Fast Covariance Computation and Dimensionality Reduction for Sub-Window Features in Images

Vivek Kwatra and Mei Han

Google Research, Mountain View, CA 94043

Abstract. This paper presents algorithms for efficiently computing the covariance matrix for features that form sub-windows in a large multi-dimensional image. For example, several image processing applications, *e.g.* texture analysis/synthesis, image retrieval, and compression, operate upon *patches* within an image. These patches are usually projected onto a low-dimensional feature space using dimensionality reduction techniques such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), which in-turn requires computation of the covariance matrix from a set of features. Covariance computation is usually the bottleneck during PCA or LDA ($O(nd^2)$ where n is the number of pixels in the image and d is the dimensionality of the vector). Our approach reduces the complexity of covariance computation by exploiting the redundancy between feature vectors corresponding to overlapping patches. Specifically, we show that the covariance between two feature components can be reduced to a function of the relative displacement between those components in patch space. One can then employ a lookup table to store covariance values by relative displacement. By operating in the frequency domain, this lookup table can be computed in $O(n \log n)$ time. We allow the patches to *sub-sample* the image, which is useful for hierarchical processing and also enables working with filtered responses over these patches, such as local *gist* features. We also propose a method for fast projection of sub-window patches onto the low-dimensional space.

1 Introduction

We consider the problem of efficiently computing the covariance matrix for feature vectors that can be expressed as sub-windows in a large image. This problem occurs in construction of codebooks for image patches, where each patch (sub-window) in the image is projected to a low-dimensional space using a dimensionality reduction technique such as Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). This low-dimensional representation is then useful for several tasks such as matching (search for patches with matching feature vectors in texture analysis/synthesis, example-based super-resolution, non-local image denoising and inpainting), compression (using Vector Quantization), and detection/recognition (e.g. face recognition using wavelet features).

Sub-window features may not be limited to 2D images, but also useful in 1D time-series such as audio signals and for 3D analysis in volumetric data or video.

We present an algorithm for efficiently computing the covariance matrix from these sub-window features by exploiting the redundancy between overlapping windows. Specifically, we show that the covariance between two feature components can be expressed as a function of the relative displacement between those components in patch space. This further reduces to a cross-correlation operation which can be computed quickly in frequency domain. Using a similar analysis, the projection of sub-window features onto the low-dimensional PCA or LDA basis can also be expressed as a cross-correlation (or filtering) operation, and therefore computed efficiently.

We are particularly motivated by texture analysis and synthesis tasks, where image patches or their filtered representations are used as descriptors of local image texture. Recent work on scene analysis employs *gist* descriptors for images [21]. The *local* version which computes gist features for sub-images and provides textural information for similar patch search is also based on sub-window features. Computing these descriptors requires learning weights for filter bank responses of the image. An intermediate step involves performing PCA over features representing sub-windows in the filtered response images. Due to the high dimensionality of these feature vectors, image windows are usually sub-sampled before performing PCA. However, using our approach, PCA can be performed efficiently without resorting to sub-sampling.

In example-based synthesis, super-resolution, and denoising algorithms [19, 7, 2], image patches matching a target patch are searched for repeatedly, making low-dimensional representations valuable for faster performance. PCA is a popular choice for this purpose, but may need to be applied to each example image independently for superior synthesis quality. Our fast covariance computation and low-dimensional projection algorithms significantly speed up the pre-processing time for these applications. Note that the local gist features described above can also be used in synthesis tasks for searching similar patches.

2 Related Work

Data analysis techniques such as PCA [22], LDA [6] and factor analysis [5] employ covariance matrix computation as an essential step. We specifically focus on dimensionality reduction of image patches, and the fast computation of covariance matrices for that purpose. Such efficient covariance computation would benefit several image processing applications including texture synthesis [19, 17, 27], image and video compression [28, 20], super resolution [7, 13, 26], non-local denoising [2, 1], inpainting [4, 15], image modeling [14], and image descriptors computation [12, 21].

Covariance estimation for high dimensional vectors is a classically difficult problem because the number of coefficients in the covariance grows as the dimension squared [25, 8, 10]. Most work on estimation of covariance matrices approximates the actual covariance matrix on the basis of a sample from a multivariate

distribution. Higham [9] provided a method for computing the nearest covariance matrix when only partially observed data are available. Cao and Bouman [3] presented a technique based on constrained maximum likelihood estimation for covariance matrices with $n < d$, where n versions of a d dimensional vector are given. We are solving the $n >> d$ case in which the observations are complete. We provide an efficient approach for the unique situation where the high dimensional vectors are sub-windows sliding in a large domain, such as from an image or an acoustic signal.

