x}} y_{\mathbf{x}} \right). \tag{23}$$

Initialize the set $X = U$. Let $H_0^{-1} = (\tilde{U}\tilde{U}^\top + \lambda_1 I + \lambda_2 \tilde{U} L \tilde{U}^\top)^{-1}$ and $M_0 = \tilde{U} Y_U$; then the initial total error is

$$\mathcal{E} = \|\tilde{U}^\top H_0^{-1} M_0 - Y_U\|^2. \tag{24}$$

Assume that $t (t \geq 0)$ points have been removed from X. If $t = n - l$, we stop and return X. Otherwise, we select the $(t + 1)$th point as

$$\mathbf{v}^* = \arg\min_{\mathbf{v} \in X} \|\tilde{U}^\top (H_t - \tilde{\mathbf{v}}\tilde{\mathbf{v}}^\top)^{-1} (M_t - \tilde{\mathbf{v}} y_{\mathbf{v}}) - Y_U\|^2. \tag{25}$$

To finish this iteration, we remove $\mathbf{v}^*$ from X and update $H_{t+1}^{-1} = (H_t - \tilde{\mathbf{v}}^*\tilde{\mathbf{v}}^{*\top})^{-1}$, $M_{t+1} = M_t - \tilde{\mathbf{v}}^* y_{\mathbf{v}^*}$. Note that, similarly to the case of (14), we can update the matrix inverse incrementally in each iteration. The detailed procedure is summarized in Algorithm 1 and a flowchart is also depicted in Fig. 1 for illustrative purposes.

This formulation looks very similar to the previous algorithm with a generic kernel function. However, the computation complexity for selecting one point drops from $O(n^3)$ to $O(nm^2)$. On the other hand, by using the kernel PCA map, we still retain the power of the nonlinear kernel function. Specifically, the previous algorithm with a generic kernel function is a special case of this framework when $Z = U$, which is stated formally in Theorem 1.

Theorem 1: When $Z = U$, the optimal function learned via a linear kernel on the feature set obtained from the kernel PCA map φ_Z is the same as the function learned via the original nonlinear kernel. Formally, we have

$$\mathbf{w}^{*\top} \tilde{\mathbf{u}} = \sum_{i=1}^{n} \alpha_i^* \kappa(\mathbf{u}, \mathbf{u}_i). \tag{26}$$

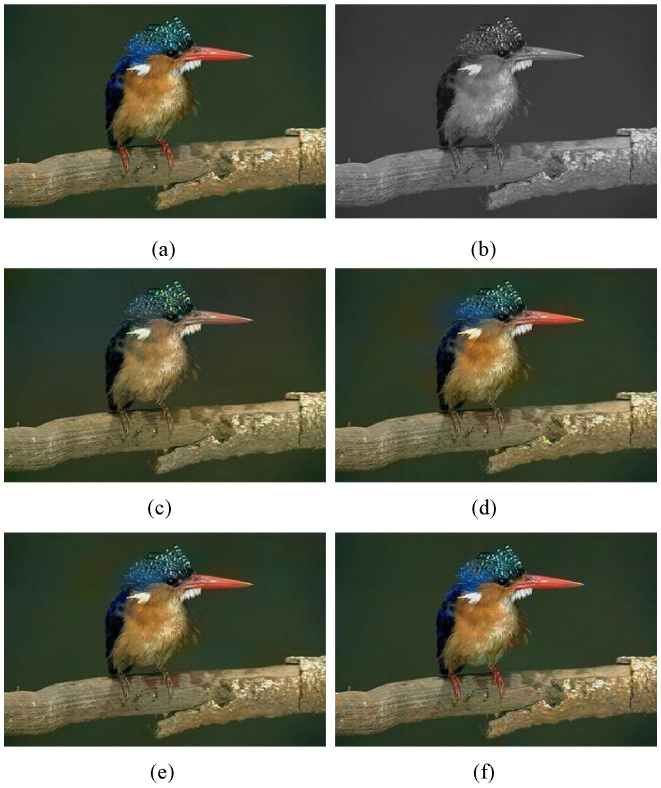


Fig. 2. Image compression example. The original 384 × 256 compressed image of 25% size and the decompressed results from various algorithms are shown. (a) Original image. (b) Gray image. (c) Cheng. (d) GRED. (e) TEM. (f) TEM-C.

The proof is provided in the appendix and we will empirically study how the size of the set Z affects the performance and computation cost in Section VI-E.

V. Colorization With Difference Image

So far, we have demonstrated that with the full color pixels available at the encoding stage, we can do better than traditional active learning to select the color seeds. In this section, we illustrate how we can use the full color pixels to calculate and store the "difference image" so that the colorization quality at the decoding stage can be further improved.

Consider an illustrative example in Fig. 2. The original image and its grayscale version are shown in Fig. 2(a) and (b), followed by the colorization results of various algorithms under the same compression ratio.

The result of Cheng's method looks grayish. GRED looks better except for the diffused color near the back of the bird's head and the wrong color for the feather near the neck. TEM further improves the result by reducing the color diffusion. However, all three algorithms fail to colorize the talons of the bird. In this section, we will analyze this problem and propose an improved algorithm named TEM-C as a solution. As depicted in Fig. 2(f), TEM-C colorizes the talons (and other details) very well and the result looks almost identical to the original image.

Basically, the colorization procedures via standard semisupervised regression algorithms are not good at handling small color patches that are very different from the surroundings. Because the mean square error loss function assigns equal weight for every pixel, small areas tend to get small weights as a sum and are considered to be unimportant. One possible solution is to detect those regions and assign higher weights. However, this solution requires a detector that is both accurate and computation efficient.

