Hierarchical PCA on Video from Traffic Surveillance Camera

Tetiana Rabiichuk

June 26, 2021

Introduction

Principal Component Analysis is a widely used classical dimensionality reduction method. In the current project, we consider Principal Component Analysis for background extraction from static surveillance camera video (`traffic.avi`). We test whether the hierarchical application (composition) of PCA can produce better background extraction than the straightforward application of PCA.

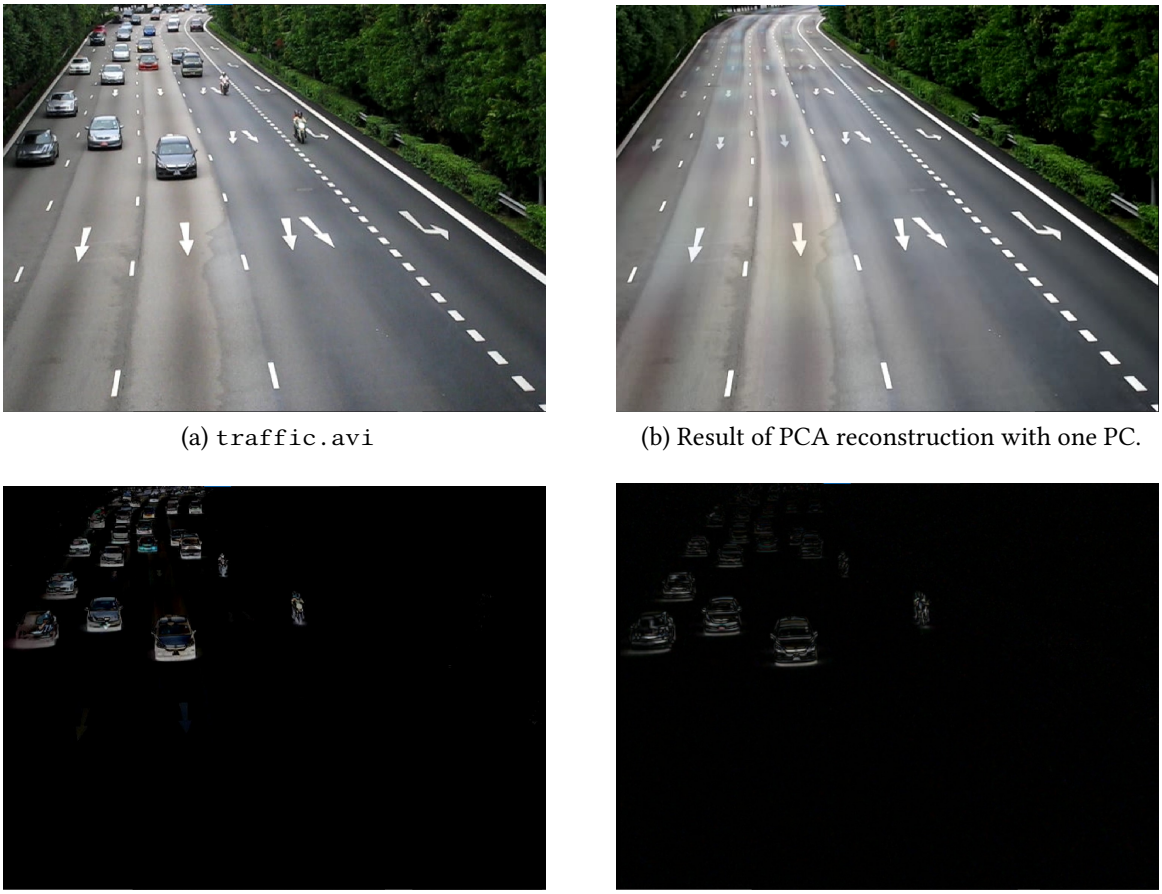
Approaches

Data description

Reading and writing operations on video files were performed with the OpenCV library. Since we would like to apply PCA to all video frames simultaneously, reading video frame-by-frame is not convenient. Hence, in `save_frames_as_npy.py` script we extract frames from input video `trafffic.avi` and stack them into NumPy array of dimensionality $(299, 480, 640, 3)$, where the first dimension corresponds to frames, 480×640 is a dimensionality of a frame, 3 - number of color channels. We will process each color channel separately to preserve colors. We have used scikit-learn PCA implementation.

0.1 Naive PCA

This method is a straightforward application of the PCA to the input video. First, we flatten frames of each color channel obtaining 3 matrices of dimensionality 299×307200. Next, we apply PCA to each of them separately. We use the same number n of principal components for each channel. Next, we reconstruct original frames from this n components. Even though the first principle component explains only ~ 0.04 of variance, it is enough to obtain quite a descent background extraction, as one can observe in Figure 1. We store corresponding result in `results\naive_pca\inverse_pca.avi`. One can also consider the absolute difference between original frames and extracted background. If the background extraction is good, we obtain foreground objects, in our case, it would be moving vehicles. Corresponding result is stored in (`results\naive_pca\abs_diff.avi`). For comparison, the project folder also contains an example of foreground extraction that was provided in the Intelligent Transportation Systems course (`example_background_ITS_course.avi`).



