FAST IMAGE VECTOR QUANTIZATION USING SPARSE OBLIQUE REGRESSION TREES

Rasul Kairgeldin *Miguel Á. Carreira-Perpiñán*

Dept. Computer Science & Engineering, University of California, Merced. Merced, CA, USA

ABSTRACT

Vector quantization is a fundamental model for image coding, but large codebooks require a long training and encoding time. This can be sped up with tree-structured codes. We propose the use of sparse oblique decision trees, which have hyperplane splits with few nonzero weights in the decision nodes. Such trees can be trained on a dataset of image patches using tree alternating optimization. Experimentally with different datasets, we show these trees consistently achieve a low distortion, close to that of a flat codebook, and a much faster encoding, exceeding other tree-structured vector quantizers.

***Index Terms*—** Vector quantization, tree-structured codes, oblique regression trees, tree alternating optimization.

1. INTRODUCTION

Vector quantization (VQ) is a fundamental method for image coding, where a patch in an image (considered as a vector in a Euclidean space) is approximately represented by one of a finite set of K vectors (the codebook). The usual way to determine a good codebook is by optimizing a loss function, typically the squared distortion (squared Euclidean distance), over a large collection of patches extracted from an image dataset. This is commonly optimized by the k-means algorithm, which monotonically decreases the loss by alternately updating the codebook vectors and the set of patches assigned to each codebook vector. With a large, diverse image dataset and a large enough number of codebook vectors K, the distortion can be reduced as desired. However, the cost both of training and encoding a test patch is proportional to K, since we have to find the closest codebook vector to the test vector (by computing K distances).

A convenient way to speed this up is by using a tree-structured codebook having one codebook vector in each leaf. Encoding requires traversing a single root-leaf path, rather than scanning all K leaves, which has a logarithmic rather than linear cost on K. This has two critical difficulties. One is *how to define the decision nodes of the tree*, so that the partition induced in the patch vector space is flexible enough to model complex data distributions, but the computational cost remains low. The other is *how to learn such a tree from data*, which is a nonconvex, nondifferentiable optimization problem.

As tree model, we use a *sparse oblique regression tree*. This is a binary, complete tree of depth Δ, where each decision node uses a sparse hyperplane split (with few nonzero weights) to route the patch down its left or right child. Each leaf contains a constant, codebook vector. This partitions the space into convex polytopes, each defined by the intersection of the halfspaces in the root-leaf path, and is quite different from the Voronoi partition induced by regular VQ.

To train this tree (i.e., the hyperplane parameters and the codebook vectors) we rely on a recent algorithm, *Tree Alternating Optimization (TAO)* [1, 2], in what we think is its first application to VQ. To do this, we reformulate the usual VQ squared distortion over a finite codebook with flat assignment variables so that the codebook is defined by the tree output, i.e., its 2^{Δ} leaves. This means the assignments have a hierarchical structure, implicit in the decision nodes.

We evaluate our method on several image datasets and compare it with k-means and several tree-structured VQ methods, showing significant improvements in terms of distortion (which remains close to that of a flat, unconstrained codebook) and encoding time.

2. RELATED WORK

The traditional way to learn regression trees is based on greedy recursive partitioning algorithms such as CART [3] or C5.0 [4], typically using axis-aligned decision nodes (which split based on a single input feature). These algorithms do not optimize a loss over the whole tree and they produce overly large, yet inaccurate trees, since axis-parallel partitions are a poor model for high-dimensional data. The TAO algorithm [1] monotonically decreases an arbitrary objective function over an arbitrary parametric tree model (such as axis-aligned or oblique) by iteratively optimizing over one node at a time. This produces trees that are much smaller and accurate. We use here the regression verison of TAO [2].

Tree-structured vector quantization (TSVQ) constructs a codebook by hierarchically partitioning the multi-dimensional feature space [5, 6, 7]. The tree structure enables significantly faster encoding via a root-to-leaf traversal. For a sufficiently balanced tree, encoding complexity scales logarithmically with the number of leaves. TSVQ is typically constructed using a heuristic greedy recursive partitioning approach, where splits (either oblique or axis-aligned) are chosen based on some predefined metrics [8, 9, 10] or randomly [11]. The tree is grown until some stopping criteria is met and then pruned to achieve optimal distortion-rate trade-off [12, 13].In image processing, segmentation-based coding is a common technique where an image is partitioned into regions, and quantization is applied to each segment separately. This approach is especially effective for image and medical image compression, enabling adaptive encoding based on local characteristics [14, 13, 5, 10].

