PCA For Image Compression Extended with Shearing and Rectanglular Patch Sizes

Milan Bombsch, Tizian Zeltner, Rafael Häuselmann
Group: Zenigame
Computational Intelligence Lab 2015, ETH Zürich, Switzerland

***Abstract*—We present an extension to the regular PCA approach for image compression which uses a more generalized form of feature selection. The image is deformed with a shearing transformation to make the most dominant edge orthogonal to one of the image borders and we use rectangles instead of only square patches. The evaluation shows that for some images this method decreases the reconstruction error, while the compression rate stays the same. The running time of the extended algorithm increases due to the shearing operation.**

I. Introduction

One of the most basic approaches for image compression uses the PCA algorithm to reduce the image storage size. The image is usually divided into square patches of a fixed size. Each image patch is one high-dimensional point in the set of data on which PCA is conducted. See [1] for a detailed explanation of PCA. Due to the dimensionality reduction of PCA, this is generally a lossy compression. We show two possibilities to extend the feature extraction part:

- Image shearing is used to deform the image in such a way, that the most dominant edge in the image is orthogonal to one of the image borders.
- Instead of fixed-sized square patches, we use rectangular patches which depend on the images structure.

Both of these modifications help to align the patches with the features in the image, which should decrease the reconstruction error as we get more correlated features. We validate our ideas with images of sheared rectangles and then test the approach on real world examples.

II. Models and Methods

A. Base Algorithm - PCA

PCA [1] is a matrix factorization algorithm used for dimensionality reduction. In the case of image compression, we take the image as a matrix $I \in \mathbb{R}^{\mathrm{H}\times\mathrm{W}\times\mathrm{C}}$ and divide it into N patches of size $d \times d$. Each patch is then vectorized and used as a column in the feature matrix $X \in \mathbb{R}^{\mathrm{D}\times\mathrm{N}\cdot\mathrm{C}}$, where $D = d \times d$ and the features of the different color channels are just concatenated. The PCA of X then gives us μ, the mean value for each dimension, λ, the eigenvalues and U, the corresponding eigenvectors. To now save space for the compression we only take the k highest eigenvalues, where k is at the knee in the eigenspectrum. The image can be trivially reconstructed from its feature matrix, by forming the image patches from the columns of the feature matrix.

The compression algorithm therefore works in the following way:

The original image: I
The original image features: $X = \texttt{extract}(I)$
The compressed image features: $Z_k = U_k^T \cdot (X - \mu)$
The decompressed image features: $\tilde{X} = U_k \cdot Z_k + \mu$
The decompressed image: $\tilde{I} = \texttt{deextract}(\tilde{X})$

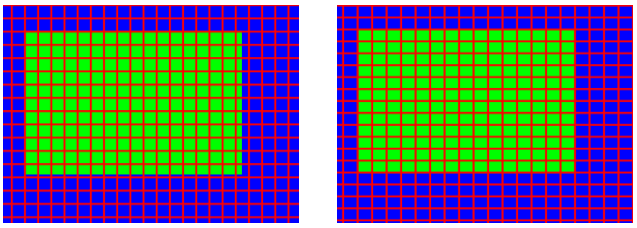
$k = D$ would give a perfect reconstruction.

B. Patch Sizes

In the following, we show that small differences in the size of the image patches during the feature extraction can lead to a dramatic decrease in reconstruction error while only minimally affecting running time and compression rate.

Figure 1 shows a test image of a single rectangle that is carefully placed. A grid is overlaid to visualize the patch size.

On the left, we use fixed-sized patches of 10×10 pixels, which produces a small overlap. On the right, we chose a customized patch size of 9×11 pixels which aligns perfectly with the rectangle. Figure 2 shows results of compressing both images. The well aligned patch size of 9×11 leads to 0 reconstruction error, which is of course far better than the one from the standard 10×10 patch size. The reason behind this is that if the patches are perfectly aligned we only get two types of features: completely green ones, and completely blue ones. As PCA does dimension reduction we are now able to describe the image perfectly by two image patches (points in the feature space). If the patches are not aligned with the green area we also get image features which are green *and* blue. This leads to a lossy compression, as we only consider the k highest eigenvalues. $k = 2$ holds in both cases, since the patches with mixed color do not form a unique third dimension in the feature space. This leads to a lossy reconstruction in the case of patches of size 10×10 and a lossless one for the patch size 9×11. The difference in the execution time is due to normal fluctuation of measuring CPU execution time and the small difference in the compression rate is due to the fact that the image padding (to make the size and width divisible by the patch size) is different for different patch size, which results in a differently shaped feature matrix X.