Qi and Leahy [24] described an approximate technique for fast computation of the covariance using maximum *a-posteriori* estimation. They extracted the covariance from multiple images. Porikli and Tuzel [23] presented an integral image based algorithm to efficiently extract covariance matrices from a given image. Their feature vector is composed of values defined at a single pixel. The typical dimensionality used in [23] is $d \approx 7$. On the contrary, in our method feature vectors are composed of values that span multiple pixels (patches) and have much higher dimensionality ($d = 3072$ for 32×32 RGB patches). One could express a patch based feature vector by unrolling the entire patch at every pixel and subsequently apply the integral based method for covariance computation. However, as per [23], computing the integral image takes $O(nd^2)$ time and storage (as $d + d^2$ integral images need to be computed). For large d-values, this is much slower than our method, which takes $O(n \log n)$ time. Moreover, the storage requirements for the integral image method are prohibitive in this case, requiring more than 20GB for a 100×100 image! The advantage of the integral image based method is that it allows covariance calculation over arbitrary windows in $O(d^2)$ time once the integral images have been computed. Our method on the other hand operates over the whole image (or a fixed window), but can handle an arbitrary mask or pixel weights if they are known *a-priori*.

3 Fast Covariance Computation

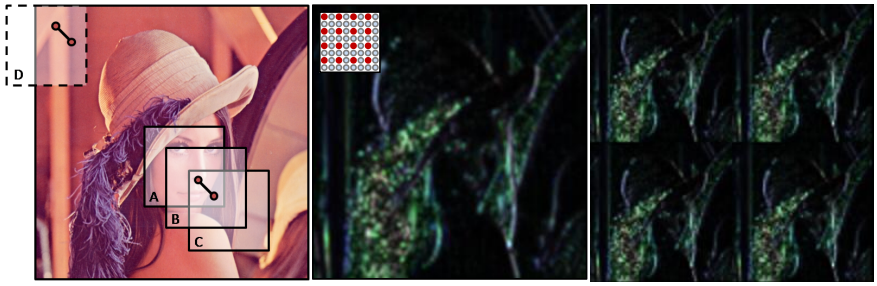
Computation of the covariance matrix from a given set of feature vectors is an expensive operation when the number and/or dimensionality of the feature vectors is large. A set of n feature vectors of dimensionality d can be expressed as the feature matrix $\mathbf{F}$:

$$\mathbf{F} = (\mathbf{f}_1 \ \mathbf{f}_2 \ \ldots \ \mathbf{f}_n), \qquad \text{where } \mathbf{f}_i = (f_{i1} \ f_{i2} \ \ldots \ f_{id})^T$$

is the i^{th} feature vector. The covariance matrix over these feature vectors (assuming zero-mean)[1] is:

$$\mathbf{C} = \frac{1}{n}\mathbf{F}\mathbf{F}^T = \frac{1}{n}\sum_{i=1}^{n}\mathbf{f}_i\mathbf{f}_i^T,$$

[1] The true covariance matrix is obtained by subtracting the outer product of the mean vector from $\mathbf{C}$.



(a) (b) (c)

Fig. 1: (a) The *diagonal pixel pair* in the middle of the image corresponds to different pixel pair locations (equivalently pairs of feature vector components) for *patches A, B, and C* (w.r.t their patch origins). Therefore the same pixel pair contributes to all pairs of feature vector component products that have the same relative pixel displacement. Pixel pairs near the boundary of the image contribute to the covariance of some "imaginary" patches (like *patch D*) that do not fully lie inside the image. (b) A filtered *local gist image* is shown along with *a sub-sampled patch*. The sub-window feature in this case is formed by collecting only *the red pixels* from the patch. Each local gist pixel stores the integrated filter response over the cell anchored at that pixel (see Section 4.1 for explanation). (c) Image obtained after repacking the local gist image

where each term $\mathbf{f}_i\mathbf{f}_i^T$ in the summation is the outer product of the feature vector $\mathbf{f}_i$ and takes $O(d^2)$ time, leading to a total time complexity of $O(nd^2)$.

Now consider the case where the feature vectors form sub-window patches in a training image. If the patch size is, say 32×32, then for a grayscale image, the dimensionality of the feature vector is $d = 32 \times 32 = 1024$. This is quite large, given that the covariance computation varies by d^2. However, since the patches are sub-windows in an image, we can exploit the redundancy between overlapping patches to speed up the computation.

For the i^{th} image patch, its feature vector's component f_{ij} corresponds to a location in the image, say $\mathbf{q}_{ij} = (q_{ij}^x, q_{ij}^y)$. If the patch's origin is anchored at location $\mathbf{t}_i = (t_i^x, t_i^y)$ in the image, we can express this location as $\mathbf{q}_{ij} = \mathbf{t}_i + \mathbf{p}_j$, where $\mathbf{p}_j = (p_j^x, p_j^y)$ is the location expressed w.r.t the patch's origin and is the same for all patches. Therefore, we can express f_{ij} as a function of the image from which features are extracted. This could be an intensity image if we are looking at intensity features, or a processed image containing filter responses, but returns a scalar feature value as a function of the pixel location[2]. If $\mathcal{I}$ denotes the image, then

$$f_{ij} = \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j). \tag{1}$$

[2] We consider vector-valued images in the next section.