(a) `traffic.avi`

(b) Result of PCA reconstruction with one PC.

(c) Absolute difference between frame and reconstruction

(d) Foreground extraction provided in Intelligent Transportation Systems course.

Figure 1: Example of naive PCA application.

Adding mode principle components to the reconstruction restores traffic movement. However, dimensionality reduction is a classical application of PCA. Since we did not have any downstream task (classification/segmentation) as a project goal, we could not compare the quality of created embedding. However, as one can observe in Figure 2, for some frames, we also extract foreground. Hence, in the rest of the project, we have focused on verifying whether applying hierarchical PCA would improve the obtained background extraction.

0.2 PCA on tiles

We have split frames of the original video into four tiles: top left, top right, bottom left and bottom right with dimensions $299 \times 240 \times 320$. We have applied PCA to each tile deck separately and restored original frames with one principal component. As one can infer from the input video, the first principle components in the top right and bottom right tiles have larger explained variance ratios (~ 0.1, ~ 0.13) since the traffic intensity there is low. Results are contained in `results\pca_on_tiles`. However, this approach introduced artifacts where corresponding tiles have been glued together (see Figure 3). The next step was to verify whether applying multiple levels of PCA would reduce the artifacts on tile edges and whether further multiple levels of compression improve background extraction.

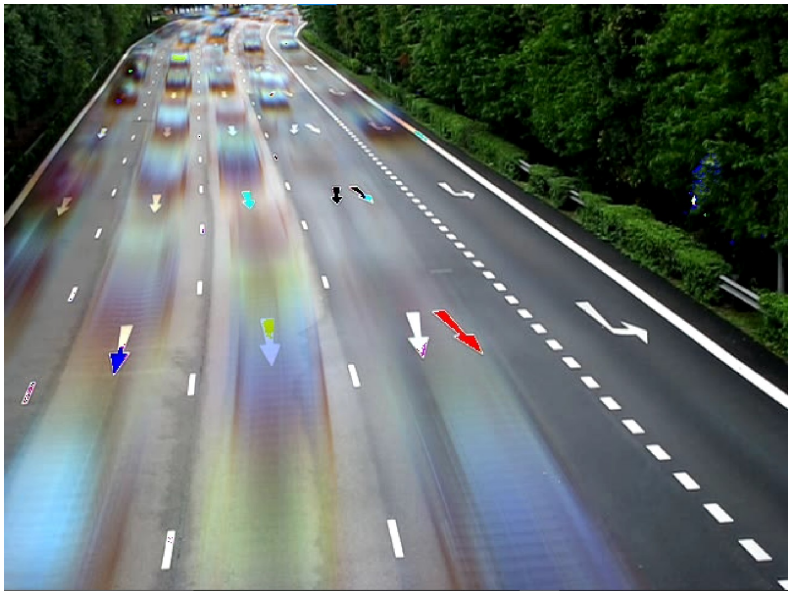


Figure 2: Background extraction obtained with naive PCA application is not perfect.

Figure 3: Applying PCA on each tile separately introduces border artifacts.

0.3 Two-layer PCA

We wanted to verify whether applying PCA on top of the features extracted from tile decks would help reducing border artifacts since this includes a global view of the frame. Resulting architecture consists of 2 passes: *forward pass* and *backward pass*. The function *two_layer_pca* is implemented in `hierarchical_pca.py`. First, we split input frames into four tiles. Next, we apply PCA to each of the tile decks. We stack together results in the matrix, keeping the location information. Next, we flatten matrices that correspond to frames and apply PCA transformation. Hence, we further reduce the dimensionality of feature space. This concludes the forward pass. We project data back into the original space in the backward pass, starting from the last transformation. Hence, we first reconstruct features extracted from tiles in the first hidden layer and then reconstruct corresponding tiles. Since we use only the first $n \lll m$

principal components, we retrieve the information that stores the background. However, the obtained reconstruction still contains border artifacts. Best background extraction was obtained using 1 principle component in the first PCA layer and 1 in the last layer. Using more components restores traffic information. The results are stored in `results\hierarchical_pca\two_layered_pca`.

0.4 Three-layer PCA

Code for this part is contained in `hierarchical_pca.py`, sample output in `results\hierarchical_pca\three_layered_pca`. In this approach, we expand previous architecture by adding one more hidden layer of PCA application. First, we split frames into four tiles, apply PCA to each of the tiles. Next, we apply PCA to the transformation of the upper part (PCA of the top left and top right tiles) and lower part (PCA of bottom left and bottom right tiles). We stack together obtained transformations and apply the final PCA transformation on top of it. In the backward pass, we consecutively restore the corresponding layer back to frames. This approach gives slightly better results (with minimum number of PC used for reconstruction). However, border artifacts are still visible (see Figure 4).