(a) Standard 10x10 pixel patches (b) Customized 9x11 pixel patches

Figure 1: Effect of different patch sizes on misalignedRectangle.png

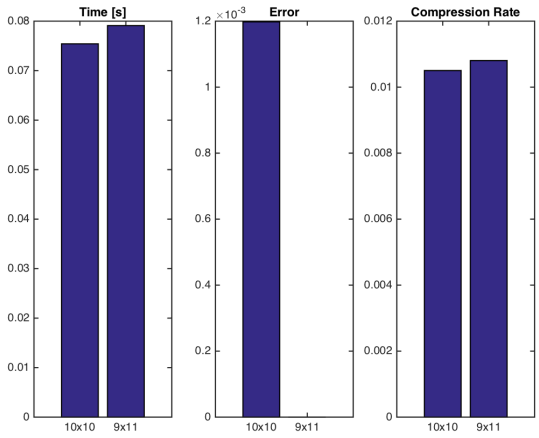


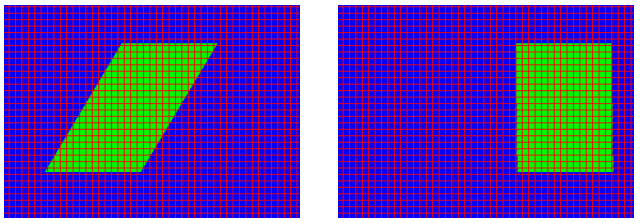
Figure 2: Difference of patches measured in Time, Error and Compression Rate on misalignedRectangle.png

C. Orientation

There are also other ways to make the image patches more aligned with the features in the image. The image could be deformed in such a way, that the orthogonal grid is optimally aligned with the edges in the image.

One such transformation could be image rotation, but this introduces an additional lossy step to the compression which is hard to invert. Instead, we followed an approach of image shearing, which is much easier to invert and has similar properties.

Figure 3 shows a similar setup as in the last section, where we use a simple shape for testing. On the left, there is a deformed rectangle, while on the right, the image is sheared to make the shape rectangular. Figure 4 shows results of compressing both images. The result is not as dramatic as for hand picked patch sizes, but still good enough to further investigate into this technique. The reason why the sheared image results in a lower reconstruction error is, that we again have more similar patches and the most relevant dimensions picked by PCA can better describe those patches. The compression rate stays the same, as we use exactly the same patch size which leads to a identical size of the feature matrix X. The difference in the execution time is due to normal fluctuation of measuring CPU execution time.



(a) Original (b) Deformed with shearing

Figure 3: Effect of image shearing on shearedRectangle.jpg

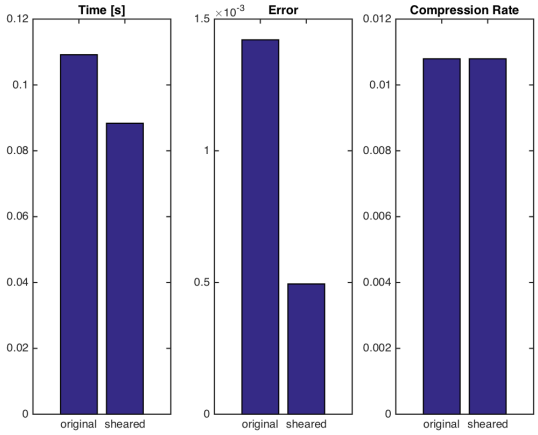


Figure 4: Difference of shearing measured in Time, Error and Compression Rate on shearedRectangle.jpg

D. Model Selection

For the model selection we need to find our parameters d and e for the patch sizes and x and y for the shearing, where y/x is the shear factor.

