Visual Recognition System

Final Project Report

COEN 266 Artificial Intelligence

Winter 2019

Student ID: W1419691

Name: Akshay Borse

Table of Contents

Abstract:

We consider the problem of face recognition and human motion recognition. Instead of going for conventional L_2 − norm subspaces we use L_1 − norm subspace. In contrast to conventional L_2 − norm subspace, L_1 − norm subspaces are seen to offer significant robustness to image variations, disturbances, and rank selection. Also, instead of training a common L_1 − subspace for robust face recognition, we propose an individual L_1 − subspace face representation algorithm. As for Video recognition, we use optical flow and Canny edge detector and Keras Inception version 3 Network for feature extraction. Then use SVM classification.

Introduction:

Face recognition has been a problem of growing importance in the past decade with a wide range of application. Among different face recognition techniques available, subspace learning based algorithm has attracted significant interest. The base for subspace learning for face recognition has been the (L_2 − norm) principal component analysis (PCA), where we find the linear projection matric by maximizing the data variance in the projection subspace. The columns computed projection matrix are the so-called principal components, or features, or eigenfaces. The features extracted by PCA can be easily affected by outliers in the training data. To eliminate this problem they proposed to compute the principal components by L_1 − norm maximization.

To compute these principal components given $X \in R^{D\ X\ N}$ of N signal samples of dimension D, the computation of the L_1 − principal component of the data by maximum L_1 − norm projection is NP-hard jointly in N, D as shown. However, when the signal dimension D is fixed with D ≤ N – which is arguably of more interest in signal processing application – the L_1 principal component can be computed in polynomial time with complexity $O\left(N^{rank(X)}\right)$, rank(X) ≤ D, as presented.

They proposed a new suboptimal algorithm to compute the L_1 − principal component. The proposed algorithm for finding a near-optimal solution to the problem has two steps. (i) Obtain an initial binary vector, and (ii) perform optimal iterative single-bit-flipping (SBF) to obtain a final binary vector.

In this work, we consider the computation of "individual" L_1 − subspaces, in which an L_1 − subspace is computed for each class of face images using the training face images from that particular class. Then, classification is performed by the nearest subspace (NS) criterion, which assigns a class label to the unknown testing sample according to the subspace that most closely represents the testing sample.

Related Work (Literature Survey):

Here we consider the problem of recognizing the image. Considering an image which when represented as data would be very high dimensional.

Through all the literature survey I learned that not all the features of a data are important. So, we can perform dimensionality reduction on the image data. The most popular method of dimensionality reduction is Principal Component Analysis. To calculate PCA there are two popular ways of doing it by doing $L_1 -$ norm or by $L_2 -$ norm. Going through the first research paper it has been proven that $L_1 -$ PCA is more robust to outliers than $L_2 -$ PCA.

$$(R_{L_2}, S_{L_2}) = \arg \min_{\substack{R \in \mathbb{R}^{D\ X\ K}, R^T R = I_K \\ S \in \mathbb{R}^{N\ X\ K}}} ||X - RS^T||_2$$

Which is equivalent to following.

$$R_{L_2} = \arg \max_{\substack{R \in \mathbb{R}^{D\ X\ K} \\ R^T R = I_K}} ||X^T R||_2$$

We can see from the equation of $L_2 -$ PCA the effect of outliers is squared in dimensionality reduction. While in the case of $L_1 -$ PCA, as shown below the effect is less as we do not square.

$$R_{L_1} = \arg \max_{\substack{R \in \mathbb{R}^{D\ X\ K} \\ R^T R = I_K}} ||X^T R||_1$$

Also, the first paper states to calculate $L_1 -$ subspace for individual classes of the face image.

The results show the robustness to outliers for $L_1 -$ PCA.

The second paper proposes an optimal method for calculating L1-PCA. First, they have proven that calculating the PCA by L_1 when 'N' and 'D' are asymptotically large the problem becomes NP-hard. While if we make D constant the problem is not NP-hard. The problem can be solved in $O(N^d)$. Where 'N' is the number of samples and 'd' is the rank of the data matrix for which we calculate $L_1 -$ optimal principal component.

The third paper shows experimental results for the above-mentioned algorithms. Also, the paper shows how the algorithm can be scaled by the greedy algorithm to calculate more features instead of only one feature.

The fourth paper shows fast greedy single-bit flipping sub-optimal algorithm for fast computation of L_1 PCA in $O(N^3)$ time complexity.

We know from the 2nd paper that L_1 principal component is given by

$$r_{L_1} = \frac{Xb_{opt}}{||Xb_{opt}||_2}$$

Where

$$b_{opt} = \arg \max_{b \in \{\pm 1\}^{N \times 1}} b^T X^T X b$$

Now to efficiently identify the bit flip in a binary vector b^k at the k^{th} iteration step of the algorithm. The associated quadratic value can be algebraically written as

$$b^{k^T} X^T X b^k = Tr(X^T X) + \sum 2b_i^k \{\sum_{j>i} b_j^k (X^T X)_{i,j}\}$$

Now changing the i^{th} bit of b_i^k in b^k will change the quadratic value by

$$\alpha_i^{(k)} = \pm 4 b_i^k \{\sum_{j>i} b_j^k (X^T X)_{i,j}\}$$

Therefore, if the above equation is negative changing b_i^k to $-b_i^k$ will increase the quadratic value.

Now according to the fifth paper, human action recognition is done in two steps action representation and action classification. Now there are many approaches mentioned in paper 5 for action recognition.