Tree-structured codebooks promise a logarithmic rather than linear speedup over a flat codebook, but for this to work well the tree should be balanced and trained to optimize the distortion.

3. VECTOR QUANTIZATION WITH SPARSE OBLIQUE DECISION TREES

3.1. Sparse oblique decision trees

We consider a binary tree of depth Δ with a set of decision nodes $\mathcal{D}$ and leaf nodes $\mathcal{L}$. Each decision node $i \in \mathcal{D}$ contains a routing function $g_i(\mathbf{x};\boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \{\texttt{left}_i, \texttt{right}_i\} \subset \{\mathcal{D} \cup \mathcal{L}\}$. We use linear routing function $g_i(\mathbf{x};\boldsymbol{\theta}_i) = \texttt{left}_i$ if $\mathbf{w}_i^T\mathbf{x} + w_{0i} < 0$, otherwise $\texttt{right}_i$, where $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{0i}\}$ is learnable. Each leaf

Work partially supported by NSF award IIS–2007147.

node $j \in \mathcal{L}$ contains codebook vector $\boldsymbol{\mu}_j$. The tree's parameters are $\boldsymbol{\Theta} = \{(\mathbf{w}_i, w_{0i})\}_{i \in \mathcal{D}} \cup \{\boldsymbol{\mu}_j\}_{j \in \mathcal{L}}$ and a tree routing function is $\mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})$ that directs a patch $\mathbf{x}_n$ from a root to a single leaf and predicts a corresponding codeword $\boldsymbol{\mu}_j$.

3.2. Tree-structured vector quantization

We formulate the tree-structured VQ problem as follows:

$$\min_{\boldsymbol{\Theta}} \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})\|^2 + \lambda \sum_{i \in \mathcal{D}} \|\mathbf{w}_i\|_1 \qquad (1)$$

where the ℓ_1 regularization term promotes sparsity in the decision node hyperplanes, which brings up computational savings in memory and time. Problem (1) can be seen as generalizing the regular squared distortion over a dataset of patches $\{\mathbf{x}_n\}_{n=1}^N \subset \mathbb{R}^D$ from a flat codebook to a hierarchical one. We do optimize the same loss (left term in (1)) and we do have a codebook (the constant vectors $\{\boldsymbol{\mu}_j\}_{j \in \mathcal{L}}$ at the leaves of the tree $\mathbf{T}$). But the assignment of a patch $\mathbf{x}_n$ to a codebook vector $\boldsymbol{\mu}_j$ is not anymore a free, independent variable (as in k-means); it is implicit in the tree structure and decision nodes. Thus, the region of patch space that is assigned to a codebook vector is not a Voronoi cell, but a trainable polytope.

In tree-structured vector quantization, the root-to-leaf path determines the codeword by sequentially partitioning the input space through decision nodes until a specific leaf is reached. The codebook size is given by the number of leaves $\mathcal{L}$. A sparsity parameter λ controls the weight sparsity and, if large enough, can prune the tree; thus, the codebook size (hence, the code rate) is primarily controlled by Δ and secondarily by λ (hyperparameters). The total number of parameters in the vector quantizer is at most twice the codebook size (covering both decision and leaf parameters) but is often lower due to sparsity in decision nodes.

3.3. Learning the sparse oblique tree with the TAO algorithm

From a machine learning point of view, problem (1) can also be seen simply as fitting a regression tree to a dataset where the inputs equal the outputs. We can use the recently proposed Tree Alternating Optimization (TAO) algorithm to find a local optimum. For a fixed-structure oblique tree $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ with initial parameters (random or one produced by greedy-recursive partitioning algorithms), TAO directly optimizes the objective function (1). TAO relies on two key theorems. First, the *separability condition* states that the objective can be optimized independently and in parallel over non-descendant nodes (of the same depth), a result of hard decision boundaries. Second, the *reduced problem over a node* ensures that optimizing the objective for a given node i simplifies to a well-defined problem involving only the node's parameters and the subset of patches $\mathcal{R}_i$ that reach it. For a decision node $i \in \mathcal{D}$ problem (1) is reduced to a weighted $0/1$ loss binary classification problem:

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} \overline{L}(\overline{y}_n, g_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \lambda \|\mathbf{w}_i\|_1 \qquad (2)$$

Here, $\overline{y}_n \in \{\texttt{left}_i, \texttt{right}_i\}$ is a pseudolabel indicating the 'best' child that minimizes loss for $\mathbf{x}_n$ by passing it to left and right subtree of a node i, and $\overline{L}(\cdot,\cdot)$ is a weighted 0/1 loss based on the loss difference between the two children. Optimizing an oblique node is generally NP-hard but can be efficiently approximated using a surrogate loss, such as cross-entropy (logistic regression). The top-level objective (1) is guaranteed to decrease monotonically by accepting updates only if they improve (2), though this is rarely needed in practice [2].