If we focus on a single entry in the covariance matrix at (f_j, f_k), then:

$$\mathbf{C}(f_j, f_k) = \frac{1}{n}\sum_{i=1}^{n} f_{ij} f_{ik} \tag{2}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j)\mathcal{I}(\mathbf{t}_i + \mathbf{p}_k) \tag{3}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j)\mathcal{I}(\mathbf{t}_i + \mathbf{p}_j + \mathbf{v}_{jk}), \tag{4}$$

where $\mathbf{v}_{jk} = \mathbf{p}_k - \mathbf{p}_j$ is the displacement vector between the pixel locations corresponding to f_{ij} and f_{ik}. Now, if we treat the image as infinite, that is the number of patches $n \to \infty$ and the patch displacements $\mathbf{t}_i$ span all integer-valued locations in the plane, then we can drop $\mathbf{p}_j$ from the term $\mathbf{t}_i + \mathbf{p}_j$ in (4). This is possible because under this infinite span assumption, the pixels spanned by both $\mathbf{t}_i$ and $\mathbf{t}_i + \mathbf{p}_j$ are the same, and therefore the sum in (4) tends to the same value. Hence, we can rewrite (4) as:

$$\mathbf{C}(f_j, f_k) \approx \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}(\mathbf{t}_i)\mathcal{I}(\mathbf{t}_i + \mathbf{v}_{jk}) = \mathcal{C}(\mathbf{v}_{jk}), \tag{5}$$

i.e. the covariance value is only a function of the displacement between pixel locations corresponding to the feature vector's scalar components. Intuitively, this works because the same pixel pair in the *image* contributes to the sums for different pixel pairs in different *patches*, all with the same relative displacement, as shown in Fig. 1a. In practice, for a finite sized image, this formulation results in an approximation since pixel pairs near the boundary would not contribute to all products with the same relative displacement (also shown in Fig. 1a). However, for large enough images, this is an acceptable approximation: because we are aggregating these products, the error due to the extra accumulation from boundary pixels diminishes with increasing image size.

3.1 Algorithm

To compute the covariance matrix using (5), one can compute the product for all pixel pairs in the image with the same relative displacement and sum them up. These sums of products are stored in a lookup table indexed by the relative displacement $\mathbf{v}$. The entry $\mathbf{C}(f_j, f_k)$ in the covariance matrix is then assigned as value in the lookup table at index $\mathbf{v}_{jk} = \mathbf{p}_k - \mathbf{p}_j$, where $\mathbf{p}_j$ and $\mathbf{p}_k$ are corresponding pixel locations as defined above. To analyze the complexity of this algorithm, observe that we need to do this computation for d displacement vectors because the possible integer-valued relative displacements in a $w \times w$ sized patch is $w^2 = d$ (the dimensionality of the patch feature vectors). Also, each computation is done over all pixel pairs in the image which are $O(n)$, where n is the number of pixels in the image. Therefore the total complexity is $O(nd)$.

This is much better compared to the original complexity of $O(nd^2)$. For a 32×32 patch for example, this is three orders of magnitude faster.

One can further speed up covariance computation by observing that (5) represents the 2D auto-correlation function of the image $\mathcal{I}$, which can be computed efficiently in frequency domain using the Fast Fourier Transform (FFT). The complexity of this algorithm is bounded by the complexity of FFT computation, which is $O(n \log n)$. For patches with large dimensionality $d >> \log n$, this is faster than computing the lookup table by explicit summation of products.

4 Extension to Vector Images and Gist Features

The covariance computation approach described above assumes scalar-valued images. It can be extended to vector-valued images, where the feature vector is formed by concatenation of the vector components at each pixel in the patch. Vector-valued images may include multi-channel color images, or images obtained as responses of filter banks applied to the original image. For example, it is common to apply gradient or Gabor filters [11] to images for texture analysis as well as for computation of global scene features in the gist algorithm [21].