1) Shallow Approaches
    a. Action Representation
        i. Holistic Representation
            1. Motion Energy Image and Motion History Image
            2. Poisson equation

3. Optical Flow Algorithm

ii. Local Representation

1. Space-Time Interest Points

b. Action Classifiers

i. Direct Classification

1. K-Nearest Neighbor
2. Bag-of-Words Model

ii. Sequential Approaches

1. Hidden Markov Model
2. Conditional Random Fields
3. Structured Support Vector Machines

iii. Space-time Approaches

1. Global Gaussian Mixture Model
2. Context-Dependent Graph Kernel

iv. Part-based Approaches

1. Constellation Model
2. Discriminative Models

v. Manifold Learning Approaches

1. Principal Component Analysis
2. Condition Random Field

vi. Mid-Level Feature Approaches

1. Hierarchical Approaches

vii. Feature Fusion Approaches

1. Multi-Task Sparse Learning Model
2. Multi-Feature Max-Margin Hierarchical Bayesian Model

c. Classifiers for Human Interaction

i. Linear Support Vector Machine

d. Classifiers for RGB-D Videos

2) Deep Architectures

a. Space-Time Networks

i. Convolutional Neural Networks

b. Multi-Stream Networks

i. Multiple Convolutional Networks

c. Hybrid Networks

i. Long Short-Term Memory

Deep networks perform better in action recognition, but shallow networks are still used. Shallow methods are easy to train and generally perform better on a small database.

Methodology:

1) Face Recognition:

- Method – 1:

1. PCA by L_1 − norm approximation

Instead of training a "common" L_1 − subspace for robust face recognition, we propose an "individual" L_1 − subspace face representation algorithm, in which an "individual" subspace is trained/calculated for each class of face images using all training samples from the corresponding class. We consider the total number of C classes. Without loss of generality, each class has N training samples that can be organized in matrix form as $X^{(j)} = x_1^{(j)}, x_2^{(j)}, \ldots, x_N^{(j)} \in R^{D \times N}$ where $j = 1, 2, \ldots, C$ is the class index and each column is a vectorized training image of class j that has D pixels. To compute the L_1 − subspace of $X^{(j)}$, we first subtract the sample-mean $\mu^{(j)} = \frac{1}{N} X^{(j)} 1_N$ from each column of $X^{(j)}$ so that the training samples are zero-centered. Then, a rank−K L_1 −subspace representation of class j is computed as follows. We calculate the first L_1 − principal component using the fast algorithm, i.e. we find

$$q_1^{(j)} = \arg \max_{q \in R^D, \|q\|_2 = 1} \left\| q^T X^{(j)} \right\|_1$$

Then, the contribution of q(j) is removed from the entire training ensemble as follows

$$X^{(j)} = X^{(j)} - q^{(j)} q^{T(j)} X^{(j)}$$

The updated training ensemble $X^{(j)}$ is used to calculate the next principal component and the representation continues until the desired number of components K is reached. The above greedy "individual" L_1 − subspace is calculated for each class $j = 1, 2, \ldots, C$ to obtain $Q^{(j)} = \left[q_1^{(j)}, q_2^{(j)}, \ldots, q_K^{(j)} \right]$.

Next, for classification we adopt a nearest-subspace (NS) approach. For each test face image x_t, we subtract the mean of the $j - th$ class $\mu^{(j)}$ from x_t. Then, the zero-centered test data point is projected onto the $j - th$ L_1 − subspace $Q_{L_1}^{(j)}$ and the reconstruction error using the $j - th$ class L_1 − subspace is calculated as $\left\| \left(x_t - \mu^j \right) - Q_{L_1}^{(j)} Q_{L_1}^{(j)T} \left(x_t - \mu^j \right) \right\|_2$. The

procedure is performed for each calculated "individual" $L_1 -$ subspace and the classification criterion is

$$\hat{j} = \arg \max_{1 \le j \le C} \left\| (x_t - \mu^j) - Q_{L_1}^{(j)} Q_{L_1}^{(j)T} (x_t - \mu^j) \right\|_2$$

2. Support Vector Machine:

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

3. Exhaustive Search

Exhaustive Search searches over all possible values to find b_{opt}

$$b_{opt} = \arg \max_{b \in \{\pm 1\}^{N \times 1}} b^T X^T X b$$

Exhaustive Search is NP-hard when both D and N are variable for $X \in R^{DXN}$

4. A neural network with Keras sequential model

The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer.

We use the 'add()' function to add layers to our model.

Our first 2 layers are Conv2D layers. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices.

64 in the first layer and 32 in the second layer are the number of nodes in each layer. This number can be adjusted to be higher or lower, depending on the size of the dataset. In our case, 64 and 32 work well, so we will stick with this for now.

Kernel size is the size of the filter matrix for our convolution. So a kernel size of 3 means we will have a 3x3 filter matrix. Refer back to the introduction and the first image for a refresher on this.

Activation is the activation function for the layer. The activation function we will be using for our first 2 layers is the ReLU or Rectified Linear Activation. This activation function has been proven to work well in neural networks.

Our first layer also takes in an input shape. This is the shape of each input image, 28,28,1 as seen earlier on, with the 1 signifying that the images are greyscaled.

In between the Conv2D layers and the dense layer, there is a 'Flatten' layer. Flatten serves as a connection between the convolution and dense layers.

'Dense' is the layer type we will use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks.

We will have 10 nodes in our output layer, one for each possible outcome (0–9).

The activation is 'softmax'. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

2) Human Action Recognition:

- Feature Extraction

  a. Method 1:

     i. Optical Flow:

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. Optical flow can also be defined as the distribution of apparent velocities of movement of brightness pattern in an image.

Sequences of ordered images allow the estimation of motion as either instantaneous image velocities or discrete image displacements. Fleet and Weiss provide a tutorial introduction to gradient-based optical flow. John L. Barron, David J. Fleet, and Steven Beauchemin provide a performance analysis of several optical flow techniques. It emphasizes the accuracy and density of measurements.