Algorithm 1: Learning a tree-structured vector quantizer with TAO

```
input training set $\{\mathbf{x}_n\}_{n=1}^N$;
initial tree $\mathbf{T}(\cdot; \boldsymbol{\Theta})$ of depth $\Delta$;
for depth $d = 0$ to $\Delta$ do
    for $i \in$ nodes at depth $d$ do
        if $i$ is a leaf then
            $\boldsymbol{\mu}_i \leftarrow$ fit a constant regressor at a leaf eq. (3)
             with patches in $\mathcal{R}_i$ as targets;
        else
            $\boldsymbol{\theta}_i \leftarrow$ fit a weighted binary classifier (eq. (2));
        end
    end
end
return trained tree $\mathbf{T}$
```

For a leaf $j \in \mathcal{L}$ the problem (1) is reduced to an original loss (squared distortion) between the codeword of a leaf μ_j and the leaf's reduced set $\mathcal{R}_j$. It can be solved by finding a mean (similar to k-means):

$$\min_{\boldsymbol{\mu}_j} \sum_{n \in \mathcal{R}_j} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \qquad (3)$$

When applied to problem (1), TAO is reminiscent of k-means: the update of the codebook vectors is identical, but the update of the assignment variables (in k-means) and the decision node parameters (TAO) is very different.

3.4. Computational complexity: training and encoding

For a dataset of patches $\{\mathbf{x}_n\}_{n=1}^N \subset \mathbb{R}^D$, a complete tree of depth Δ with K leaves has a root node optimization complexity of $\mathcal{O}(\Delta DN) + \mathcal{O}(cDN)$(pseudolabel assignment and binary classifier fitting with c iterations). Decision nodes at each level Δ_i can be optimized in parallel within $\mathcal{O}(\Delta_i DN) + \mathcal{O}(cDN)$, leading to a total complexity of at most $\mathcal{O}(\Delta^2 DN) + \mathcal{O}(c\Delta DN)$. In practice, warm-starting logistic regression with previous TAO weights significantly reduces training time. For large trees with a large codebook size K, decision node training is approximately $\mathcal{O}(DN \log^2 K)$, much faster than k-means' $\mathcal{O}(DNK)$. Leaf optimization matches k-means' centroid step at $\mathcal{O}(ND)$.

Test patch encoding is $\mathcal{O}(D \log K)$ for a tree-structured vector quantizer (root-to-leaf path) versus $\mathcal{O}(DK)$ for k-means. Thus, both the training and the encoding benefit from a logarithmic speedup over k-means, which is more important with large K (as needed for low distortion).

4. EXPERIMENTS

Experimental results demonstrate that our proposed approach consistently outperforms other methods (RP-tree [11], PCA-tree [9] and k-means) in terms of encoding time. Furthermore, it achieves much lower distortion for a given bit rate compared to other tree-based methods, we demonstrate it on 5 RGB image datasets: Kodak, The Oxford-IIIT Pet, Oxford 102 Flower, Urban100, Describable Textures Dataset (DTD). For larger datasets we selected a subset of images. Additionally, we demonstrate improved image quality of proposed quantization approach for different bit rates and different patch sizes.

	K		ours	RP-tree	PCA-tree	k-means
Oxford Pet (140839,768)	$(128, 128, 127)$	$E \times 10^{-3}$	**4.11**	7.84	4.82	3.90
		Δ	7/7/7	7/7/7	7/7/7	-/-/-
		MFLOPs	**3.62**	7.86	7.86	216
		Δ^*	7/7/7	14/14/15	9/9/9	-/-/-
		MFLOPs*	3.62	15.7	9.83	180
	$(1.8k, 1.6k, 1.8k)$	$E \times 10^{-3}$	**2.94**	5.91	3.47	2.64
		Δ	11/11/11	11/11/11	11/11/11	-/-/-
		MFLOPs	**3.25**	12.0	12.0	2930
		Δ^*	11/11/11	16/16/16	13/13/13	-/-/-
		MFLOPs*	3.25	17.3	14.1	2440
Oxford Flowers (147494,192)	$(64, 64, 64)$	$E \times 10^{-3}$	**4.01**	8.00	5.22	3.55
		Δ	6/6/6	6/6/6	6/6/6	-/-/-
		MFLOPs	**8.16**	9.84	9.84	159
		Δ^*	6/6/6	12/11/11	8/8/8	-/-/-
		MFLOPs*	8.16	17.7	12.5	132
	$(3.3k, 3.5k, 3.5k)$	$E \times 10^{-3}$	**1.41**	3.51	2.02	1.05
		Δ	12/12/12	12/12/12	12/12/12	-/-/-
		MFLOPs	**17.0**	19.2	19.2	8450
		Δ^*	12/12/12	16/16/16	14/14/14	-/-/-
		MFLOPs*	1.70	2.58	2.25	704