Consider a vector-valued $\mathcal{I}$ image with c channels. A feature value in an image patch now corresponds to a channel in addition to a pixel location. For the i^{th} patch, feature component f_{ij} corresponds to location $\mathbf{q}_{ij} = \mathbf{t}_i + \mathbf{p}_j$ and channel c_j. Hence, (1) and (4) respectively become

$$f_{ij} = \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j, c_j), \quad \text{and}$$

$$\mathbf{C}(f_j, f_k) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j, c_j) \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j + \mathbf{v}_{jk}, c_k).$$

By applying the same argument as used for deriving (5), we obtain

$$\mathbf{C}(f_j, f_k) \approx \frac{1}{n} \sum_{i=1}^{n} \mathcal{I}(\mathbf{t}_i, c_j) \mathcal{I}(\mathbf{t}_i + \mathbf{v}_{jk}, c_k),$$

i.e. , the covariance value corresponding to a pair of features is a function of the channels they belong to in addition to the relative displacement in patch space. Instead of representing the auto-correlation function of the image, the covariance now represents the cross-correlation between the respective channels of the image. Therefore, frequency domain computation can still be employed. However, the cross-correlation needs to be computed across all (unordered) pairs of image channels, making the total complexity $O(c^2 n \log n)$. However, this is still better than the complexity of the exact brute-force algorithm, which is $O(nd^2) = O(nc^2w^4)$, where w is the window size.

4.1 Sub-sampled Windows and Gist Features

We now consider sub-window features that sub-sample the original image. Figure 1b shows an example sub-sampled patch. Such sub-sampling of patches is

useful for computation of *local* gist features, which are gist features computed over patches. Compared with *global* gist features, which compute a gist image for entire image, we compute a gist patch for *every* image patch, where each cell in the gist patch is computed by integrating over a subset of pixels within the patch.

More specifically, local gist features are computed as weighted filter responses over local image patches. Firstly, a multi-channel image is obtained by applying several filters to the image such as Gabor wavelets and/or oriented gradient filters. Every patch over which the feature vector needs to be extracted is further divided into a grid of cells, where each cell contains $s \times s$ pixels (typically $s = 4$). The filtered images are integrated within these cells for each patch to form a feature vector of size $\frac{w}{s} \times \frac{w}{s} \times c$, where $\frac{w}{s}$ is the number of cells along each dimension within a patch's grid and c is the number of filtered channels. One can then organize these integrated cell responses into a *local gist image*, where each pixel stores the integrated response for the cell anchored at that pixel (see Fig. 1b for a visualization of the gist image, shown with 2×2 cells). The feature vector corresponding to a patch can then be obtained by sub-sampling the local gist image every s pixels.

These features are used to form patch-level scene descriptors in retrieval and recognition tasks. Local gist features are also useful for searching patches within an image for graphics applications such as example-based texture synthesis and super resolution.

Another application of sub-sampled patches is hierarchical processing. For example, in [18], a Gaussian stack (instead of a pyramid) is used as the multi-scale representation of an image. Patches at lower resolutions in the stack are obtained by sub-sampling from corresponding filtered images with a successively larger step size.

Covariance computation for features corresponding to such sub-sampled patches follows the observation that feature values only interact with other feature values that are a multiple of s pixels away in either dimension, where s is the sub-sampling step size. Therefore one can *re-pack* the image pixels so that it results in a grid of $s \times s$ sub-images (as shown in Fig. 1c), where each sub-image now consists of densely sampled $\frac{w}{s} \times \frac{w}{s}$ patches. Covariance matrices may then be computed independently for each of these sub-images and averaged together to obtain the combined covariance. Alternately, because the sub-images need to be processed independently, there is no performance benefit to processing all of them together (as was the case with processing all patches together). Hence, it may be sufficient to compute the covariance based on just one of the sub-sampled images. Since each sub-image contains $\frac{n}{s^2}$ pixels, the complexity is $O(c^2 \frac{n}{s^2} \log \frac{n}{s^2})$ per sub-image (or $O(c^2 n \log \frac{n}{s^2})$ if all sub-images are used).

The re-packing described above may also be used for processing multiple images simultaneously by concatenating them together into a larger collage if the number of images is small. Alternatively, covariances for each image can be computed independently followed by weighted averaging.

5 Weighted Features

The above approach for covariance computation can be extended to the case in which pixels have arbitrary weights. This may be useful in case certain patches are more preferrable than others, *e.g.* those near interest points or high gradients. The caveat is that the weights need to be expressed per-pixel, as opposed to per-patch. However, a simple way to achieve that is to assign to every pixel the average weight of patches overlapping it. The per-pixel weights may also be used to specify an image mask that selects the pixels to be considered. In presence of weights, (5) becomes

$$\begin{aligned}\mathbf{C}(f_j, f_k) &\approx \frac{\sum_{i=1}^{n} \mathbf{W}(\mathbf{t}_i)\mathcal{I}(\mathbf{t}_i)\mathbf{W}(\mathbf{t}_i + \mathbf{v}_{jk})\mathcal{I}(\mathbf{t}_i + \mathbf{v}_{jk})}{\sum_{i=1}^{n} \mathbf{W}(\mathbf{t}_i)\mathbf{W}(\mathbf{t}_i + \mathbf{v}_{jk})} \\ &= \frac{\sum_{i=1}^{n} \mathbf{W}\mathcal{I}(\mathbf{t}_i)\mathbf{W}\mathcal{I}(\mathbf{t}_i + \mathbf{v}_{jk})}{\sum_{i=1}^{n} \mathbf{W}(\mathbf{t}_i)\mathbf{W}(\mathbf{t}_i + \mathbf{v}_{jk})}\end{aligned}$$

where $\mathbf{W}$ denotes the per-pixel weights and $\mathbf{W}\mathcal{I}$ denotes the *weighted image*, obtained by multiplying the weights with the image at every pixel. The numerator and denominator denote cross-correlation and auto-correlation operations respectively and therefore can be computed efficiently as described earlier.