The optical flow methods try to calculate the motion between two image frames which are taken at times t and $t + \Delta t$ at every voxel position. These methods are called differential since they are based on local Taylor series approximations of the image signal; that is, they use partial derivatives with respect to the spatial and temporal coordinates.

For a 2D+t dimensional case (3D or n-D cases are similar) a voxel at location (x, y, t) with intensity $I(x, y, t)$ will have moved by Δx, Δy and Δt between the two image frames, and the following brightness constancy constraint can be given:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Assuming the movement to be small, the image constraint at $I(x, y, t)$ with Taylor series can be developed to get:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + H.O.T$$

$H.O.T. -> \ Higher\ Order\ Terms$

From these equations it follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

Or

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

which results in

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

where V_x, V_y are the x and y components of the velocity or optical flow of $I(x, y, t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x, y, t) in the corresponding directions. I_x, I_y and I_t can be written for the derivatives in the following.

Thus:

$$I_x V_x + I_y V_y = -I_t$$

or

$$\nabla I^T \cdot \vec{V} = -I_t$$

This is an equation in two unknowns and cannot be solved as such. This is known as the aperture problem of the optical flow algorithms. To find the

optical flow another set of equations is needed, given by some additional constraint. All optical flow methods introduce additional conditions for estimating the actual flow.

Methods for determination:

- Phase correlation – the inverse of normalized cross-power spectrum
- Block-based methods – minimizing sum of squared differences or sum of absolute differences, or maximizing normalized cross-correlation
- Differential methods of estimating optical flow, based on partial derivatives of the image signal and/or the sought flow field and higher-order partial derivatives, such as:
    - Lucas–Kanade method – regarding image patches and an affine model for the flow field.
    - Horn–Schunck method – optimizing a functional based on residuals from the brightness constancy constraint, and a particular regularization term expressing the expected smoothness of the flow field.
    - Buxton–Buxton method – based on a model of the motion of edges in image sequences.
    - Black–Jepson method – coarse optical flow via correlation.
    - General variational methods – a range of modifications/extensions of Horn–Schunck, using other data terms and other smoothness terms.
- Discrete optimization methods – the search space is quantized, and then the image matching is addressed through label assignment at every pixel, such that the corresponding deformation minimizes the distance between the source and the target image. The optimal solution is often recovered through Max-flow min-cut theorem algorithms, linear programming or belief propagation methods.

Many of these, in addition to the current state-of-the-art algorithms, are evaluated on the Middlebury Benchmark Dataset.

We use the Lucas-Kanade method from Differential methods for estimating optical flow.

Lucas–Kanade method:

In computer vision, the Lucas–Kanade method is a widely used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. It assumes that the flow is essentially constant in a local neighborhood of the pixel under consideration and solves the basic optical flow equations for all the pixels in that neighborhood, by the least squares' criterion.

By combining information from several nearby pixels, the Lucas–Kanade method can often resolve the inherent ambiguity of the optical flow equation. It is also less sensitive to image noise than point-wise methods. On the other hand, since it is a purely local method, it cannot provide flow information in the interior of uniform regions of the image.

The Lucas–Kanade method assumes that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point p under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at p. Namely, the local image flow (velocity) vector (V_x, V_y) must satisfy

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned}$$

where $q_1, q_2, \ldots, q_n$ are the pixels inside the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives of the image I with respect to position x, y and time t, evaluated at the point (q_i) and at the current time.
These equations can be written in matrix form $A\,v = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad b = \begin{bmatrix} -I_t(q_1) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

This system has more equations than unknowns and thus it is usually over-determined. The Lucas–Kanade method obtains a compromise solution by the least squares' principle. Namely, it solves the 2 × 2 system.
$A^T A v = A^T b$ or
$v = (A^t A)^{-1} A^T b$
where A^T is the transpose of matrix A. That is, it computes

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)\, I_y(q_i) \\ \sum_i I_y(q_i)\, I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)\, I_t(q_i) \\ -\sum_i I_y(q_i)\, I_t(q_i) \end{bmatrix}$$

where the central matrix in the equation is an Inverse matrix. The sums are running from $i = 1$ to n.

The matrix $A^T A$ is often called the structure tensor of the image at the point p.

ii. Canny edge detector:

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

Detection of edge with the low error rate, which means that the detection should accurately catch as many edges shown in the image as possible

The edge point detected from the operator should accurately localize on the center of the edge.

A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

To satisfy these requirements Canny used the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

Among the edge detection methods developed so far, the Canny edge detection algorithm is one of the most strictly defined methods that provide good and reliable detection. Owing to its optimality to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection.

Process of Canny edge detection algorithm:

1. Gaussian filter

   Since all edge detection results are easily affected by image noise, it is essential to filter out the noise to prevent false detection caused by noise. To smooth the image, a Gaussian filter is applied to convolve with the image. This step will slightly smooth the image to reduce the effects of obvious noise on the edge detector. The equation for a Gaussian filter kernel of size (2k+1)×(2k+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp\left(-\frac{\left(i-(k+1)\right)^2 + \left(j-(k+1)\right)^2}{2\sigma^2}\right)$$
$$; 1 \leq i, j \leq (2k+1)$$

2. Finding the intensity gradient of the image

   An edge in an image may point in a variety of directions, so the Canny algorithm uses four filters to detect horizontal, vertical and diagonal edges in the blurred image. The edge detection operator (such as Roberts, Prewitt, or Sobel) returns a value for the first derivative in the horizontal direction (G_x) and the vertical direction (G_y). From this the edge gradient and direction can be determined:

$$G = \sqrt{{G^2}_x + {G^2}_y}$$
$$\theta = atan2(G_y, G_x)$$

3. Non-maximum suppression

   Non-maximum suppression is an edge thinning technique.