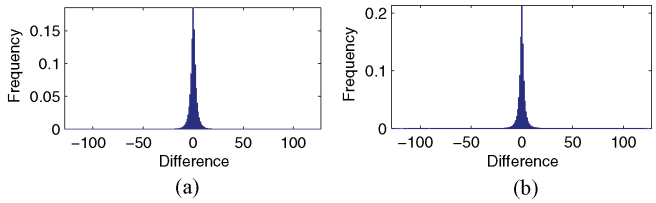


Fig. 3. Histogram of the difference image. Histogram of (a) Cb and (b) Cr channels. Near-zero values account for the majority.

Instead of searching for a good reweighting scheme, we adopt a much easier yet effective approach, which again exploits the full label set available at the encoding stage. Considering the colorization result of our TEM algorithm [e.g., Fig. 2(e)], we observe that the overall quality is good. Most parts of the image differ only slightly from the original image. There may be regions with totally wrong colors (e.g., the talons), but the sum of the area of those regions is small. So if we subtract the original image from the one recovered by TEM to get a difference image, the majority of the pixels in the difference image will be near zero. We randomly pick 500 images from the Corel image dataset, obtain regression results via TEM by selecting 250 labeled pixels each image, and count the pixel value of the difference image for the two color channels. The histograms are shown in Fig. 3. We can see that the majority values are concentrated at zero.

This property makes the difference image ideal to be compressed to a very compact representation via an entropy encoding scheme [26]. So, we improve our TEM algorithm by storing a compact difference image, and changing the colorization stage to two steps: 1) we use LapRLS to obtain a rough result and 2) the difference image is decoded and added to the regression result to get a compensated result. In order to leave space for the difference image, we select fewer pixels. The quality of the regression result will drop slightly. But when compensated with the difference image, we can reduce the total error to a level beyond the best attainable via a pure regression model. The result is visually appealing, especially for the small details that are very difficult for regression models to handle. We call the improved algorithm TEM-C.

This extension has an advantage that it is independent of the algorithm used to generate the initial colorization result. However, it does have some requirements for the underlying algorithm. Basically, the difference between the original image and the initial colorization result should be small in most parts. Or else the storage occupied by the compressed difference image will be so high that this extension becomes the bottleneck of the compression ratio. Ideally, we can achieve zero total error by using a lossless encoding scheme for the difference image. However, empirically, we find that the storage of lossless encoding is still a bit large, so we use

lossy encoding instead. The JasPer[2] utility is used to encode the difference image in our experiments. We will not further discuss the combination of difference image with other active learning algorithms or lossy versus lossless encoding in this paper due to the space limitation.

VI. Experiments

In this section, we first describe the implementation details. Then we present experimental results and comparisons of our methods with other algorithms.

A. Implementation Details

We use the MATLAB function rgb2ycbcr and ycbcr2rgb to convert between the color spaces without downsampling.

In TEM and TEM-C, we use the same 3-D feature vector as in the work of He *et al.* [11] (i.e., the gray value plus the pixel location). LapRLS needs to construct a neighborhood graph and a kernel gram matrix of size $n \times n$, where n is the number of pixels in an image. In order to reduce the memory and computation burden, a normalized cut [18] is used in [7] to obtain an oversegmented superpixel representation [27], and one pixel is randomly picked from each region for subsequent learning. However, a normalized cut is itself computationally expensive and He *et al.* [11] showed that simply segmenting the image uniformly into small rectangles is enough. We follow the latter scheme in TEM and TEM-C. In all our experiments, 2000 points are sampled from each image for subsequent learning.

To select suitable values for the hyperparameters, we randomly select some images and use grid search to find out the hyperparameters with the best average performance for each algorithm. Those images are picked from various categories of the Corel image dataset to avoid overfitting into some particular kind of images. To ensure fair comparison, all algorithms use the same set of images for parameter tuning. The tuned hyperparameters are then fixed and used in all the experiments. For KNN graph construction, we get $k = 4$. For the Gaussian kernel function, we set the parameter

$$\sigma = C \cdot \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{u}_i - \bar{\mathbf{u}}\| \tag{27}$$

where $\bar{\mathbf{u}}$ is the mean of all data points and C is a constant. The best parameter obtained for GRED, TEM, and TEM-C is $C = 0.34$, and for Cheng's method $C = 1.3$. Since extra local texture information is used, the parameter for Cheng's method is different. For the regularizer coefficients [see (5)], we get $\lambda_1 = 0.001$, $\lambda_2 = 0.002$ for TEM and TEM-C, $\lambda_1 = 0.005$, $\lambda_2 = 0.002$ for GRED, and $\lambda_1 = 0.006$, $\lambda_2 = 0.007$ for Cheng's method. For TEM and TEM-C, we use the kernel PCA map with $m = 100$ and choose the set Z via k-means clustering.

B. Evaluation on Colorization Quality

In this section, we compare the quality of the image compressed with Cheng's method, GRED, and our algorithms under the same compression ratio.

As for the evaluation of the performance of image recovery, we note that it is very subjective to judge whether a recovered image is "good" or "bad." So we simply use some objective image quality measurements. Specifically, the colorization quality is mainly evaluated with the peak signal-to-noise ratio (PSNR), which is a standard measurement commonly used in image compression community. The PSNR is calculated as

$$\text{PSNR} = 20 \log_{10} \frac{255}{\sqrt{\text{MSE}}} \tag{28}$$

where MSE is the mean squared error between the recovered image and the original image. We calculate PSNR for each channel and report the averaged PSNR. Since our (and the compared) algorithms only focused on compressing the chrominance channels and the luminance channel is handled with standard techniques, we only measure the PSNR of the chrominance channels here. Larger PSNR means smaller recover error and accordingly better quality. For reference, we also include results of some other well-known image quality assessments (IQA), including multiscale SSIM index (MS-SSIM) [28], visual signal-to-noise ratio (VSNR) [29], and noise quality measure (NQM). The performance of an IQA can be evaluated by its relevance to evaluations from human subjects. In [30], the authors collected a dataset with evaluations from 838 human observers and compared various IQAs on this dataset. According to their report, the Spearman correlation with the mean opinion score of the human evaluations of MS-SSIM, VSNR, and NQM are 0.853, 0.705, and 0.624, respectively. The code from MeTriX MuX[3] is used to compute these assessments in the two color channels and the average values are reported.

