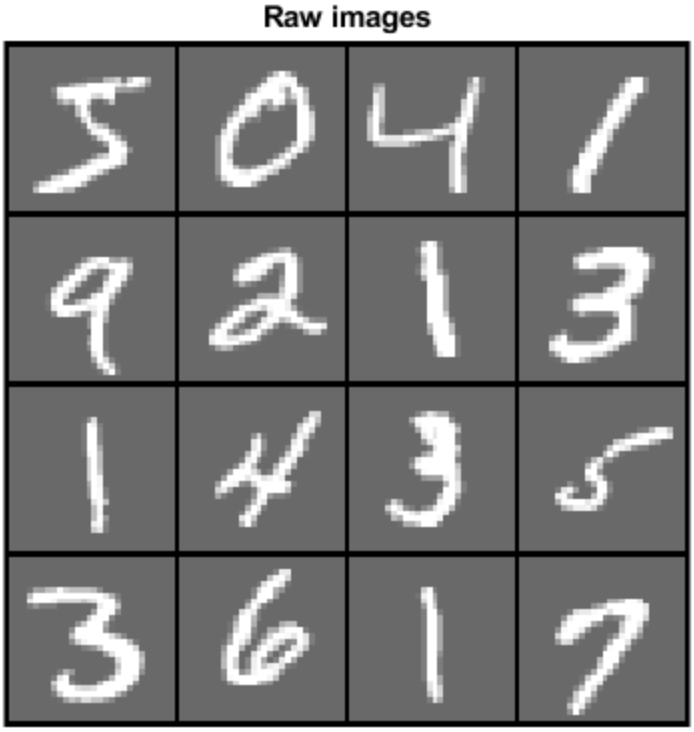
Table of Contents


```
clc, close('all'), clear variables
%%=====================================================================
```

Step 0a: Load data

```
        Here we provide the code to load natural image data into x.
        x will be a 784 * 600000 matrix, where the kth column x(:, k)  to
        the raw image data from the kth 28x28 image patch sampled.
        You do not need to change the code below.

x = loadMNISTImages('train-images-idx3-ubyte');

% Figure of first images (observations) to visualization.
first_images = (1:16)';
display_network(x(:,first_images));
title('Raw images');

%%=====================================================================
```



Step 0b: Zero-mean the data (by row)

You can make use of the mean and repmat/bsxfun functions.
Compute the mean of each image.

```
meanX = mean(x, 1);
x = bsxfun(@minus, x, meanX);

% Observe that the meanX now is zero.
meanX = mean(x, 1);
disp(meanX(1:5));

%%================================================================
```

```
   1.0e-15 *

    0.3034    0.1709    0.0415   -0.0470    0.5126
```

Step 1: Implement PCA

Implement PCA, the matrix in which the data is expressed with respect to the eigenbasis of sigma, which is the matrix U.

```
% Compute Covariance Matrix of x (sigma).
```

```
sigma = x * x' / size(x, 2);
% Perform Singular Value Decomposition (SVD)
[U, S, V] = svd(sigma);

%%================================================================
```