   Non-maximum suppression is applied to find "the largest" edge. After applying the gradient calculation, the edge extracted from the gradient value is still quite blurred. With respect to criterion 3, there should only be one accurate response to the edge. Thus non-maximum suppression can help to suppress all the gradient values (by setting them to 0) except the local maxima, which indicate locations with the sharpest change of intensity value. The algorithm for each pixel in the gradient image is:

a. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
b. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the y-direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

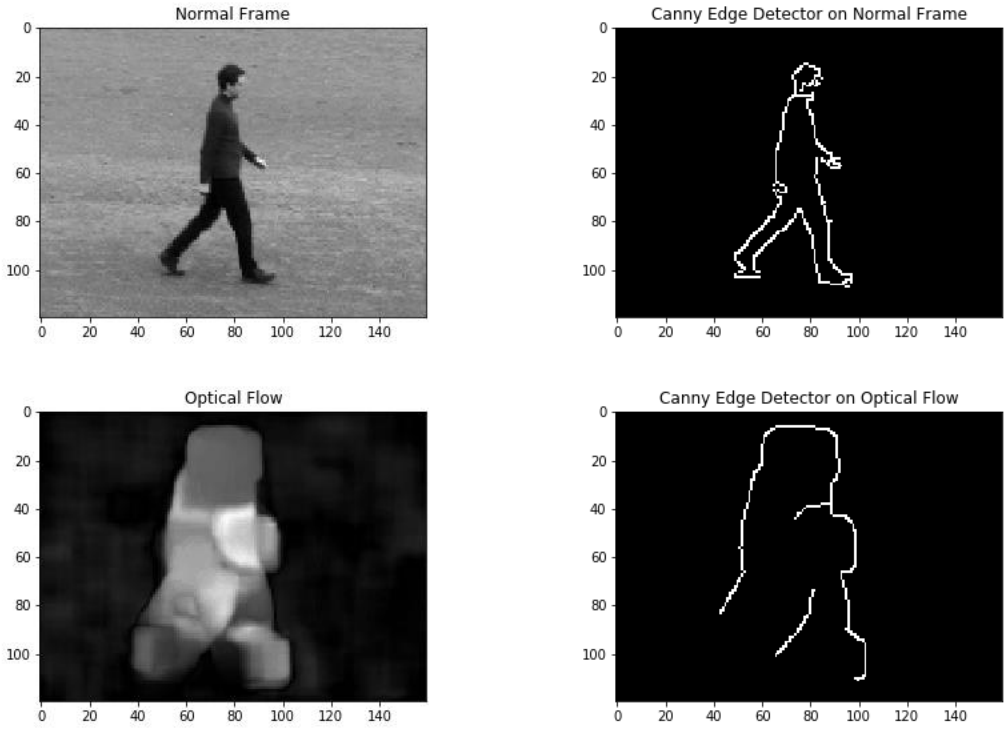
4. Double threshold

After application of non-maximum suppression, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation. In order to account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. This is accomplished by selecting high and low threshold values. If an edge pixel's gradient value is higher than the high threshold value, it is marked as a strong edge pixel. If an edge pixel's gradient value is smaller than the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel. If an edge pixel's value is smaller than the low threshold value, it will be suppressed. The two threshold values are empirically determined, and their definition will depend on the content of a given input image.

5. Edge tracking by hysteresis

So far, the strong edge pixels should certainly be involved in the final edge image, as they are extracted from the true edges in the image. However, there will be some debate on the weak edge pixels, as these pixels can either be extracted from the true edge or the noise/color variations. To achieve an accurate result, the weak edges caused by the latter reasons should be removed. Usually, a weak edge pixel caused by true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection, blob analysis is applied by looking at a weak edge pixel and its 8-connected neighborhood pixels. As long as there is one strong edge pixel that is involved in the blob, that weak edge point can be identified as one that should be preserved.

The following figure shows the output of different layers.



b. Method 2:

In the 2nd method we do feature extraction using Inception Network Version 3.

The Inception of deep convolutional architecture was introduced as GoogLeNet in (Szegedy et al. 2015a), here named Inception-v1. Later the Inception architecture was refined in various ways, first by the introduction of batch normalization (Ioffe and Szegedy 2015) (Inception-v2). Later by additional factorization ideas in the third iteration (Szegedy et al. 2015b) which will be referred to as Inception-v3 in this report.

	Network	Top-1 Error	Top-5 Error	Cost Bn Ops
Inception-v1	GoogLeNet [20]	29%	9.2%	1.5
	BN-GoogLeNet	26.8%	-	**1.5**
Inception-v2	BN-Inception [7]	25.2%	7.8	2.0
Inception-v3	Inception-v3-basic	23.4%	-	3.8
Inception-v3 + RMSProp	Inception-v3-rmsprop RMSProp	23.1%	6.3	3.8
Inception-v3 + RMSProp + Label Smoothing	Inception-v3-smooth Label Smoothing	22.8%	6.1	3.8
Inception-v3 + RMSProp + Label Smoothing + 7×7 Factorization	Inception-v3-fact Factorized 7 × 7	21.6%	5.8	4.8
Inception-v3 + RMSProp + Label Smoothing + 7×7 Factorization + Auxiliary Classifier	Inception-v3 BN-auxiliary	**21.2%**	**5.6%**	4.8

Using single-model single-crop, we can see the top-1 error rate is improved when proposed techniques are added on top of each other:

For more details on Inception network visit references.

➢ Dimensionality Reduction:

Dimensionality Reduction is done by using PCA by $L_1 -$ norm maximization similar which we used for the image.