Figure 4: Three-layered architecture still produces border artifacts.

0.5 Best result

We have observed that even though the result slightly improves, we still get border artifacts. This inspired us to try methods described above not on the original frames split into tiles, but first, compress frames with PCA, and then apply hierarchical approaches to the obtained compressed matrix. This way, in the last step of the backward pass, we do not restore parts of the image independently (that introduces artifacts), but we restore the whole image. Hence, this may help with artifacts. It turned out to be the best approach. The function *hierarchical_pca*is implemented in `hierarchical_pca.py` One can find corresponding result in `results\hierarchical_pca\best_result`. The schematic architecture is described in Figure 5

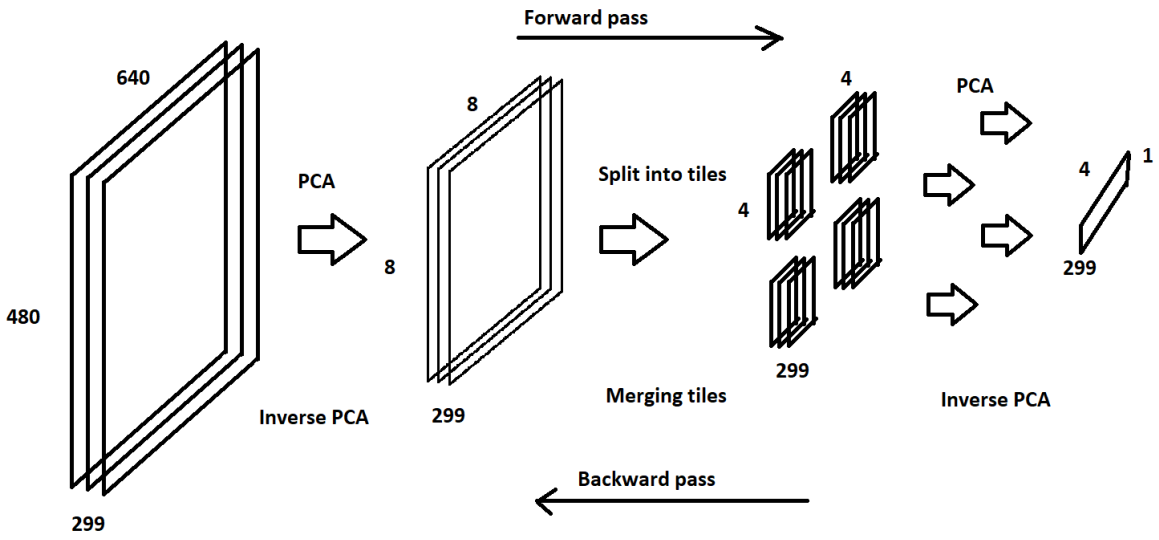
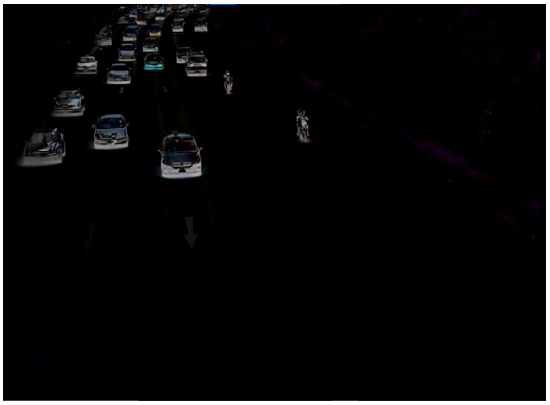


Figure 5: Architecture that produced best result (`hierarchical_pca.avi`). Note that that color channel information is not depicted.

In the first step, we have applied PCA to the original stack of frames, keeping only 64 components (~ 0.84 of explained variance ratio). Next, we have reassembled those components into matrix and applied the two-layered PCA function introduced in subsection Two-layer PCA. As a result, we have obtained visually perfect background reconstruction with no border artifacts, as one can observe in Figure 6.

(a) Reconstruction.



(b) Absolute difference between frame and reconstruction

Figure 6: Output generated by best architecture for the same timestamp as the one in Figure 1.

1 Discussion

We have observed that one can use PCA for background extraction in the case of static camera input. Moreover, introducing the composition of PCA transformations (hierarchical application) improves the result obtained by straightforward PCA application. However, when

we split frames into tiles and fit each tile separately, we introduce border artifacts since, in the backward pass, we still restore corresponding tiles independently. This can be avoided if we restore the original frame as a whole in the last step of the backward pass. This is achieved by first applying PCA to frames with original dimensions. Then, reshaping corresponding representation and applying hierarchical PCA on top of it. As a result, we obtain background with no visible border artifacts, and we do not pick up traffic information anymore.