Besides, we also measure the effectiveness of the algorithms via compression ratio of the chrominance channels, which is defined as

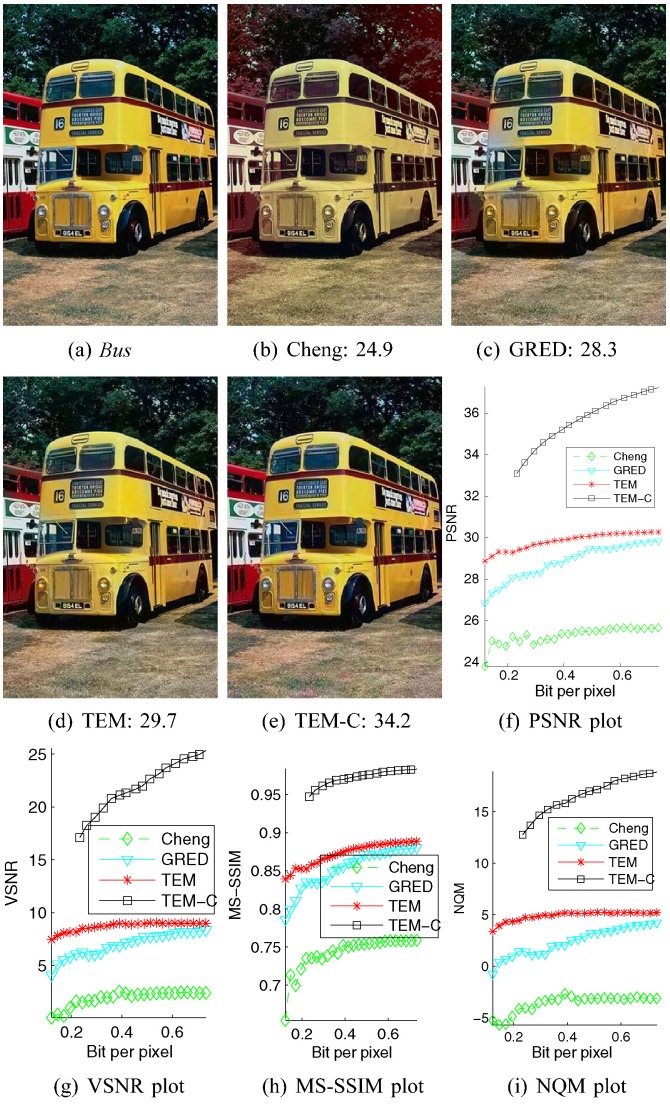
$$R = \frac{N}{48l + D_{\text{Cb}} + D_{\text{Cr}}} \tag{29}$$

where N is the number of bits in the chrominance channels, l is the number of stored color pixels, and D_{Cb} and D_{Cr} are the number of bits of the encoded difference image for the Cb and Cr channels, respectively. For example, for an image of 256×384, there are 98 304 pixels so $N = (8 + 8) \times 98\,304 = 1\,572\,864$ b, because each of the chrominance channels consumes 8 b per pixel. To store the selected color pixels, 32 b are needed for the x and y coordinates and 16 b for the Cb and Cr values. Note that D_{Cb} and D_{Cr} are 0 except for TEM-C.

Our first example is an image of buses of size 256×384, shown in Fig. 4(a). We select 600 color pixels with Cheng's method, GRED, and TEM, so the compression ratio is 55:1. The colorization results for the three algorithms are shown in Fig. 4(b)–(d). (The numbers in the caption indicate the PSNR of each result.) Cheng's method does not give a satisfactory result in this case. GRED is better but it mistakenly gives a white color near the left of the front of the yellow bus. TEM predicts the correct color there, but, as with the other two methods, colorizes the tree as red near the top boundary

[2]Available at http://www.ece.uvic.ca/~mdadams/jasper.

[3]Available at http://foulard.ece.cornell.edu/gaubatz/metrix_mux/#introduction



(a) *Bus* (b) Cheng: 24.9 (c) GRED: 28.3

(d) TEM: 29.7 (e) TEM-C: 34.2 (f) PSNR plot

(g) VSNR plot (h) MS-SSIM plot (i) NQM plot

Fig. 4. Colorization result for the *Bus* image. (a) Original image of size 256×384. (b)–(e) Colorization results at the compression ratio 55:1. (f) PSNR, (g) VSNR, (h) MS-SSIM, and (i) NQM value varying with the b/p for storing.

of the red bus. On the contrary, in Fig. 4(e), TEM-C gives a nearly identical result to the original image. In order to achieve the same compression ratio, TEM-C selects only 250 color pixels; the encoded difference images for the two color channels occupy 5376 and 5360 bits, respectively.

Fig. 4(f) depicts the PSNR plot, showing the PSNR achieved by different algorithms with different bits per pixel (b/p). This plot gives trends of the colorization quality under the same compression ratio. As we can see, TEM outperforms GRED and Cheng's method, and TEM-C is significantly better than the other algorithms.

We also include plots of some other well-known IQA in Fig. 4(g)–(i). Specifically, we show plot of VSNR in Fig. 4(g), MS-SSIM in Fig. 4(h), and NQM in Fig. 4(i). As we can see from the figures, the relative performance of the compared algorithms agrees on all the assessments.

For demonstration, we also depict the selected pixels of different algorithms in Fig. 5. Cheng's heuristic method selects very few pixels on the trees, which might be the main

(a) Cheng (b) GRED (c) TEM

Fig. 5. Demonstration of selected color pixels. Color pixels selected by (a) Cheng's method, (b) GRED, and (c) TEM.

reason for its reddish colorization there. GRED produced more balanced results. Comparing the TEM and GRED, we find that TEM selects more pixels on "difficult" regions like the left front of the yellow bus and the top of the red bus. This explains why TEM gives better colorization than GRED on those regions. Besides, TEM also selects fewer pixels on "easy" regions like the ground.