Let us first consider the shearing as this is done before extracting the image patches. The shearing should increase the number and the significance of the edges orthogonal to the image borders, as the patches will also be orthogonal to the image borders. To keep the computational overhead at a minimum we restrict ourselves to only aligning the most significant edge with the image borders. This can result in a worse compression if all edges except the most significant one are already oriented in the right way, but most of the time this will make the whole image more aligned with its borders. To detect the most significant edge we use the Hough Transform [2]. We then calculate the slope of this edge and from this infer the parameters x and y for the shearing of the image. The result of this can be seen in Figure 5a and 5b.

Finding the optimal patch size is a much harder task. The patch size should result in groups of patches which are very similar. This is definitely a non-convex problem, since a small change in the patch size leads to completely different patches. The patch size also highly affects the computation time for PCA and the size of the compressed image. Since the main edge of an image most of the time also separates different image parts, we decided to align the patches with this edge. The image patches on the same side of the main edge should then be quite similar. To stay in the same range for compression rate as PCA we start with the default 10×10 patch size and adjust it as little as possible to get the patches aligned with the main edge of the image. Since the shearing also moves the main edge within the image, we only search for the ideal patch size if the main edge is already vertically or horizontally aligned. Calculating the best patch size for arbitrary skewed main edges remains future work.

E. Extended Algorithm

The extended compression algorithm is listed here:

The original image: I
The path size and shearing parameters:
$[d, e, x, y] = \texttt{findParameters}(I)$
The sheared image: $I_S = \texttt{shearImage}(I, x, y)$
The sheared image features: $X = \texttt{extract}(I_S, d, e)$
The compressed image features: $Z_k = U_k^T \cdot (X - \mu)$
The decompressed image features: $\tilde{X} = U_k \cdot Z_k + \mu$
The decompressed sheared image:
$\tilde{I}_S = \texttt{deextract}(\tilde{X}, d, e)$
The decompressed image:
$\tilde{I} = \texttt{shearImage}(I, x, -y)$

In Figure 5, the individual steps of our algorithm are visualized on an example image.

III. Results

In this part we show the results of our algorithm relative to two benchmark algorithms: The standard PCA-approach described in [1] and an SVD-algorithm [3]. We show graphs which visualize the differences in running time, reconstruction error and compression rate. The three algorithms were tested on a set of 44 images from the USC-SIPI image database.[1]

A. Running Time

In Figure 6 can be observed that the running time of our algorithm is significantly higher than the running time of the standard PCA algorithm, due to our computation overhead for finding the right parameters and shearing the image. At the same time our algorithm takes roughly the same time as the SVD approach which makes our algorithm competitive.

[1] http://sipi.usc.edu/database/database.php?volume=misc

	PCA	SVD	Our Algorithm
Average	0.0314 s	0.1458 s	0.1178 s
Std dev	0.0289 s	0.1648 s	0.1154 s

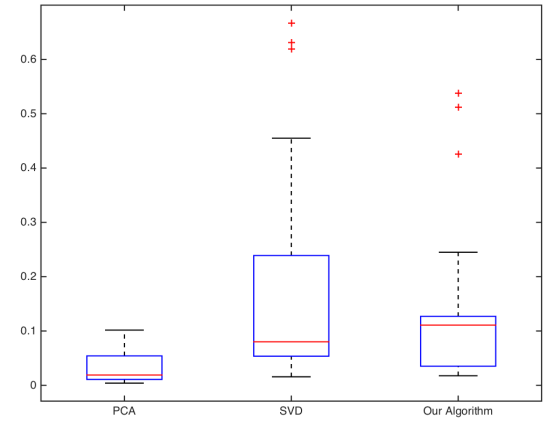


Figure 6: Evaluation of Running Time (in seconds)

B. Reconstruction Error

Figure 7 shows that our algorithm leads to an reconstruction error that is similar to the one from PCA since our algorithm is most of the time not able to find the optimal shearing and patch sizes to get the best reconstruction error. For prepared examples our algorithm performs really well, but for natural image the model selection needs to be improved.