	K		ours	RP-tree	PCA-tree	k-means
Urban (148264,300)	$(128, 128, 128)$	$E \times 10^{-2}$	**1.39**	2.18	1.54	1.29
		Δ	7/7/7	7/7/7	7/7/7	-/-/-
		MFLOPs	**5.59**	7.74	7.74	213
		Δ^*	7/7/7	14/14/14	9/9/9	-/-/-
		MFLOPs*	5.59	14.6	9.10	178
	$(3.8k, 3.7k, 3.6k)$	$E \times 10^{-3}$	**8.35**	16.1	10.1	7.06
		Δ	12/12/12	12/12/12	12/12/12	-/-/-
		MFLOPs	**7.48**	13.1	13.1	6190
		Δ^*	12/12/12	16/16/16	14/14/14	-/-/-
		MFLOPs*	7.48	17.2	14.8	5100
Dtd (170555,300)	$(128, 128, 128)$	$E \times 10^{-3}$	**9.10**	15.5	9.99	8.49
		Δ	7/7/7	7/7/7	7/7/7	-/-/-
		MFLOPs	**3.61**	7.27	7.27	200
		Δ^*	7/7/7	14/14/14	9/9/9	-/-/-
		MFLOPs*	3.61	13.7	8.41	157
	$(3.4k, 3.9k, 3.9k)$	$E \times 10^{-3}$	**5.56**	10.2	6.55	4.83
		Δ	12/12/12	12/12/12	12/12/12	-/-/-
		MFLOPs	**6.14**	12.3	12.3	5940
		Δ^*	12/12/12	16/16/16	14/14/14	-/-/-
		MFLOPs*	6.14	16.3	14.0	4600

Table 1. Comparison of quantization methods on 4 RGB image datasets. Each dataset is represented by (N, D), where N is the sample count and D size of flattened patch. We run experiments with varying codebook sizes per channel (K_R, K_G, K_B), reporting MSE, MFLOPs, and tree depth Δ per channel. MFLOPs* and Δ^* denote metrics needed for other methods to match our distortion. **Bold** marks the best results.

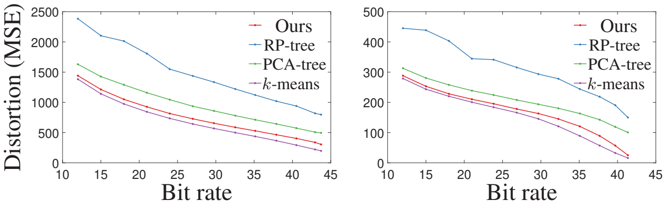

Fig. 1. Distortion (MSE) vs bit rate (Rate-Distortion curve) for different patch size (from left to right 5×5 and 15×15) of each method (TAO-tree, k-means, PCA-tree and RP-tree) on Kodak dataset.

We implement all methods in Python 3.8. To train linear classifier in decision nodes of our method and k-means, we used `scikit-learn`'s implementation [15] with LIBLINEAR [16]. Our algorithm starts with a complete tree of depth Δ and random median splits at decision nodes. All VQ algorithms minimize squared distortion on $P \times P$ image patches extracted from collection of images. Following a multi-codebook quantization approach, we train three quantizers (one per RGB channel), similar to product quantization [17], and concatenate the quantized outputs. Typically k-means achieves equal or lower distortion (ignoring cases of poor local optima) due to unconstrained assignments but has the slowest encoding time.

4.1. Encoding time

In this study, we assessed inference (encoding) time using the Kodak dataset. Images were divided into patches of different sizes (5×5, 10×10 and 15×15) and encoded. Measurements were averaged across all images in the dataset. We report two metrics per image: the actual encoding time, recorded using Python's `time.perf_counter` and averaged over five runs, and the number of floating point operations (FLOPs).