The second example is a photo of a man, as shown in Fig. 6(a). Fig. 6(b)–(e) shows the results with a compression ratio 55:1. Cheng's method only colorizes the grass and water. GRED is much better than Cheng's method, but gives diffused colors for the water near the left leg of the man. TEM reduces the diffusion significantly. TEM-C uses difference image to erase the diffusion effects completely and achieves a very high-quality result. The plots for PSNR, VSNR, MS-SSIM, and NQM are given in Fig. 6(f)–(i), which again show the superiority of TEM-C to previous algorithms.

Finally, we present experiments on several standard JPEG test images. The results for *Baboon*, *Lenna*, and *Pepper* are shown in Figs. 7, 8, and 9, respectively. Our algorithms (especially TEM-C) consistently outperform previous methods in all those examples.

C. Benchmark Evaluation

In this section, we evaluate the average performance of the given algorithms on a standard benchmark dataset. Specifically, we randomly pick 1000 images from the Corel dataset. Fig. 10 shows some sample images we use here. We evaluate Cheng's method, GRED, TEM, and TEM-C on each image and show the averaged PSNR plot in Fig. 11. Note that for TEM-C, the b/p coordinates of the PSNR curve for different images are not the same, so the PSNR plot shown in the figure is actually generated by averaging the piecewise interpolated PSNR curves of each image.

As we can see from the figure, GRED outperforms Cheng's method. TEM is better than GRED, especially when the number of selected points is small. TEM-C greatly improved the results of regression-based algorithms by achieving much higher PSNR values.

D. Comparison With Standard Techniques

As a reference, in this section, we compare our algorithm with the industrial standard JPEG and JPEG2000 codec. We

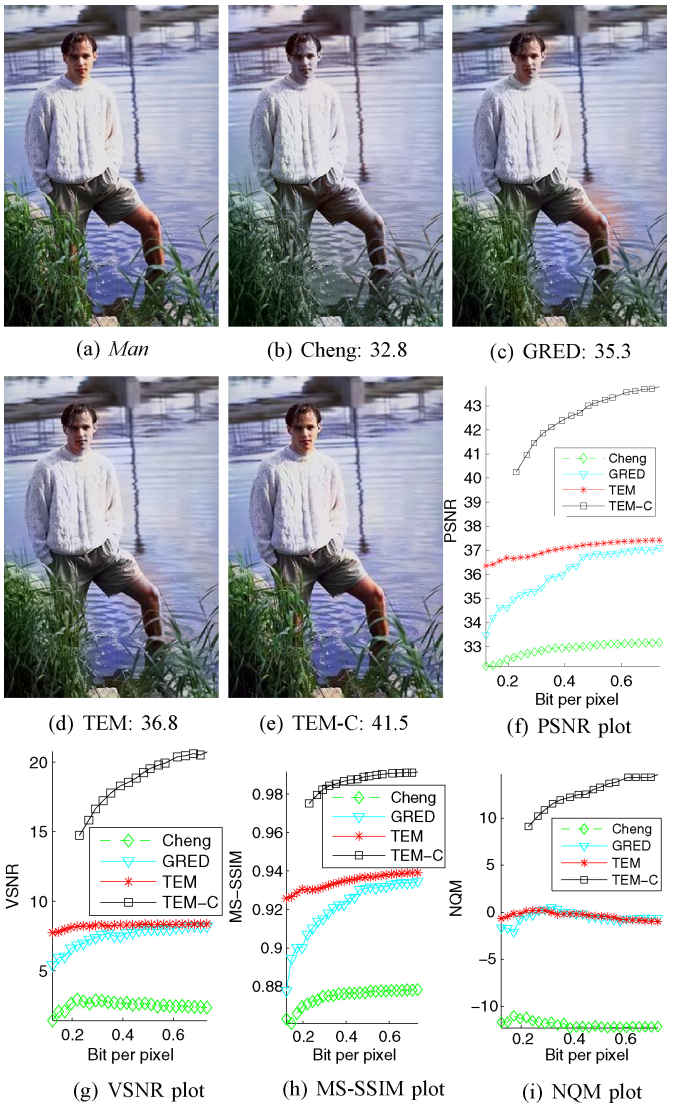


Fig. 6. Colorization result for the *Man* image. (a) Original image of size 256 × 384. (b)–(e) Colorization results at the compression ratio 55:1. (f) PSNR, (g) VSNR, (h) MS-SSIM, and (i) NQM value varying with the b/p for storing.

fix the PSNR and compare the compression ratio achieved by different algorithms. In order to have a fair comparison with JPEG and JPEG2000, we will take the luminance channel into consideration. Although JPEG/JPEG2000 can handle each channel independently, machine learning-based algorithms require information on the luminance channel to decode the chrominance channels. So in this section, PSNR is averaged over all channels. Cheng's method, GRED, and TEM cannot achieve a PSNR competitive to those of JPEG/JPEG2000 even after 1500 color pixels are selected, so we only compare TEM-C with JPEG and JPEG2000 here.