6 Fast Dimensionality Reduction

The covariance matrix computation described above can be used as a pre-process for performing PCA or LDA on the original feature vectors. However, to use the computed principal components for dimensionality reduction, it is necessary to project the original high-dimensional feature vectors onto the low-dimensional space represented by the principal basis. We can again exploit the redundancy across overlapping sub-windows to perform this operation efficiently as well.

Projecting a sub-window patch onto a single principal basis vector entails computing a dot product between the two vectors which is an $O(d)$ operation, where $d = c \times w \times w$ is the dimensionality of the patch. Therefore projecting *all* sub-windows within the image onto a single basis vector requires $O(nd) = O(ncw^2)$ computation for an image with n pixels. However, if we interpret each principal component vector as a patch, then the basis coefficient b_k for an image patch anchored at location $\mathbf{t}_i$ w.r.t the k^{th} principal basis patch $\mathcal{B}_k$ can be expressed as:

$$b_k(\mathbf{t}_i) = \sum_{l=1}^{c} \sum_{j=1}^{w^2} \mathcal{I}(\mathbf{t}_i + \mathbf{p}_j, c_l)\mathcal{B}_k(\mathbf{p}_j, c_l)$$

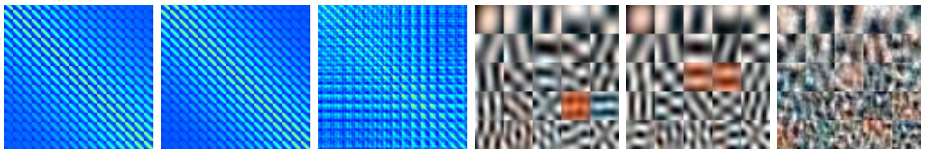
where $\mathbf{p}_j$ spans the $w \times w$ patch window. Since we want to compute b_k for all values of $\mathbf{t}_i$, this is equivalent to filtering the image $\mathcal{I}$ with the basis patch $\mathcal{B}_k$. This can be again efficiently computed in $O(cn \log n)$ time in the frequency domain, which is significantly faster when the patch size is non-trivial, *i.e.* $w^2 >> \log n$.

(a) Crowd (b) Traffic (c) Leopard skin (d) Green scales

Fig. 2: Reference images used in quantitative experiments (also refer Lena in Fig. 1a)

7 Experimental Results

In our experiments, we compute covariances for patches extracted from RGB images and gist images (which are 6 channel Gabor-filtered images with responses integrated over cells of 4×4 pixels). We have used a dataset consisting of texture images, natural scenes, and urban imagery (see Fig. 1a and Fig. 2 for a few examples that we will refer to subsequently). We compute covariances using three methods: (1) the *exact* method that computes the average covariance over all feature vectors explicitly, (2) our frequency domain *FFT*-based method, and (3) a *sampling* method that sub-samples the image for feature vectors, only using $\frac{n}{w^2}$ vectors, either randomly or over a regular grid.



(a) (b)

Fig. 3: Comparison of covariance matrices and PCA basis vectors computed over 16×16 patches extracted from the Crowd image. Visualization of resulting *covariance matrices* (a) and *top 25 basis vectors* (b): shown from left-to-right are the results for the exact method, our FFT-based method, and the sampling method, respectively. The covariance matrix obtained by our method has the same structure as the exact covariance, while the sampling method exhibits aliasing which is also evident in the principal components. The principal components obtained by our method closely resemble the smooth exact bases, with the top few components being nearly identical

Table 1 demonstrates the performance gain we achieve in covariance computation as well as PCA projection over the respective exact methods. Our covariance computation is 2-3 orders of magnitude faster, while projection is about an order of magnitude faster. The sampling method is slower than ours for the chosen sampling rate, without being as accurate. Random sampling or sampling over a grid generate similar results. Figure 3 shows a visualization of the covariance matrices and the principal components obtained using the three

methods for the Crowd image. The covariance matrix obtained by our method has the same structure as the exact covariance, while the sampling method exhibits aliasing as it is biased towards the sampled patches (also evident in the principal components). Note that the aliasing is not due to sampling on a regular grid since contributions from all samples are averaged together. The principal components obtained by our method, on the other hand, closely resemble the smooth exact bases, with the top few components being nearly identical.