Step 2 and 3: Find k and implement PCA with dimension reduction

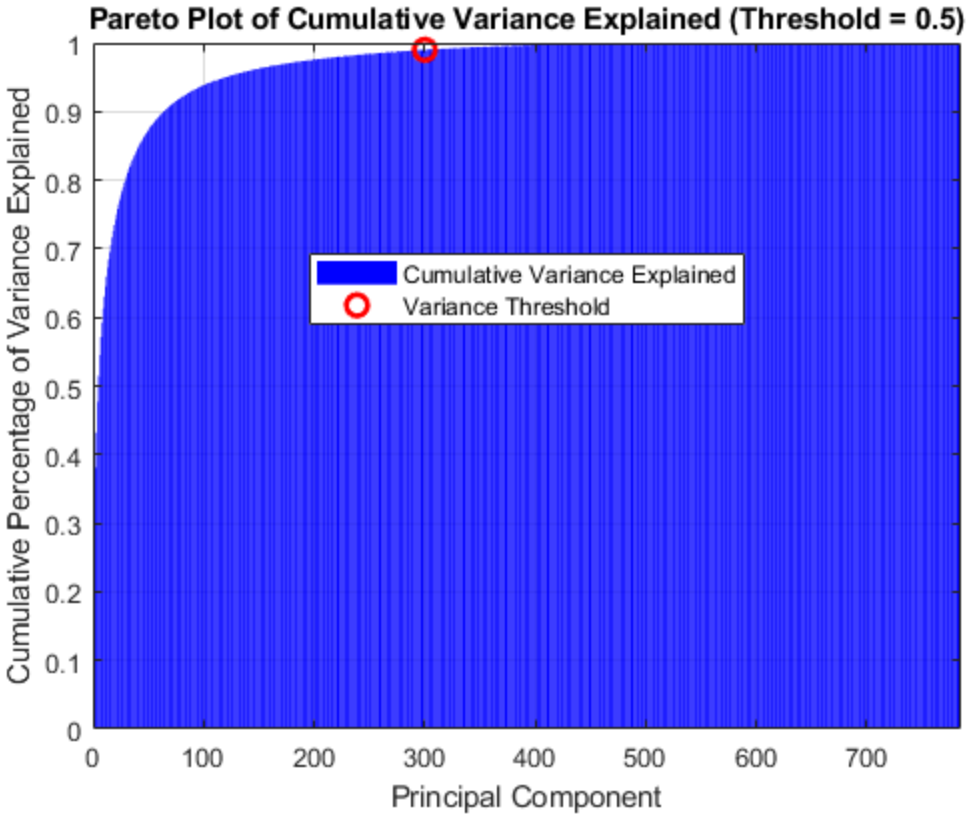
Write code to determine k, the number of components to retain in order to retain at least 99%, 80% and 50% of the variance.

Now that you have found k, you can reduce the dimension of the data by discarding the remaining dimensions. In this way, you can represent the data in k dimensions instead of the original 784, which will save you computational time when running learning algorithms on the reduced representation.

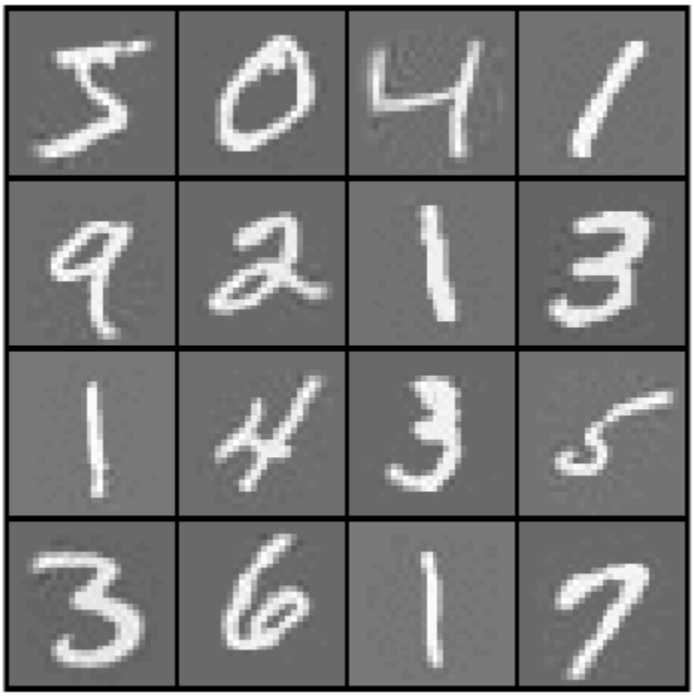
Following the dimension reduction, invert the PCA transformation to produc the matrix xHat, the dimension-reduced data with respect to the original b Visualise the data and compare it to the raw data. You will observe that there is little loss due to throwing away the principal components that correspond to dimensions with low variation.