Figure 2 shows that trees trained with our proposed approach consistently has much faster encoding compared to all other methods. First two columns of fig. 2 shows the comparison to k-means. The computational complexity of k-means encoding is $\mathcal{O}(KD)$, while our trees operate at $\mathcal{O}(sD \log K)$, where s represents a sparsity coefficient. As the codebook size K increases, the difference in encoding time between our trees and k-means becomes substantial, exceeding 100 times faster for $K \approx 20000$ and over 1000 times faster in terms of FLOPs.

The last two columns of Figure 2 compare our method to other tree-based models, showing that our trees are up to 30% faster in encoding time and up to three times faster in FLOPs. This significant speed-up is primarily due to two factors: proper optimization and sparsity. Due to proper optimization our trees achieve the same distortion with much smaller depth and number of leaves. Therefore, in both PCA-trees and RP-trees samples need to travel much bigger root-to-leaf path. Additionally, our approach enables the learning of sparse oblique trees, which significantly decreases the number of parameters in decision nodes. Notably, the smaller gap in encoding time between our method and others is partly attributed to the use of vectorization in NumPy, which enhances vector operations but does not fully leverage sparsity. Incorporating proper sparse vectorization in our quantization method could potentially yield even greater improvements in encoding time.

4.2. Quantitative analysis

Table 1 demonstrate comparison of various methods in terms of distortion (MSE), tree depth Δ, and MFLOPs for different codebook

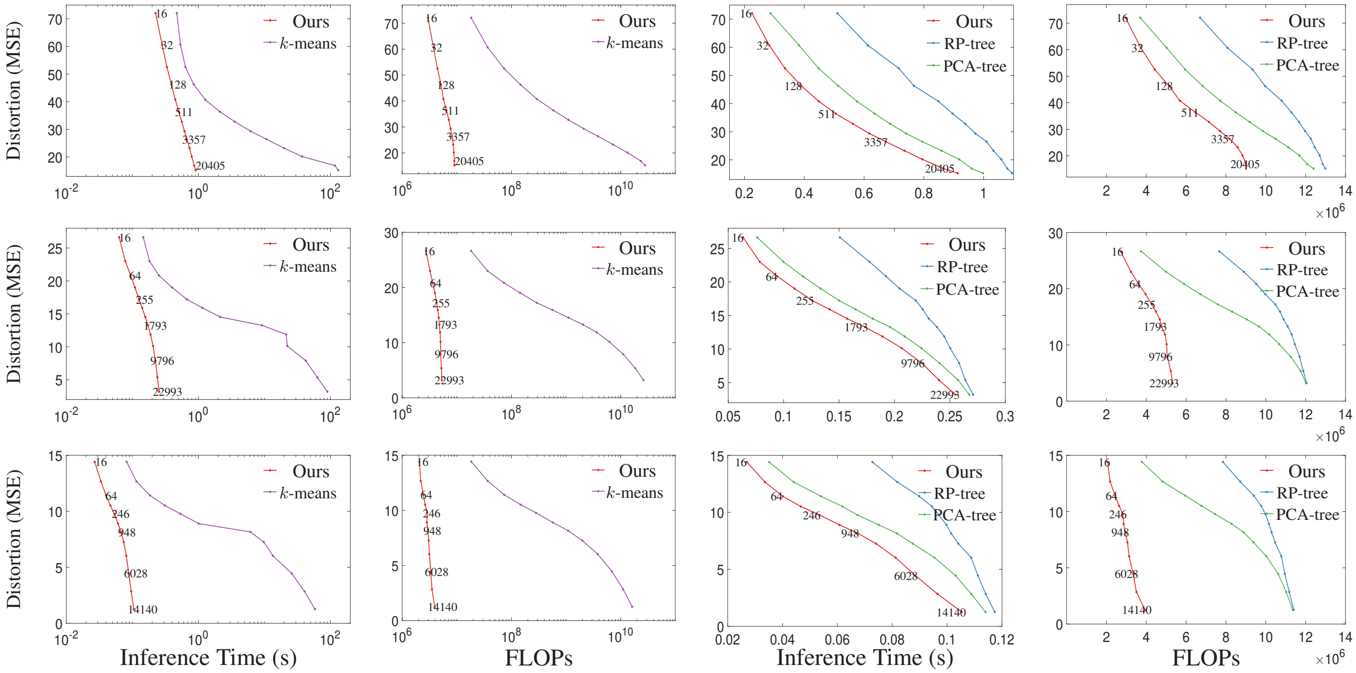

Fig. 2. Dependence between average inference time (encoding time) and floating-point operations of each method (TAO-tree, k-means, PCA-tree and RP-tree) and distortion for 1 image from Kodak dataset for different patch size. We show average codebook size per channel of our trees. First row patch size 5×5, second –10×10, third – 15×15.

size across 4 datasets. Across all datasets without exception our trees achieve significantly lower distortion compared to other tree-based methods – up to 30% lower compared to PCA-tree and 2 times lower compared to RP-tree – at both low and high bit rates. Furthermore, to match our performance, both greedy algorithms must grow much deeper trees, further widening the MFLOPs gap. While k-means, being unrestricted by a hierarchical structure, can achieve lower distortion, it does so at the expense of much higher MFLOPs, sometimes up to 1000 times more.