	PCA	SVD	Our Algorithm
Average	0.0041	0.0018	0.0041
Std dev	0.0046	0.0018	0.0045

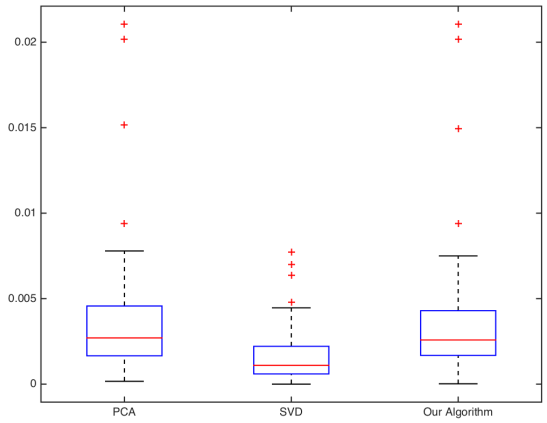


Figure 7: Evaluation of Reconstruction Error

(a) Original image with main edge visualization (b) Sheared image to make main edge vertical (c) Sheared image with path size visualization

Figure 5: Extended algorithm illustrated on an example image

C. Compression Rate

In Figure 8 we see that the compression rate of our algorithm is quite similar to one from PCA. This is due to the fact that we explicitly tried to get a similar compression rate as PCA, by using only slightly adjusted patch sizes. If we would use arbitrary patch sizes it would be hard to compare our Algorithm with PCA, since the compression rate would go down really fast by using bigger patches. Instead we wanted to concentrate on the correct alignment of these patches. The compression rate of the SVD algorithm is not included in this graph, because it is on average higher by a large amount than the other two.

	PCA	SVD	Our Algorithm
Average	0.0229	0.1596	0.0235
Std dev	0.0086	0.0657	0.0114

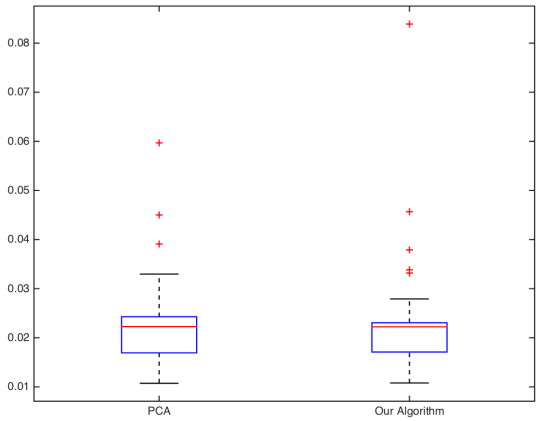


Figure 8: Evaluation of Compression Rates

IV. Discussion

As we showed with some example images, this approach can be very beneficial. But unfortunately most real world pictures do not benefit greatly from this approach, due to the complexity of their features.

We believe that the basic idea of this approach is good, since the additional information that we need to store are only four numbers (two for patch size, two for shearing) and those do not cause a big increase on the information stored in the compressed image. The problem though is that the overhead for computing this additional information is high and it does not pay off in the current form.

An even further extension of our algorithm would be to shear the image multiple times (horizontally and vertically) because the current implementation only shears in one direction per image.

Another possible extension that could be realized in future works is to rotate individual patches for feature extraction, such that every patch is aligned to a feature in the image in the best possible way. This would probably increase the size of the compressed image as we need to store extra information per patch, but it could pay off for a much smaller reconstruction error.

V. Summary

Our extension of the standard PCA algorithm, shearing the image before feature extraction and choosing rectangular patches for image extraction, can reduce the reconstruction error by a great amount, in tradeoff for an increased running time. Though on most real world images the error reduction is minimal or not existent. There are still options to further improve the extension made, but such are likely to also further increase the running time.

References

[1] I. T. Joliffe, *Principal Component Analysis*. Springer-Verlag, 1986.

[2] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, 1972.

[3] H. Prasantha, H. Shashidhara, and K. N. B. Murthy, "Image compression using SVD," *Conference on Computational Intelligence and Multimedia Applications*, pp. 143–145, 2007.