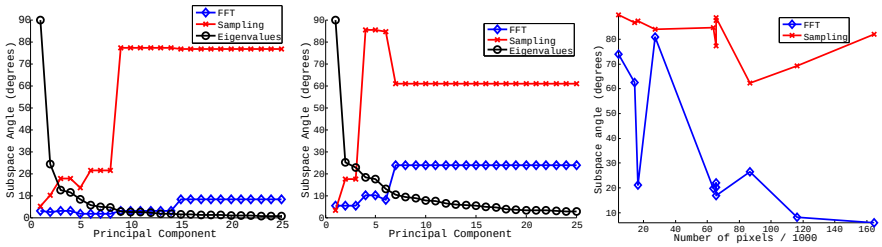
Table 1: Performance comparison between various methods. CPU time shown in seconds. Our method (FFT) is 2-3 orders of magnitude faster for covariance computation, and about an order of magnitude faster for projection, compared with exact method. The sampling method is slower than ours for the chosen sampling rate

Image (size)	Covariance Time						Projection Time	
	16×16 RGB			32×32 gist			16×16 RGB	
	Exact	**FFT**	Sampling	Exact	**FFT**	Sampling	Exact	**FFT**
Scales (64x64)	6	0.04	0.04	0.62	0.02	0.02	0.7	0.1
Grass (120x120)	24	0.07	0.12	5	0.06	0.08	3	0.5
Leopard (128x128)	29	0.07	0.15	5	0.06	0.1	3.5	0.6
Crowd (150x180)	50	0.11	0.23	10	0.09	0.17	6	1.2
Gecko (256x256)	130	0.23	0.57	29	0.23	0.5	16	3
Lena (256x256)	130	0.23	0.59	29	0.22	0.49	16	3
Text (256x256)	131	0.23	0.57	29	0.25	0.49	16	3.2
Windows (306x208)	125	0.23	0.57	27	0.22	0.45	16	3.6
Ropes (360x240)	177	0.3	0.74	39	0.32	0.64	21	5
Traffic (390x300)	239	0.41	0.97	54	0.42	0.85	29	7.5
Building (865x190)	341	0.6	1.34	74	0.74	1.22	41	8

Figure 4 is a quantitative comparison of the covariance matrices computed using our method and the sampling method against the exact covariance. Since we are ultimately interested in the principal components obtained from the covariance, we compare the subspaces induced by these components. We group successive principal components obtained from the exact method into subspaces if the ratio between their respective eigenvalues is less than a threshold (1.2 in our experiments). This is necessary because the principal eigenvectors become unstable when their corresponding eigenvalues are close to each other. Therefore it makes more sense to compare the grouped subspaces as opposed to individual eigenvectors. Note that we do consistently better than the sampling method (*i.e.* have smaller subspace angles). Also, the subspace angle increases only close to where the eigenvalue curve becomes flat, *i.e.* after most of the variance has been captured. The rightmost plot shows that the subspace angle for our method generally decreases as the number of pixels in the image increase, confirming our hypothesis that the approximation should improve with image size.

Figure 5 compares the reconstruction error and Fig. 6 compares the nearest neighbor (NN) search performance between our method (FFT) and sampling

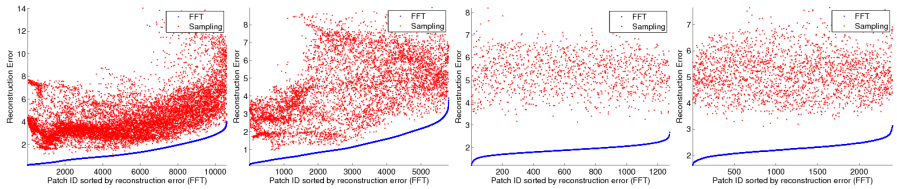
method, Our method results in lower reconstruction error, and the NNs from our method consistently have lower true distances to query patches than sampling method. The details are in the captions of Fig. 5 and Fig. 6.