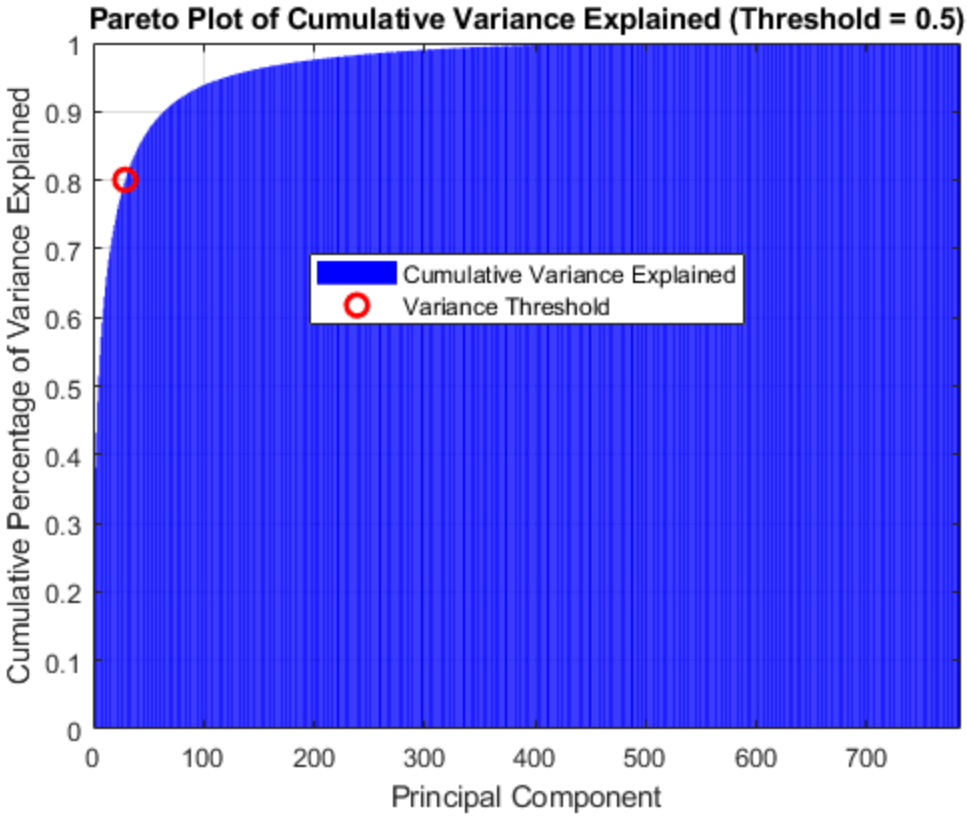
```
% Define multiple threshold values
thresholds = [0.99, 0.80, 0.50];

% Initialize a vector to store k values for each threshold
k_values = zeros(size(thresholds));

% Loop over each threshold
for i = 1:length(thresholds)
    threshold = thresholds(i);

    % Find the number of components to retain
    eigenValues = diag(S);
    cs = cumsum(eigenValues);
    cs = cs / cs(end);

    % Find the smallest k that satisfies the threshold
    k = find(cs >= threshold, 1);

    % Store the k value in the vector
    k_values(i) = k;

    % Now you can use the obtained k for further processing or
 visualization
    fprintf('For threshold %.2f, selected k = %d\n', threshold, k);
end

% Loop over each k_values
for i = 1:length(k_values)
```

```
    k = k_values(i);

    % Plot the cumulative percentage of variance explained
    figure;
    bar(1:length(cs), cs, 'b');
    hold on;
    plot(k, cs(k), 'ro', 'MarkerSize', 8, 'LineWidth', 2);
    hold off;

    % Customize the plot
    title(['Pareto Plot of Cumulative Variance Explained (Threshold =
', num2str(threshold), ')']);
    xlabel('Principal Component');
    ylabel('Cumulative Percentage of Variance Explained');
    grid on;

    % Add legend
    legend('Cumulative Variance Explained', 'Variance
Threshold', 'Location', 'Best');

    % Reduced dimension representation of the data
    xHat = U(:, 1:k) * U(:, 1:k)' * x;

    % Visualize the data, and compare it to the raw data
    first_images = (1:16)';
    figure();
    display_network(xHat(:, first_images));
    title(['PCA with dimension reduction processed images
',sprintf('(%d / %d dimensions)', k, size(x, 1))]);
end

%%================================================================

For threshold 0.99, selected k = 300
For threshold 0.80, selected k = 29
For threshold 0.50, selected k = 5
```