For JPEG and JPEG2000, we directly compress the RGB image and compute the average PSNR over the RGB channels. For TEM-C, we convert the colorization result back to RGB space before calculating the average PSNR. The compression ratio for TEM-C is calculated with a new formula as

$$R = \frac{N + |\mathrm{Y}|}{48l + D_{\mathrm{Cb}} + D_{\mathrm{Cr}} + |\mathrm{Y}|_c} \tag{30}$$

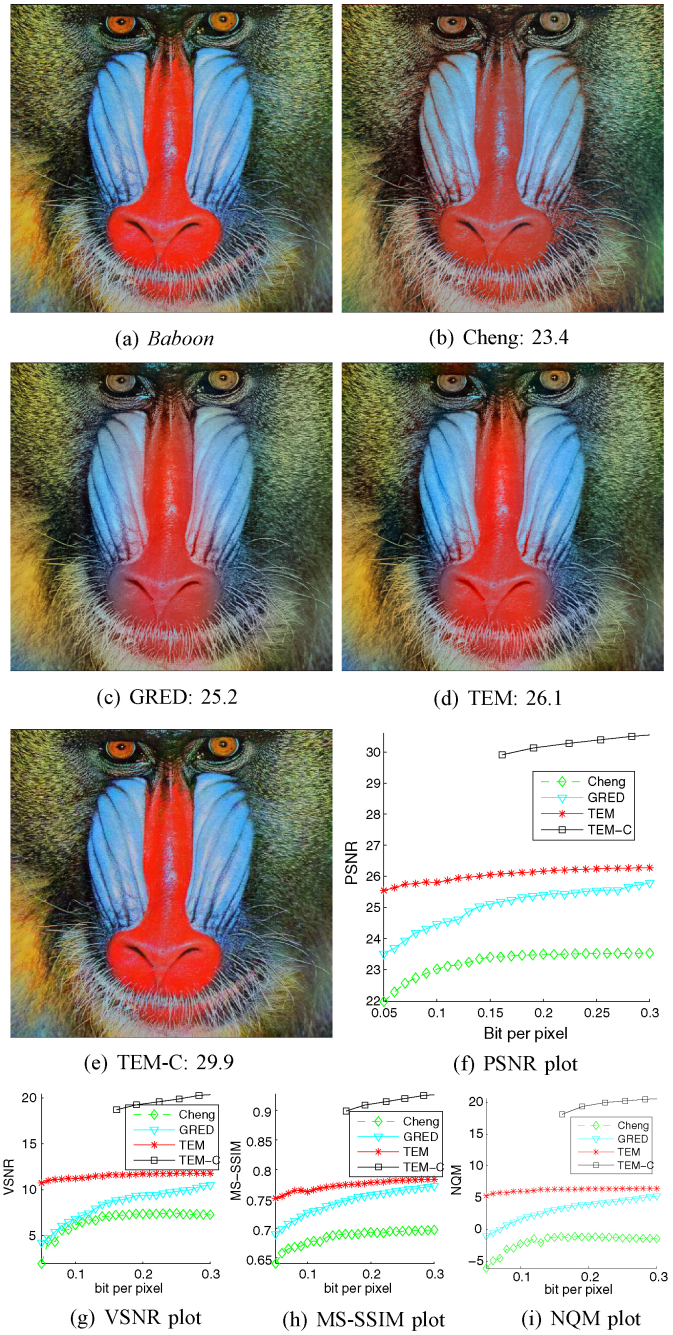


Fig. 7. Colorization result for the *Baboon* image. (a) Original image of size 500 × 480. (b)–(e) Colorization results at the compression ratio 100:1. (f) PSNR, (g) VSNR, (h) MS-SSIM, and (i) NQM value varying with the b/p for storing.

where we have added the number of bits of the luminance channel to the numerator and the compressed number of bits of it to the denominator. For JPEG and JPEG2000, the denominator is simply replaced by the number of bits of the compressed RGB image. We use the built-in implementation of JPEG and JPEG2000 of MATLAB for testing.

The results are shown in Table I. Quality factor (QF) in JPEG stands for the quality parameter for the MATLAB JPEG encoder, and QF for JPEG2000 is the compression ratio parameter for the MATLAB JPEG2000 encoder. As we can see, our TEM-C algorithm can be competitive with JPEG, and