(a) (b) (c)

Fig. 4: Subspace angles between principal components obtained by each method (FFT and sampling) w.r.t the exact method. (a) is a plot for 16×16 patches extracted from an RGB image (Traffic), and (b) is a plot for 32×32 patches in a 6-channel gist image (corresponds to an 8×8 grid of cells 4×4 pixels each). In these two plots, the eigenvalues of principal components are plotted (scaled and shifted to fit graph). Our method consistently generates smaller subspace angles compared with sampling method. There is a jump in the curve when a new subspace is created, and the curve stays flat when the new eigenvector is added to the same subspace but does not change the angle considerably. (c) shows the angle between the subspaces induced by the top 10 eigenvectors of the two methods for different images listed in Table 1, plotted as a function of number of pixels in the images. The subspace angle for our method generally decreases as the number of pixels in the image increase, confirming our hypothesis that the approximation should improve with image size

We have applied our technique for accelerating example-based super resolution and texture synthesis. A practical setting where these methods may be employed is for resolution enhancement and hole filling of building facades in large scale 3D urban environments. For super resolution, given a low resolution target image and a high resolution (partial) source image with similar texture, we synthesize a high resolution version of the target (our algorithm combines ideas from [7] and [16]). The core of the synthesis algorithm is patch-based search performed in PCA space. The most time consuming component is covariance computation and feature projection to PCA space. Figure 7a shows a sample super resolution result. We extract 32×32 patches in this 202×402 image. The computation time is 3.1s using the fast covariance computation vs. 2415s using the exact covariance computation. A similar approach may be used for hole filling (see Fig. 7b). Again, we take advantage of the fast covariance computation to improve the processing time by more than 2 orders of magnitude.



(a) Traffic (b) Lena (c) Leopard skin (d) Green scales

Fig. 5: Comparison of reconstruction error (*y-axis*) based on top 25 PCA coefficients for basis vectors computed using our FFT method (*blue curve*) and the sampling method (*red dots*). Each plot shows reconstruction error for all 16×16 patches in the image, sorted (along the *x-axis*) by increasing FFT method error. The error from our method's PCA basis is consistently smaller than that from the sampling method

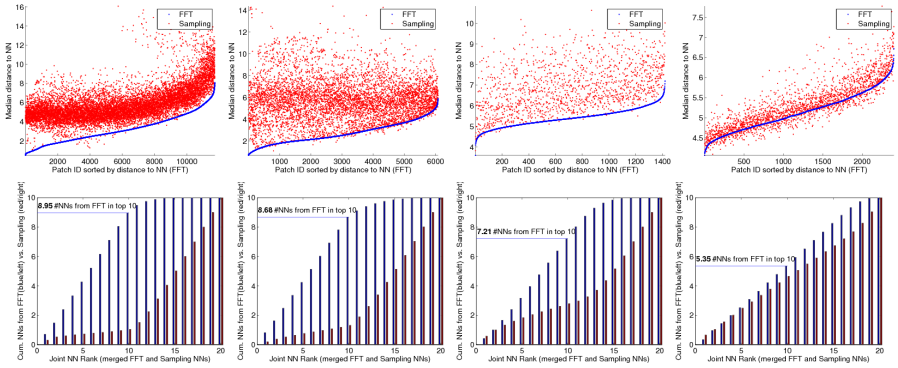


Fig. 6: Nearest neighbor performance on 16×16 patches from (left to right) Traffic, Lena, Leopard skin and Green scales. Top row plots the median distance (*y-axis*; *blue curve* for our FFT method and *red dots* for sampling method) from every patch to its top 10 nearest neighbors. Distances are computed over PCA coefficients. We use a hierarchical search tree constructed from PCA projected patches for nearest neighbor search. Patches are sorted (along the *x-axis*) by FFT method distance. These plots demonstrate that our FFT method consistently results in nearest neighbors with smaller median distance compared to the sampling method, except for Green scales where performance is more even. This is attributable to the fact that the Green scales texture is small in size (64×64) and therefore the approximation error in covariance matrix computation is not negligible. The bottom row plots cumulative histograms over the joint rank of nearest neighbors collected from the two methods (FFT and sampling). We find the top 10 nearest neighbors for each patch from both methods and jointly ranked the resulting 20 neighbors. Then for top K neighbors where K varies from 1 to 20 (plotted along the *x-axis*), we count how many neighbors come from the FFT method (*blue left-side bars*) vs. the sampling method (*red right-side bars*) and average this value over all patches. The resulting value (plotted on the *y-axis*) denotes the average number of nearest neighbors in the top-K, that come from the FFT method. Note that for the first three columns, this value for $K = 10$ lies between 7 and 9, indicating that the FFT method consistently results in better ranked neighbors. For the last column (Green scales), the performance is again evenly split (5.35) between the two methods

(a) (b)

Fig. 7: Sample Applications: (a) Example-based super resolution of building facades. Left to right: (partial) high resolution source image, low resolution target image, high resolution result. (b) Hole filling. The texture on the left contains a hole (shown in *black*) which is filled on the right using texture synthesis

8 Conclusion

We have proposed a novel algorithm to efficiently compute covariance matrices for features that can be described as sub-windows in an image. The overlapping nature of these sub-windows results in a special property for the covariance matrix, namely that the covariance between two pixel features is a function of their relative displacement. Using this property, covariance computation can be expressed as a cross-correlation operation, which can be computed quickly in the frequency domain. We have also presented extensions for vector-valued images and sub-sampled windows, as well as a method for fast low-dimensional projection of the sub-windows onto PCA space. Our formulation results in an approximation to the exact covariance, where the approximation error diminishes with increasing image size. We support this claim with both qualitative and quantitative experimental results. We also compare with a simple sampling approach to covariance estimation, and show that our technique results in a much closer approximation, while still being faster.