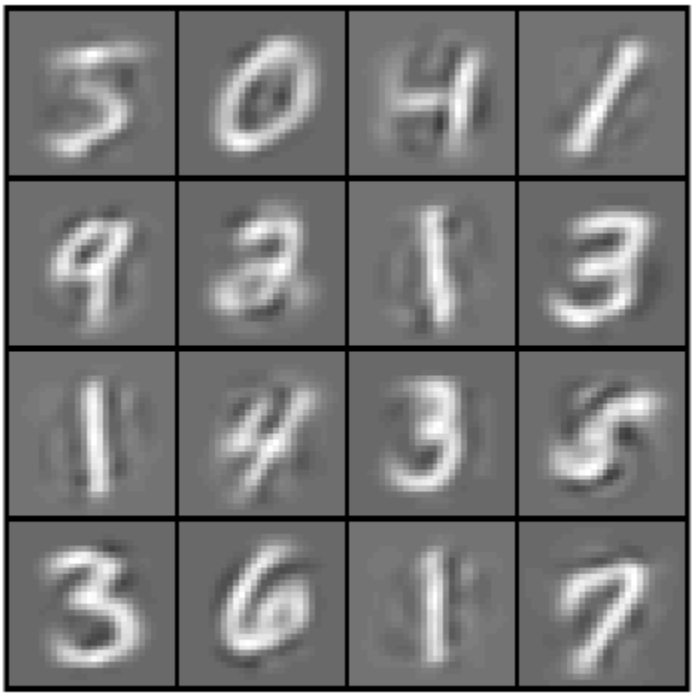


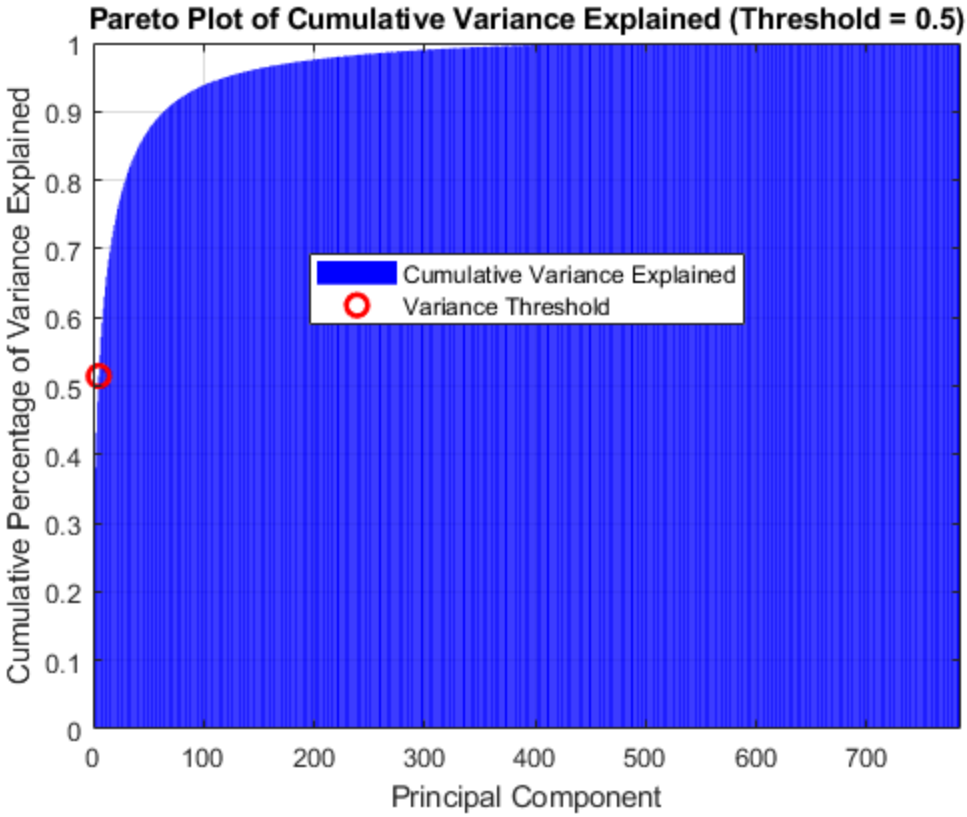
PCA with dimension reduction processed images (300 / 784 dimensions)