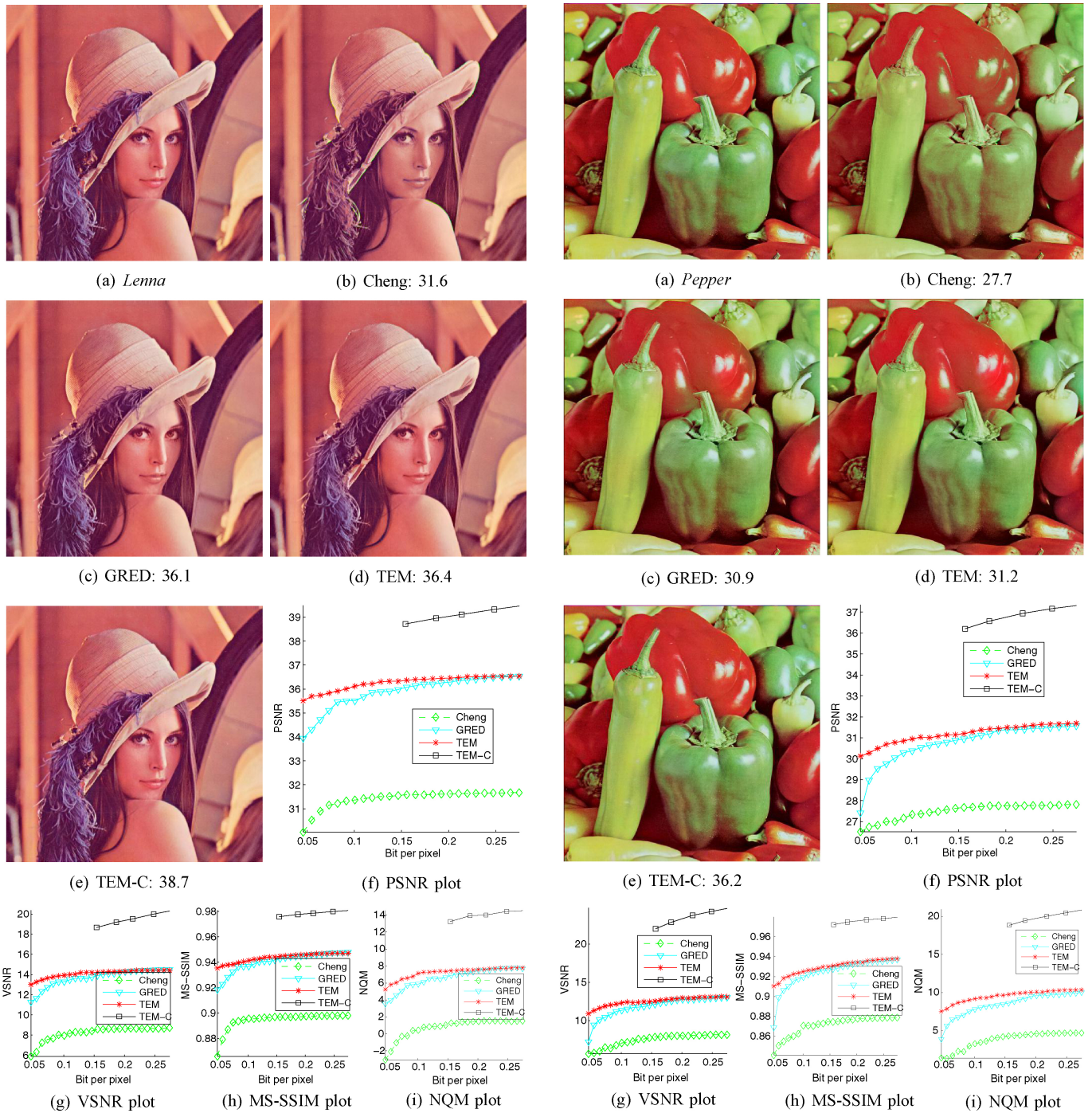


Fig. 8. Colorization result for the *Lenna* image. (a) Original image of size 512 × 512. (b)–(e) Colorization results at the compression ratio 103:1. (f) PSNR, (g) VSNR, (h) MS-SSIM, and (i) NQM value varying with the b/p for storing.

Fig. 9. Colorization result for the *Pepper* image. (a) Original image of size 512 × 512. (b)–(e) Colorization results at the compression ratio 103:1. (f) PSNR, (g) VSNR, (h) MS-SSIM, and (i) NQM value varying with the b/p for storing.

even outperforms it in the *Pepper* example. However, when comparing with JPEG2000, our performance is still not good enough.

E. Computation Efficiency Analysis

In this section, we study how the size of the kernel PCA map affects the performance and computation cost. For each given size of the kernel PCA map, we select 300 points with TEM and compute the PSNR and the average time (in seconds) for selecting one point. We also use GRED to select the same number of points and give comparisons of the results in Fig. 12. As depicted in Fig. 12(a), the computation time for TEM grows rapidly as m increases. On the other hand, in Fig. 12(b), we observe that the PSNR achieved by TEM soon reaches a relatively stable status as m increases. Therefore, with a moderately small m, we can use linear kernel to reduce the computation burden significantly, while still enjoying a good colorization quality.

Fig. 10. Sample images from the benchmark dataset we used. Each image is of size 384×256 or 256×384.

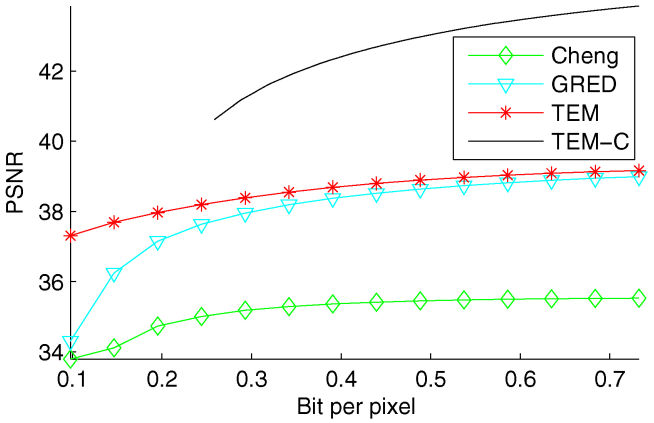


Fig. 11. PSNR plots on the benchmark image dataset.

TABLE I
COMPARISON ON COMPRESSION RATIO

Image	TEM-C		JPEG			JPEG2000		
	PSNR	c.r.	PSNR	c.r.	QF	PSNR	c.r.	QF
Bird	34.1	20.5	34.1	27.5	64	34.1	46.5	45
Bus	28.8	10.1	28.9	11.6	65	28.8	18.3	18
Man	33.4	13.4	33.4	13.7	73	33.4	22.8	22
Baboon	25.2	11.4	25.2	11.4	68	25.2	18.4	18
Lenna	33.0	23.7	33.0	25.3	66	33.0	50.2	49
Pepper	31.0	22.6	31.0	14.5	83	31.0	51.6	50

Specifically, we set the size of the kernel PCA map to 100 in all our experiments. TEM is still computationally more costly than GRED because it uses all the labels, but with this cost it achieves much higher PSNR.

Typically, all machine-learning-based algorithms will take more than an hour to encode an image (of similar sizes given in our experiments). The majority consumption of time is color pixel selection. The decoding is faster and can be done within several minutes. The high computation cost is the main factor of preventing the machine-learning-based methods from being useful in practice. However, advances in efficient machine learning algorithms and computing power of computers may help to change the situation.