➢ Classification:

1. SVM (Support Vector Machine):

After feature extraction with CNN. We classify the features by linear SVM. We use the same SVM model used for the image. We convert the features into a row vector and then feed it to SVM.

2. Nearest Neighbor:

In pattern recognition, the k-nearest neighbor algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

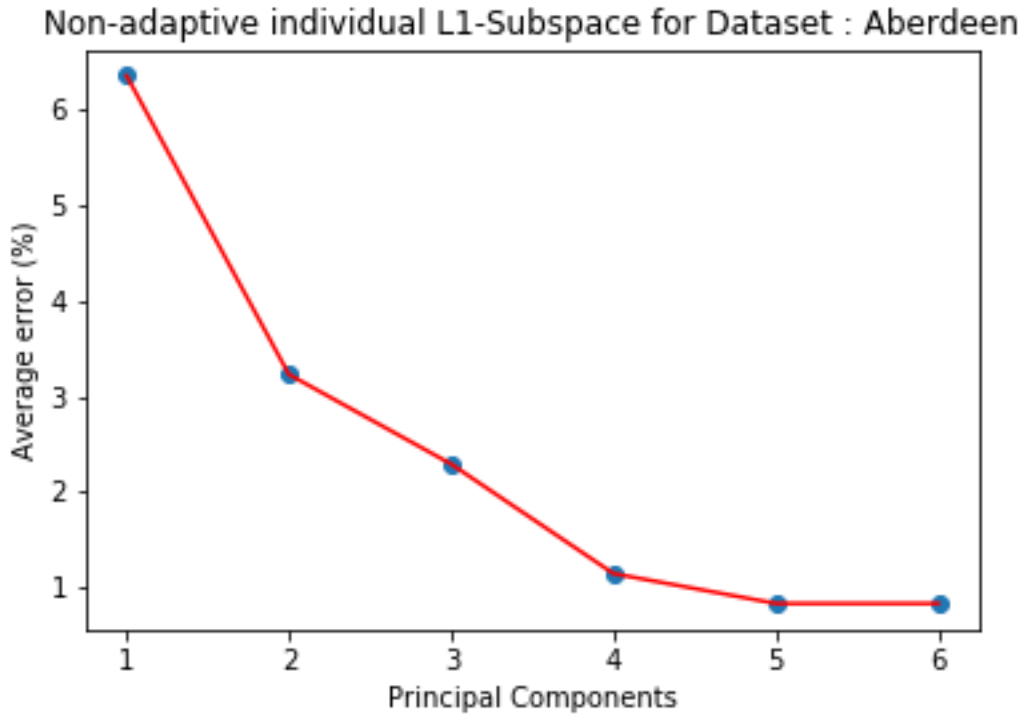
Experimental Studies:

1. Face Recognition

- Nearest neighbor and PCA by L_1 − norm maximization

a. Aberdeen Database

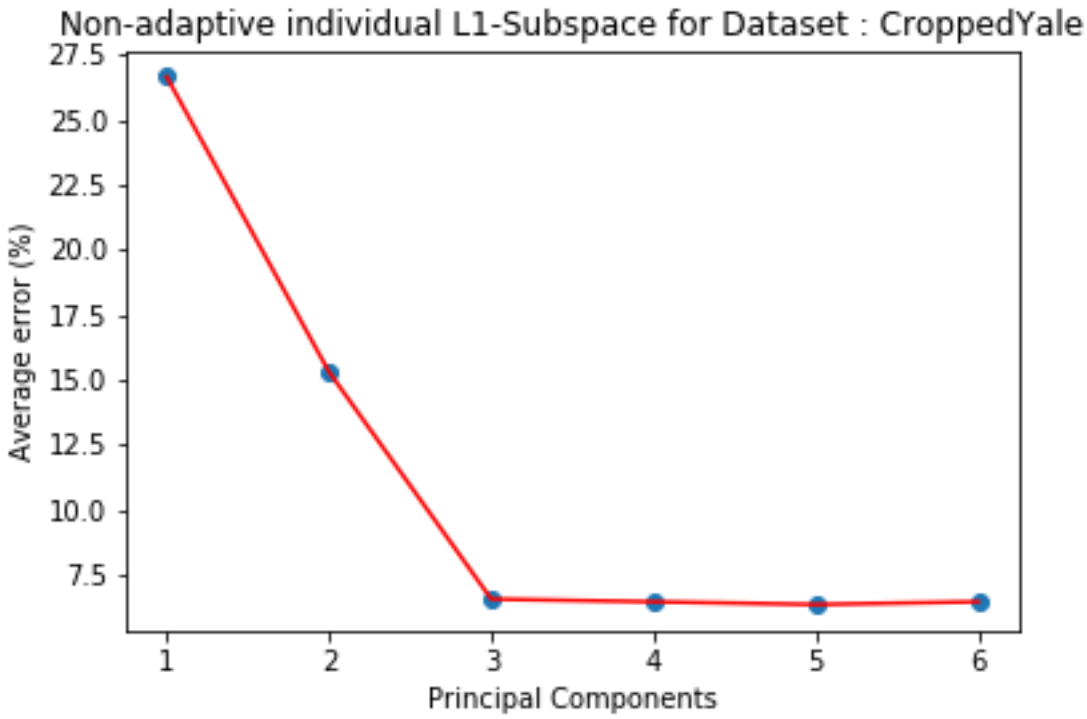
The Aberdeen database has C = 8 classes. Each class has 18 images of size 50 × 50 pixels (D = 2500).
Number of Experiment : 20
Number of Test Images for 1 experiment: 6 * 8 = 48
Total number of test images: 48 * 20 = 960
Following fig shows the recognition error rate verses the number of principal components.



b. Extended Yale Database

The Extended Yale Face database has C = 8 classes. Each class has 18 images of size 50 × 50 pixels (D = 2500).