4.3. Dependence on bit rates and patch sizes

We perform multiple experiments to evaluate and compare algorithms across various bit rates and patch sizes, focusing on distortion and image quality metrics. Figure 1 illustrates the relationship between distortion and bit rate (Rate-Distortion curve) for various quantization algorithms for different patch sizes. Each curve represents the performance of a different algorithm, showing how distortion decreases as the bit rate increases. The figure highlights that our proposed method consistently achieves lower distortion at comparable bit rates, indicating more efficient quantization. At higher bit rates, the Rate-Distortion curve of our method approaches that of k-means.

Figure 3 illustrates the quality of quantized images generated by various methods across different bit rates. The results clearly show that the RP-tree struggles to effectively learn the codebook. Consequently, the reconstructed images shows significant noise, with incorrect color reconstruction at lower bit rates, rendering the boat in the foreground unrecognizable. At higher bit rates, it manages to capture some details, such as portions of the sky's gradient and the general outline of the boat. However, the resulting image remains significantly affected by noise. While the PCA-tree performed notably better in reducing noise, it still fails to capture the diversity of patches effectively. As a result, the reconstructed images lack detail and are affected by the gridding effect, characterized by blocky artifacts and a loss of fine details (it is noticeable in the sky). In contrast, our trees successfully captured the majority of details in the quantized images. They consistently achieve results closest to k-means across all bit rates in terms of both distortion and image quality.

Finally, Figure 4 illustrates the impact of patch size on image quality. As the patch size increases, the image becomes more susceptible to blocking artifacts. Larger patch sizes require significantly more bits to capture the details of various patches because the algorithms encode each patch as a single codeword. In contrast, smaller patch sizes (e.g., 5×5) allow both algorithms to capture finer details and result in better image reconstruction. However, as shown in Figure 2, the primary advantage of using a larger patch size is the faster encoding.

5. CONCLUSION

A good tree encoder for image patches should be powerful enough to define partitions of input space that are flexible enough to achieve low distortion while allowing for fast decoding, and be adaptive so it can be learned automatically from image datasets. We have proposed to achieve this with a new type of vector quantizer, sparse oblique regression trees, trained with the tree alternating optimization algorithm. The rate-distortion performance of our oblique tree codes is better than that of previous tree-structured codes and close to that of a flat codebook, but with a much faster decoding.