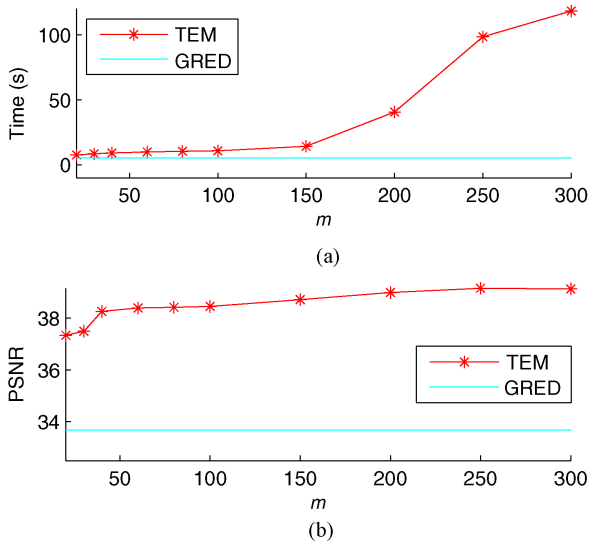


Fig. 12. Performance and computation time varying with the size of the kernel PCA map. (a) Average time consumption per point. (b) PSNR.

VII. CONCLUSION AND OUTLOOK

In this paper, we proposed the TEM algorithm and its improved version TEM-C for image compression. The key advantage over previous methods comes from the maximum exploitation of the full label set (i.e., the colors for all the pixels) at the encoding stage. Furthermore, TEM-C used the label set to generate and store a difference image for correcting the prediction error and improved the colorization quality significantly. Experimental results demonstrated the outstanding performance of the proposed methods.

Although the computation burden is still high, TEM-C is already competitive to the industrial standard JPEG in image quality and compression ratio. This depicts the potential impact of machine learning algorithms on image compression problems.

Actually, aside from the framework used here, there are other ways to exploit machine learning techniques. For example, in traditional image compression algorithms, the image is represented as a combination of basis in the frequency domain, and then the coefficients are quantized and compressed. Instead of using standard Fourier basis or wavelet basis, we can learn an optimal basis from the image data. This might not be suitable for a single image because extra storage is needed for the learned basis. But for video data, all the frames can share the same basis so that the extra storage is negligible.

APPENDIX

PROOF OF THEOREM 1

First, we use (21) to expand the left-hand side of (26)

$$\mathbf{w}^{*\top}\tilde{\mathbf{u}} = Y_X^\top \tilde{X}^\top (\tilde{X}\tilde{X}^\top + \lambda_1 I + \lambda_2 \tilde{U} L \tilde{U}^\top)^{-1}\tilde{\mathbf{u}}.$$

Before expanding the right-hand side, we note that when $Z = U$, we have

$$\tilde{\mathbf{u}} = \varphi_U(\mathbf{u}) = K^{-1/2}[\kappa(\mathbf{u}, \mathbf{u}_1), \ldots, \kappa(\mathbf{u}, \mathbf{u}_n)]^\top.$$

Hence, for $\tilde{U} = [\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_n]$ and $\tilde{X} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_l]$, we get

$$\begin{aligned}\tilde{U}^\top \tilde{X} &= [\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_n]^\top [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_l] \\ &= \begin{bmatrix} \kappa(\mathbf{u}_1, \mathbf{u}_1) & \ldots & \kappa(\mathbf{u}_1, \mathbf{u}_n) \\ \vdots & & \vdots \\ \kappa(\mathbf{u}_n, \mathbf{u}_1) & \ldots & \kappa(\mathbf{u}_n, \mathbf{u}_n) \end{bmatrix} K^{-1/2} \\ &\quad K^{-1/2} \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{u}_1) & \ldots & \kappa(\mathbf{x}_l, \mathbf{u}_1) \\ \vdots & & \vdots \\ \kappa(\mathbf{x}_1, \mathbf{u}_n) & \ldots & \kappa(\mathbf{x}_l, \mathbf{u}_n) \end{bmatrix} \\ &= K K^{-1} K_{\mathrm{UX}} = K_{\mathrm{UX}}.\end{aligned}$$

Similarly, we have $\tilde{U}^\top \tilde{U} = K$. Since the kernel gram matrix K is positive definite, we actually have $\tilde{U} = K^{1/2}$, which is also positive definite. Then, by the definition of the kernel PCA map, the right-hand side of (26) can be written as

$$\sum_{i=1}^{n} \alpha_i^* \kappa(\mathbf{u}, \mathbf{u}_i) = \boldsymbol{\alpha}^{*\top} \begin{bmatrix} \kappa(\mathbf{u}, \mathbf{u}_1) \\ \vdots \\ \kappa(\mathbf{u}, \mathbf{u}_n) \end{bmatrix} = \boldsymbol{\alpha}^{*\top} K^{1/2} \tilde{\mathbf{u}} = \boldsymbol{\alpha}^{*\top} \tilde{U} \tilde{\mathbf{u}}.$$

Expand $\boldsymbol{\alpha}^*$ by (9). Further note that $\tilde{U}^\top = \tilde{U}$ and the positive definity of $\tilde{U}$ guarantees the existence of $\tilde{U}^{-1}$, so we have

$$\begin{aligned}&\boldsymbol{\alpha}^{*\top} \tilde{U} \tilde{\mathbf{u}} \\ &= Y_X^\top K_{\mathrm{UX}}^\top (K_{\mathrm{UX}} K_{\mathrm{UX}}^\top + \lambda_1 K + \lambda_2 K L K)^{-1} \tilde{U} \tilde{\mathbf{u}} \\ &= Y_X^\top \tilde{X}^\top \tilde{U} (\tilde{U}^\top \tilde{X} \tilde{X}^\top \tilde{U} + \lambda_1 \tilde{U}^\top \tilde{U} + \lambda_2 \tilde{U}^\top \tilde{U} L \tilde{U}^\top \tilde{U})^{-1} \tilde{U} \tilde{\mathbf{u}} \\ &= Y_X^\top \tilde{X}^\top (\tilde{X} \tilde{X}^\top + \lambda_1 I + \lambda_2 \tilde{U} L \tilde{U}^\top)^{-1} \tilde{\mathbf{u}}\end{aligned}$$

which is the same as the expansion of the left-hand side of (26).