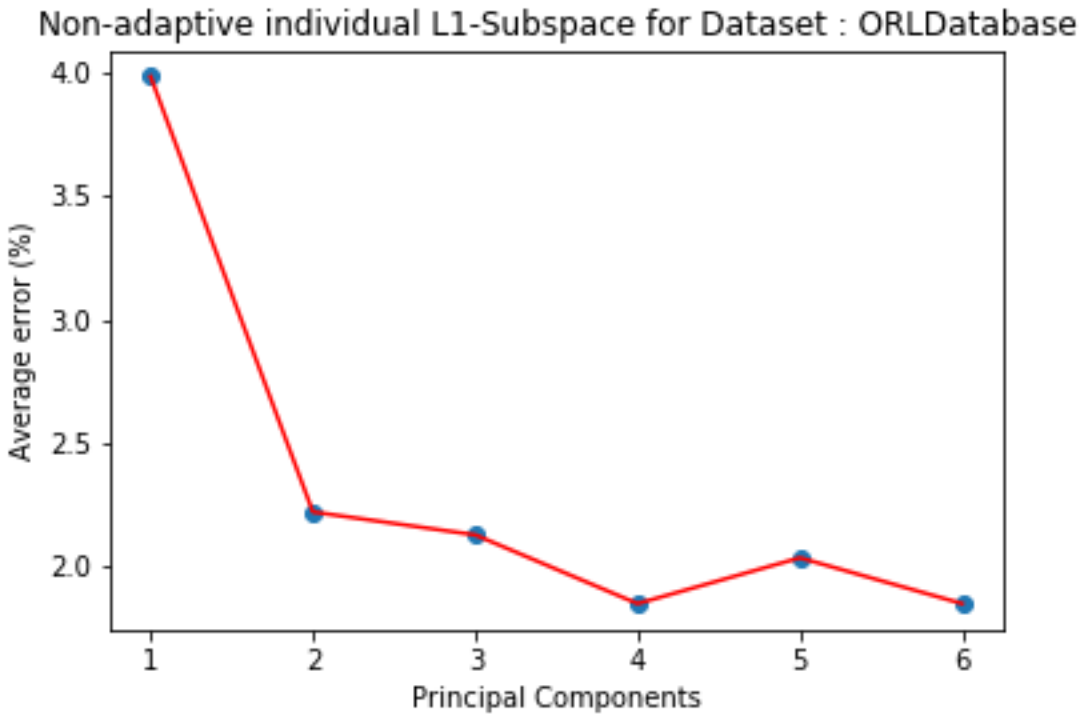
Number of Experiment : 20
Number of Test Images for 1 experiment: 6 * 8 = 48
Total number of test images: 48 * 20 = 960
Following fig shows the recognition error rate versus the number of principal components.



c. ORL Database

For ORL Database The number of images per class is only 10 so we take more classes as compared to Database

The ORL database has C = 18 classes. Each class has 10 images of size 50 × 50 pixels (D = 2500).
Number of Experiment : 20
Number of Test Images for 1 experiment: 3 * 18 = 54
Total number of test images: 54 * 20 = 1080
Following fig shows the recognition error rate versus the number of principal components.



➢ Support Vector Machine

For SVM we use the complete data set.

a. Aberdeen Database

687 faces from Ian Craw at Aberdeen. Between 1 and 18 images of 90 individuals. Some variations in lighting, 8 have a varied viewpoint.

No of experiment : 5
No of test image : 207
Total number of test images: 207 * 5 =1035
Average Accuracy: 86.00 %

b. Extended Yale Database

38 classes and around 64 near frontal images under different illuminations per class

No of experiment : 5
No of test image : 736
Total number of test images : 5 * 736
Average Accuracy: 88.00 %

c. ORL Database

ORL Database has 40 classes and 10 images per class

No of experiment : 5
No of test image : 120
Total number of test images: 5 * 120 = 600
Average Accuracy: 98.00 %

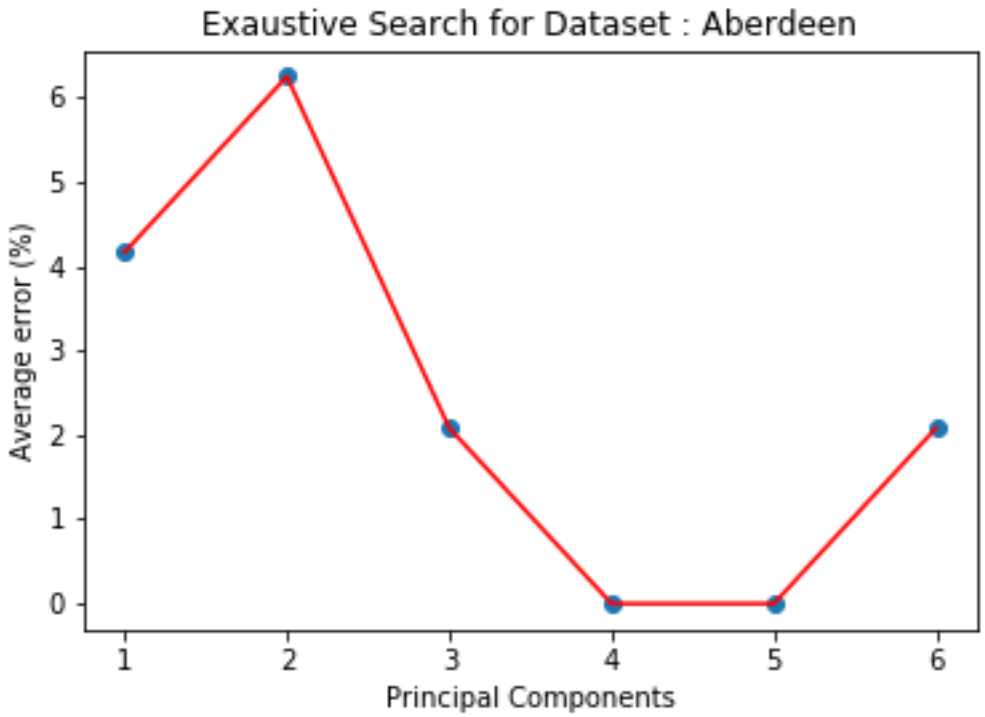
➢ Exhaustive Search

As exhaustive search takes a lot of time we just how 1 experiment to show the algorithm works.