References

1. Adams, A., Gelfand, N., Dolson, J., Levoy, M.: Gaussian kd-trees for fast high-dimensional filtering. ACM Trans. Graph., SIGGRAPH 28(3), 1–12 (2009)
2. Buades, A., Coll, B., Morel, J.M.: A non-local algorithm for image denoising. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 60–65. IEEE Computer Society, Washington, DC, USA (2005)
3. Cao, G., Bouman, C.A.: Covariance estimation for high dimensional data vectors using the sparse matrix transform. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) NIPS. pp. 225–232. MIT Press (2008)
4. Criminisi, A., Prez, P., Toyama, K.: Region filling and object removal by exemplar-based image inpainting. IEEE Transactions on Image Processing 13 (2004)
5. Darlington, R.B., Weinberg, S., herbert, W.: Canonical variate analysis and related techniques. Review of Educational Research pp. 453–454 (1973)
6. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of Eugenics 7, 179–188 (1936)

7. Freeman, W.T., Jones, T.R., Pasztor, E.C.: Example-based super resolution. IEEE Comput. Graph. Appl. (2002)
8. Fukunaga, K.: Introduction to Statistical Pattern Recognition, Second Edition (Computer Science and Scientific Computing Series). Academic Press (1990)
9. Higham, N.J.: Computing the nearest correlation matrix a problem from finance. IMA Journal of Numerical Analysis 22(3), 329–343 (2002)
10. Jain, A.K., Duin, R.P.W., Mao, J.: Statistical pattern recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(1), 4–37 (2000)
11. Jones, J.P., Palmer, L.A.: An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. J Neurophysiol 58(6), 1233–1258 (December 1987)
12. Ke, Y., Sukthankar, R.: Pca-sift: a more distinctive representation for local image descriptors. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 2, pp. 506–513 (2004)
13. Kim, K.I., Franz, M., Schlkopf, B.: Kernel hebbian algorithm for single-frame super-resolution. In: Leonardis, A., H.B. (ed.) Statistical Learning in Computer Vision. pp. 135–149. Springer, Berlin, Germany (2004)
14. Kim, K.I., Franz, M.O., Schlkopf, B.: Iterative kernel principal component analysis for image modeling. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(9), 1351–1366 (2005)
15. Korah, T., Rasmussen, C.: Pca-based recognition for efficient inpainting. In: IEEE Asian Conference on Computer Vision (2006)
16. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. ACM Trans. Graph., SIGGRAPH 24(3), 795–802 (2005)
17. Lefebvre, S., Hoppe, H.: Appearance-space texture synthesis. In: Proc. of SIGGRAPH '06. pp. 541–548 (2006)
18. Lefebvre, S., Hoppe, H.: Parallel controllable texture synthesis. In: ACM Transactions on Graphics, SIGGRAPH. pp. 777–786 (2005)
19. Liang, L., Liu, C., Xu, Y., Guo, B., Shum, H.Y.: Real-time texture synthesis by patch-based sampling. ACM Trans. Graph. 20(3), 127–150 (2001)
20. Liu, J., Wu, F., Yao, L., Zhuang, Y.: A prediction error compression method with tensor-pca in video coding. In: MCAM. pp. 493–500 (2007)
21. Oliva, A., Torralba, A.: Building the gist of a scene: the role of global image features in recognition. Progress in brain research 155, 23–36 (2006)
22. Pearson, K.: On lines and planes of closest fit to systems of points in space. Philosophical Magazine 2(6), 559–572 (1901)
23. Porikli, W.F., Tuzel, O.: Fast construction of covariance matrices for arbitrary size image. In: Proc. Intl. Conf. on Image Processing. pp. 1581–1584 (2006)
24. Qi, J., Leahy, R.M.: Fast computation of the covariance of map reconstructions of pet images. Proceedings of SPIE 3661(1), 344–355 (1999)
25. Stein, C., Efron, B., Morris, C.: Improving the usual estimator of a normal covariance matrix. Dept. of Statistics, Stanford University, Report 37 (1972)
26. Wang, Q., Tang, X., Shum, H.Y.: Patch based blind image super resolution. In: ICCV (2005)
27. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR. Eurographics Association (2009)
28. Yu, Y.D., Kang, D.S., Kim, D.: Color image compression based on vector quantization using pca and lebld. In: Proc. of the IEEE Region 10 Conference. vol. 2, pp. 1259–1262 (1999)