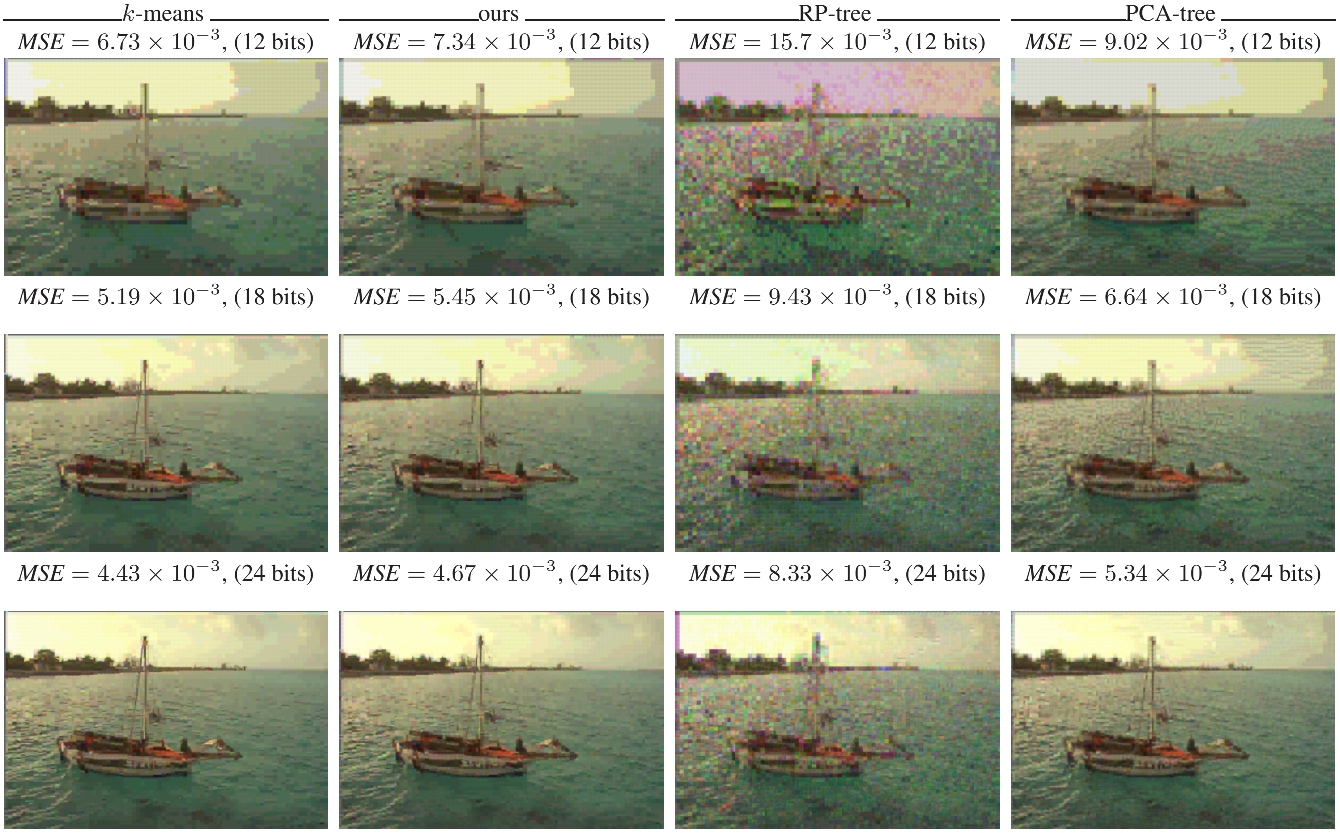
k-means
ours
RP-tree
PCA-tree
$MSE = 6.73 \times 10^{-3}$, (12 bits)
$MSE = 7.34 \times 10^{-3}$, (12 bits)
$MSE = 15.7 \times 10^{-3}$, (12 bits)
$MSE = 9.02 \times 10^{-3}$, (12 bits)
$MSE = 5.19 \times 10^{-3}$, (18 bits)
$MSE = 5.45 \times 10^{-3}$, (18 bits)
$MSE = 9.43 \times 10^{-3}$, (18 bits)
$MSE = 6.64 \times 10^{-3}$, (18 bits)
$MSE = 4.43 \times 10^{-3}$, (24 bits)
$MSE = 4.67 \times 10^{-3}$, (24 bits)
$MSE = 8.33 \times 10^{-3}$, (24 bits)
$MSE = 5.34 \times 10^{-3}$, (24 bits)

Fig. 3. Quantization quality with 10×10 patch size produced by each method (k-means, ours, RP-tree, PCA-tree) for different bit rates of the image from Kodak dataset. Distortion (MSE) and bit rate is on top of each image.