a. Aberdeen Database

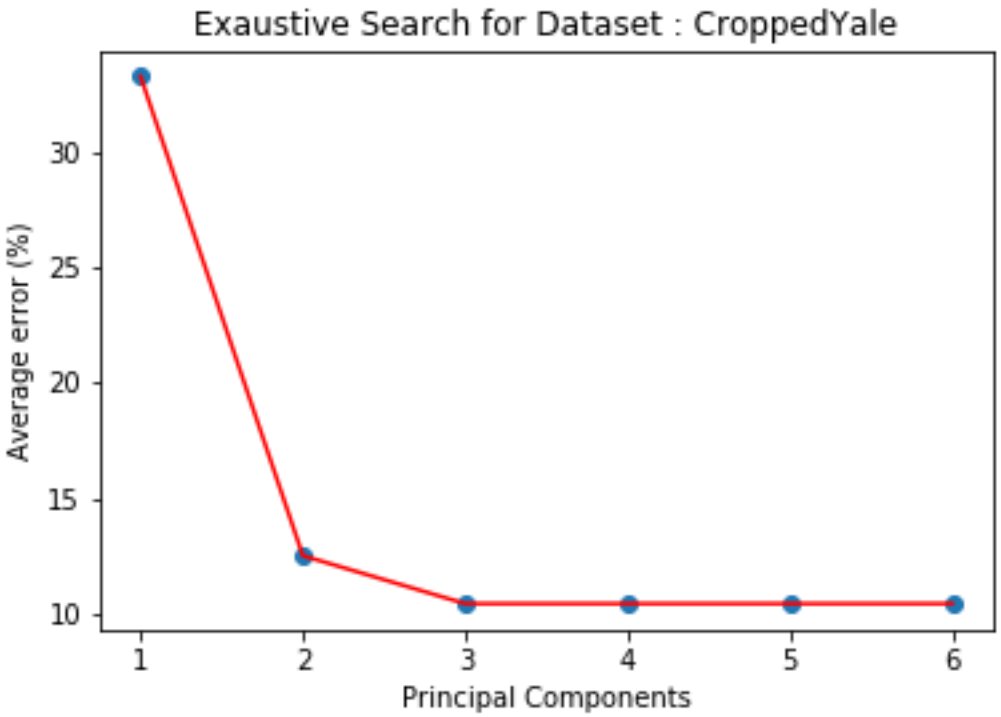
The Aberdeen database has C = 8 classes. Each class has 18 images of size 50×50 pixels (D = 2500).

As exhaustive search takes a lot of time we just how 1 experiment to show the algorithm works.
Number of Experiment : 1
Number of Test Images for 1 experiment: 5 * 8 = 40
Total number of test images: 40 * 1 = 40
Following fig shows the recognition error rate versus the number of principal components.



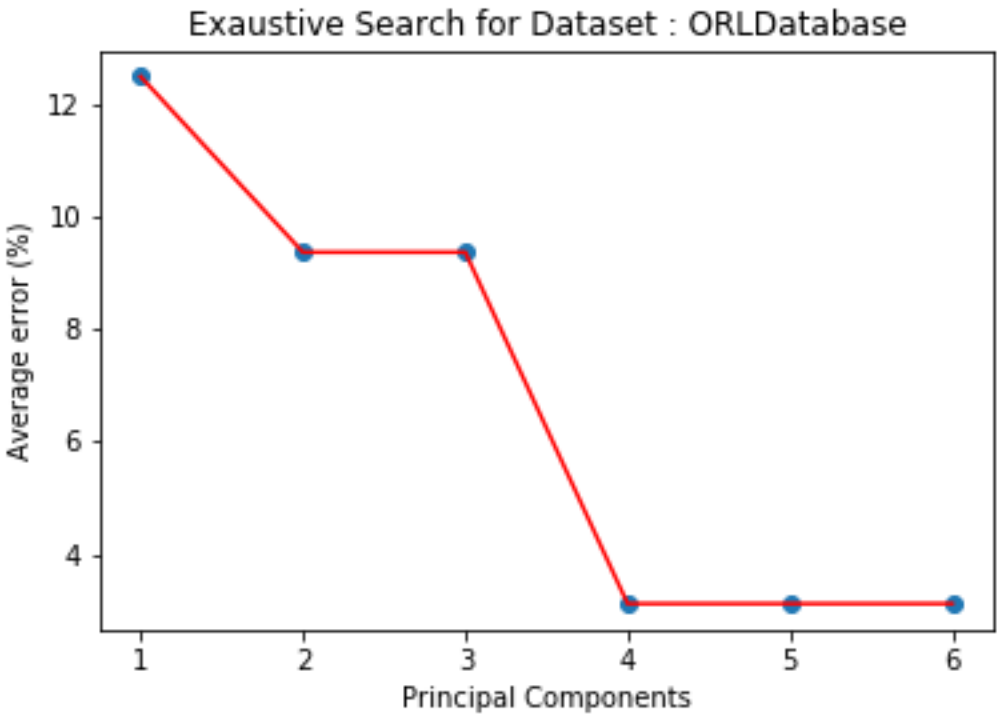
b. Extended Yale Database

The Extended Yale Face database1 has C = 8 classes. Each class has 18 images of size 50 × 50 pixels (D = 2500).
As exhaustive search takes a lot of time we just how 1 experiment to show the algorithm works.
Number of Experiment : 1
Number of Test Images for 1 experiment: 5 * 8 = 40
Total number of test images: 40 * 1 = 40
Following fig shows the recognition error rate versus the number of principal components.