REFERENCES

[1] S. Saha and R. Vemuri, "Adaptive wavelet coding of multimedia images," in *Proc. ACM Multimedia*, 1999, pp. 71–74.

[2] M. Charrier, D. S. Cruz, and M. Larsson, "JPEG2000, the next millennium compression standard for still images," in *Proc. IEEE Int. Conf. Multimedia Comput. Syst. (ICMCS)*, vol. 1. Jun. 1999, pp. 131–132.

[3] H.-C. Fang, Y.-W. Chang, T.-C. Wang, C.-T. Huang, and L.-G. Chen, "High-performance JPEG2000 encoder with rate-distortion optimization," *IEEE Trans. Multimedia*, vol. 8, no. 4, pp. 645–653, Apr. 2006.

[4] O. T.-C. Chen and C.-C. Chen, "Automatically-determined region of interest in JPEG2000," *IEEE Trans. Multimedia*, vol. 9, no. 7, pp. 1333–1345, Jul. 2007.

[5] H. Persson, A. Brunstrom, and T. Ottosson, "Utilizing cross-layer information to improve performance in JPEG2000 decoding," *Adv. Multimedia*, vol. 2007, no. 24758, pp. 1–10, 2007.

[6] N. T. N. Anh, W. Yang, and J. Cai, "Seam carving extension: A compression perspective," in *Proc. ACM Multimedia*, 2009, pp. 825–828.

[7] L. Cheng and S. V. N. Vishwanathan, "Learning to compress images and videos," in *Proc. Int. Conf. Mach. Learn.*, 2007, pp. 161–168.

[8] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM SIGGRAPH*, vol. 23, no. 3, pp. 689–694, 2004.

[9] K. I. Kim, F. Steinke, and M. Hein, "Semi-supervised regression using Hessian energy with an application to semi-supervised dimensionality reduction," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 22. 2009, pp. 979–987.

[10] B. Lin, C. Zhang, and X. He, "Semi-supervised regression via parallel field regularization," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 24. 2011, pp. 433–441.

[11] X. He, M. Ji, and H. Bao, "A unified active and semi-supervised learning framework for image compression," in *Proc. CVPR*, 2009, pp. 65–72.

[12] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, vol. 7, pp. 2399–2434, Nov. 2006.

[13] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computat.*, vol. 15, no. 6, pp. 1373–1396, 2003.

[14] D. Cai, X. He, and J. Han, "Using graph model for face analysis," Dept. Comput. Sci., Univ. Illinois Urbana-Champaign, Urbana, Tech. Rep. UIUCDCS-R-2005-2636, Sep. 2005.

[15] D. Cai, X. He, and J. Han, "Spectral regression for dimensionality reduction," Dept. Comput. Sci., Univ. Illinois Urbana-Champaign, Urbana, Tech. Rep. UIUCDCS-R-2007-2856, May 2007.

[16] D. Cai, X. He, W. V. Zhang, and J. Han, "Regularized locality preserving indexing via spectral regression," in *Proc. CIKM*, 2007, pp. 741–750.

[17] J. Zhao, K. Lu, and X. He, "Locality sensitive semi-supervised feature selection," *Neurocomputing*, vol. 71, nos. 10–12, pp. 1842–1849, 2008.

[18] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[19] A. Atkinson, A. Donev, and R. Tobias, *Optimum Experimental Designs With SAS* (Series Oxford Statistical Science). Oxford, U.K.: Oxford Univ. Press, 2007.

[20] K. Yu, S. Zhu, W. Xu, and Y. Gong, "Non-greedy active learning for text categorization using convex transductive experimental design," in *Proc. SIGIR*, 2008, pp. 635–642.

[21] L. Zhang, C. Chen, W. Chen, J. Bu, D. Cai, and X. He, "Convex experimental design using manifold structure for image retrieval," in *Proc. ACM Multimedia*, 2009, pp. 45–54.

[22] F. R. K. Chung, *Spectral Graph Theory*. Providence, RI: Am. Math. Soc., 1997.

[23] M. S. Bartlett, "An inverse matrix adjustment arising in discriminant analysis," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 107–111, 1951.

[24] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2001.

[25] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola, "Input space versus feature space in kernel-based methods," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 1000–1017, May 1999.

[26] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. New York: Cambridge Univ. Press, 2002.

[27] X. Ren and J. Malik, "Learning a classification model for segmentation," in *Proc. Int. Conf. Comput. Vision*, 2003, pp. 10–17.

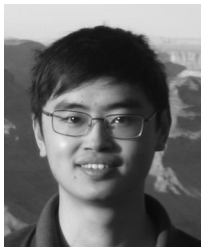
[28] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[29] D. M. Chandler and S. S. Hemami, "VSNR: A wavelet-based visual signal-to-noise ratio for natural images," *IEEE Trans. Image Process.*, vol. 16, no. 9, pp. 2284–2298, Sep. 2007.

[30] N. Ponomarenko, F. Battisti, K. Egiazarian, J. Astola, and V. Lukin, "Metrics performance comparison for color image database," presented at the 4th International Workshop on Video Processing and Quality Metrics for Consumer Electronics, Scottsdale, AZ, Jan. 2009.

Chiyuan Zhang received the B.S. degree in computer science from Zhejiang University, Zhejiang, China, in 2009, where he is currently pursuing the Masters degree in computer science.

His current research interests include machine learning and computer vision.



Xiaofei He (SM'10) received the B.S. degree in computer science from Zhejiang University, Zhejiang, China, in 2000, and the Ph.D. degree in computer science from the University of Chicago, Chicago, IL, in 2005.

He is currently a Professor with the State Key Laboratory of Computer Aided Design and Computer Graphics, Zhejiang University. He was a Research Scientist with Yahoo! Research Laboratories, Burbank, CA. His current research interests include machine learning, information retrieval, and computer vision.