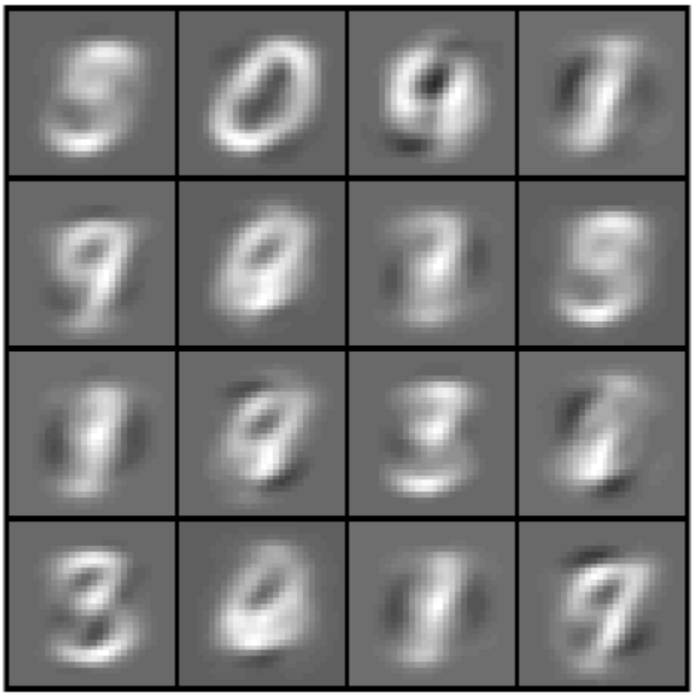


PCA with dimension reduction processed images (29 / 784 dimensions)





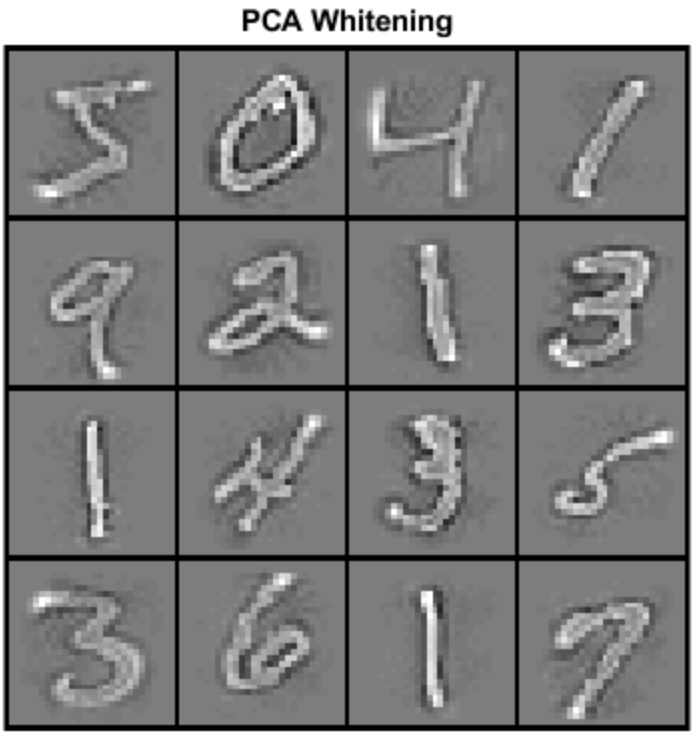
PCA with dimension reduction processed images (5 / 784 dimensions)