c. ORL Database

The ORL database has C = 8 classes. Each class has 10 images of size 50 × 50 pixels (D = 2500).
As exhaustive search takes a lot of time we just how 1 experiment to show the algorithm works.
Number of Experiment : 1
Number of Test Images for 1 experiment: 3 * 8 = 24
Total number of test images: 24 * 1 = 24
Following fig shows the recognition error rate versus the number of principal components.



➢ Keras Neural Network

For Keras NN we use the complete data set.

a. Aberdeen Database

687 faces from Ian Craw at Aberdeen. Between 1 and 18 images of 90 individuals. Some variations in lighting, 8 have a varied viewpoint.
Epochs = 80
We require more epochs because there are fewer samples per class.
Accuracy = 78.74 %

b. Extended Yale Database

38 classes and around 64 near frontal images under different illuminations per class
Epochs = 15
We require fewer epochs because there are more samples per class.
Accuracy = 92.26 %

c. ORL Database

ORL Database has 40 classes and 10 images per class
Epochs = 50
We require more epochs because there are fewer samples per class.
Accuracy = 92.50 %

2. Human Action Recognition

a. Feature extraction by Keras CNN.

We use the keras CNN to extract features from the dataset and store them in '.npy' file to later use it.

b. SVM

After the above step, we load the data and pass the data to SVM.

➢ KTH Database

We run 10 experiments with 70% training set and 30% testing set.
The average accuracy of the system = 56.85%

➢ Weizmann Database

We run 10 experiments with 70% training set and 30% testing set.
28 testing images per experiment = 280 total testing images.
The average accuracy of the system = 35.71%

➢ Moments in Time (Unconstrained Database)

We run 10 experiments with 70% training set and 30% testing set.
162 testing images per experiment = 1620 total testing images.
The average accuracy of the system = 57.45%

c. Optical Flow → Canny Edge Detector → L_1 −PCA → Nearest Neighbor/SVM

In this method, we calculate the optical flow of the video. To minimize the calculation, we only take those frames which contain human. These frame numbers are given in 00sequence.txt file provided with the dataset.

After this, we run the edge detection algorithm and compute only those points which are moving.

Then we flatten the data to a row vector.
Now we perform L_1 − PCA to reduce the size of the data.
Then we try to find the nearest neighbor.
Due to time constraint, I could only run on KTH dataset Following are the results for it.

Nearest Neighbor → Accuracy → 2 %
SVM → Accuracy → 50 %

It seems that the nearest neighbor doesn't work.

d. Feature extraction without CNN

In this method we just first few frames of the data with respect to fps so that we can capture the motion, but we only do it once. So, if the human repeating the same action n number of time we pick only 1 action among those n.

e. Keras Neural Network

Here we use keras NN which we used for image classification. So we give each frame the same label and then try to train the model and then similarly test it on the testing set.

- KTH Database

  Accuracy → 44.65%
  Epochs → 10

- Weizmann Database

  Accuracy → 34.78%
  Epochs → 10

- Moments in Time (Unconstrained Database)

Accuracy → 53.75%
Epochs → 10

f. Keras Convolutional Neural Network

In this method, I have tried to use a CNN which is generally used for mnist dataset. I modified the model to work with video data and following are the results I get.

- KTH Database

  Accuracy → 67.54%
  Epochs → 10

- Weizmann Database

  Accuracy → 26.08%
  Epochs → 10

- Moments in Time (Unconstrained Database)

  Accuracy → 44.16%

  Epochs → 10

g. Comparison:

While certainly my algorithm was not as good as the other algorithms which people have implemented. I researched a lot of things but wasn't able to implement the code.

The best algorithm I found so far was following.
Feature extraction using Space Time Interest Points.
Space time Interest Points can be calculated by log of ways. Among them histogram of oriented gradient (HOG) and histogram of optical flow(HOF) outperforms most other algorithms.
HOF looks for the variation in the time dimension and HOG looks for the variation in space dimension thus reducing the video to very low dimension. Which is really helpful because anyways we use PCA for dimensionality reduction while HOG and HOF do all the things in single iteration. Also, it doesn't require a mean centered data.
Other algorithm was Scale Invariant Feature Transform (SIFT) which is used for video and images which are skewed or rotated along any axis this noise is reduced by SIFT.

Deep learning algorithm have best accuracy when dealing with video the models like RCNN or LSTM give you the best results.

Conclusion:

Face Recognition:

From the methods I have implemented, it seems that PCA by L_1 −norm maximization outperforms every other method and is even faster than other methods. Based on this we can say that there is a lot of noise in the data since after reducing to up to 6 components from 2500 we get almost perfect results we can a lot of the image features are not useful in classification.

Human Action Recognition:

From all the methods I have implemented the Keras neural network and SVM seems to give good results. Due to time constraint and less knowledge I couldn't completely explore the action recognition topic. More research is required to comment on this topic.

References:

Inception Network

Support-vector machine

Keras Neural Network

Optical Flow(HOF)

Lucas–Kanade method

Histogram of Oriented Gradients

Canny edge detector

Aberdeen Database

Yale Database

ORL Database

Face Recognition with L1-norm Subspaces