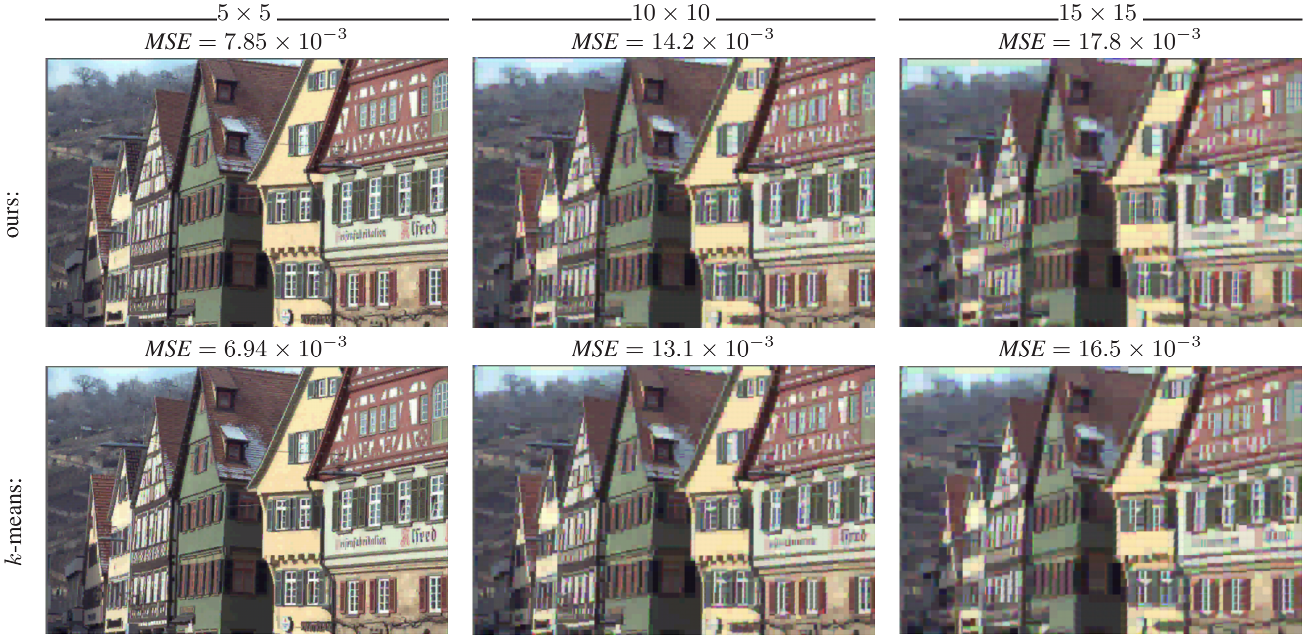
5×5
10×10
15×15
ours:
$MSE = 7.85 \times 10^{-3}$
$MSE = 14.2 \times 10^{-3}$
$MSE = 17.8 \times 10^{-3}$
k-means:
$MSE = 6.94 \times 10^{-3}$
$MSE = 13.1 \times 10^{-3}$
$MSE = 16.5 \times 10^{-3}$

Fig. 4. Quantization quality for different patch sizes produced by k-means and our method for 21 bit encoding of the image from Kodak.

6. REFERENCES

[1] Miguel Á. Carreira-Perpiñán and Pooya Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *Advances in Neural Information Processing Systems (NeurIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. 2018, vol. 31, pp. 1211–1221, MIT Press, Cambridge, MA.

[2] Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán, "Smaller, more accurate regression forests using tree alternating optimization," in *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, Hal Daumé III and Aarti Singh, Eds., Online, July 13–18 2020, pp. 11398–11408.

[3] Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, Calif., 1984.

[4] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[5] Robert M. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4–29, Apr. 1984.

[6] John Makhoul, Salim Roucos, and Herbert Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, no. 11, pp. 1551–1588, Nov. 1985.

[7] Albert Cohen, Wolfgang Dahmen, Ingrid Daubechies, and Ronald DeVore, "Tree approximation and optimal encoding," *Applied and Computational Harmonic Analysis*, vol. 11, no. 2, pp. 192–226, Sept. 2001.

[8] Jianhua Lin and James A. Storer, "Design and performance of tree-structured vector quantizers," *Information Processing & Management*, vol. 30, no. 6, pp. 851–862, Nov. – Dec. 1994.

[9] Yoav Freund, Sanjoy Dasgupta, Maynak Kabra, and Nakul Verma, "Learning the structure of manifolds using random projections," in *Advances in Neural Information Processing Systems (NIPS)*, John C. Platt, Daphne Koller, Yoram Singer, and Sam Roweis, Eds. 2008, vol. 20, pp. 473–480, MIT Press, Cambridge, MA.

[10] Lai-Man Po and Chok-Ki Chan, "Adaptive dimensionality reduction techniques for tree-structured vector quantization," *IEEE Trans. Comm.*, vol. 42, no. 6, pp. 2246–2257, June 1994.

[11] Sanjoy Dasgupta and Yoav Freund, "Random projection trees for vector quantization," *IEEE Trans. Information Theory*, vol. 55, no. 7, pp. 3229–3242, July 2009.

[12] Philip A. Chou, Tom Lookabaugh, and Robert M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Image Processing*, vol. 35, no. 2, pp. 299–315, Mar. 1989.

[13] Giovanni Poggi and Richard A. Olshen, "Pruned tree-structured vector quantization of medical images with segmentation and improved prediction," *IEEE Trans. Image Processing*, vol. 4, no. 6, pp. 734–741, June 1995.

[14] Hayder Radha, Martin Vetterli, and Ricardo Leonardi, "Image compression using binary space partitioning trees," *IEEE Trans. Image Processing*, vol. 5, no. 12, pp. 1610–1624, Dec. 1996.

[15] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011, Available online at `https://scikit-learn.org`.

[16] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008.

[17] Hervé Jégou, Matthijs Douze, and Cordelia Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, Jan. 2011.