Step 4: Implement PCA with whitening and regularisation

Implement PCA with whitening and regularisation to produce the matrix xPCAWhite.

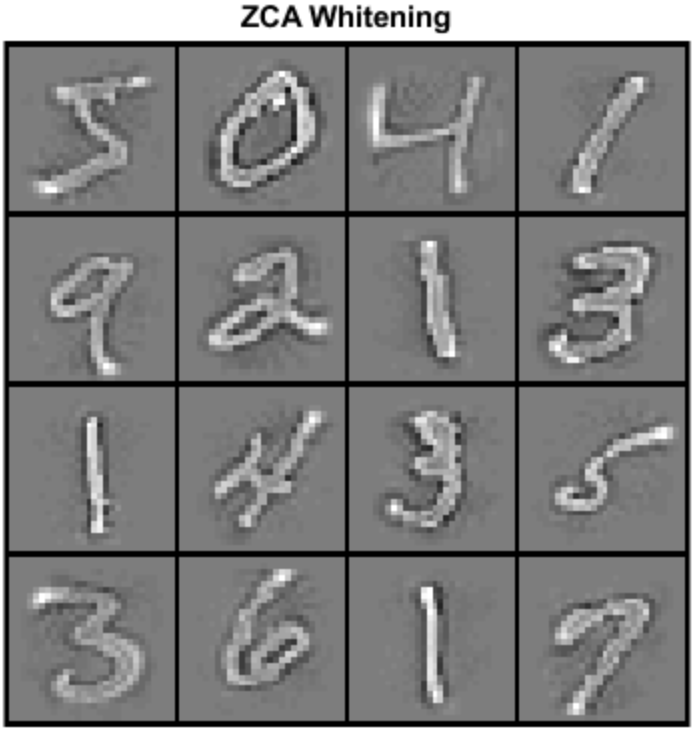
```
% Set the regularization parameter
epsilon = 1e-1;

% Assuming x is your original image data, and xPCAwhite is the PCA-
whitened data
k = 300;

% Extract the first k eigenvalues and corresponding eigenvectors
S_k = diag(diag(S(1:k, 1:k)));
U_k = U(:, 1:k);

% Perform PCA with whitening and regularization using the first k
 components
xPCAwhite = diag(1./sqrt(diag(S_k) + epsilon)) * U_k' * x;

% Reconstruct the image by transforming back to the original space
xReconstructed = U_k * xPCAwhite;

% Plot the original and reconstructed images
figure();
display_network(xReconstructed(:,first_images));
title('PCA Whitening');

%%=================================================================
```



Step 5: Implement ZCA whitening

Now implement ZCA whitening to produce the matrix xZCAWhite.
Visualise the data and compare it to the raw data. You should observe
that whitening results in, among other things, enhanced edges.

```
% Perform ZCA whitening using the first k components
xZCAWhite = U_k * diag(1./sqrt(diag(S_k) + epsilon)) * U_k' * x;

% Visualize the data, and compare it to the raw data.
% You should observe that the whitened images have enhanced edges.
figure();
display_network(xZCAWhite(:, first_images));
title('ZCA Whitening');
```



Published with MATLAB® R2019a