L_1 PCA Face Recognition Classification and Action Recognition

Nathan Hutchison
3/15/19

Abstract:

This report gives a solution the problem of face recognition classification by using L_1-norm projection. A unique solution is given where face class tested is given its own L_1 norm subspace. The final class of a test image is predicted after the test image is projected onto the subspace of each class, and then choosing the nearest subspace of that projection. L_1-norm subspaces show strong robustness to outliers images as compared to L_2-norm subspaces and the used method showed very promising results for facial classification across different datasets. After that this paper outlines how to adapt the same approach for action recognition by using motion history image still frames from a video and a similar classification process. Experimental results are shown and further ideas are given for future studies to test.

1. Introduction

Since the beginning of the twenty first century, face recognition has become an area of great focus. With applications ranging from security at events or on personal devices to medical prognosis, machine learning and facial recognition have a great potential to help society. There is a vast amount of research and studies in the area and many different approaches to the problem as outlined in the related works. Principal component analysis (PCA) has been often used in facial recognition problems since it is a fairly straightforward approach. A linear projection matrix can be found by maximizing the data variance along a projection subspace. Each column in the matrix corresponds to the principal components. L_2-norm PCA is more easily computed than L_1-norm PCA and it has been thought that L_1-norm was the superior method, but more recent studies show otherwise[1,2,3,4]. L_1-norm PCA has been shown to be much more robust to outlier data, which classification of faces with variance such as uneven lighting or noisy data[1,3].

This paper will discuss the applications of individual L_1 subspace projects as described in [1] and further extend the work to action recognition from videos. An L_1 subspace is created from each class of face images used in the training of the algorithm. In order to classify the test images, each unknown test image is then projected onto every class subspace and a prediction is made based on minimizing the reconstruction across each subspace. In order to adapt this method to action recognition from videos, still images are needed from each video. However, just taking a still frame of the video would loses information contained in the video through

time. In order to work around this issue, a motion history image process is applied to the frames analyzed for each class in order to reconcile information through time in a still frame.

2. Related Work

The essay [1], "Face Recognition with L_1-norm Subspaces", describes a unique approach to facial recognition problems. It describes how L_1-PCA algorithms are more robust in terms out handling outliers in sets of facial recognition data. The methodology was the basis of the algorithm for facial recognition used in this paper.

It goes into the details of L_2-PCA algorithm. It defines *N* training images of size *m* x *n*, vectorized as $x_i \in R^D$, with *D* = *mn*, and *I* = 1, …*N.* Then, each data x_i is vectorized and entered into a matrix $X \in \mathbb{R}^{DxN}$. L_2-PCA then finds a projection of the matrix into a smaller (rank-*K*) dimension while minimizing the sum of the element wise-squared error of the original matrix and the new rank-*K* representation:

$$(\mathbf{R}_{L_2}, \mathbf{S}_{L_2}) = \arg \min_{\substack{\mathbf{R}\in\mathbb{R}^{D\times K}, \mathbf{R}^{\mathrm{T}}\mathbf{R}=\mathbf{I}_K \\ \mathbf{S}\in\mathbb{R}^{N\times K}}} \|\mathbf{X} - \mathbf{R}\mathbf{S}^{\mathrm{T}}\|_2,$$

$$\mathbf{R}_{L_2} = \arg \max_{\substack{\mathbf{R}\in\mathbb{R}^{D\times K} \\ \mathbf{R}^{\mathrm{T}}\mathbf{R}=\mathbf{I}_K}} \|\mathbf{X}^{\mathrm{T}}\mathbf{R}\|_2.$$

It then goes over how using L_1-norm instead of L_2-norm runs into a few problems, because finding D, N in

$$\mathbf{R}_{L_1} = \arg \max_{\substack{\mathbf{R}\in\mathbb{R}^{D\times K} \\ \mathbf{R}^{\mathrm{T}}\mathbf{R}=\mathbf{I}_K}} \|\mathbf{X}^{\mathrm{T}}\mathbf{R}\|_1.$$

is NP-hard. But by fixing D, one is able to create an optimal algorithm. When looking for a solution for a single principal component the paper shows a solution:

$$\mathbf{r}_{L_1} = \arg \max_{\mathbf{r}\in\mathbb{R}^D, \|\mathbf{r}\|_2=1} \left\|\mathbf{X}^{\mathrm{T}}\mathbf{r}\right\|_1$$

$$\mathbf{r}_{L_1} = \frac{\mathbf{X}\mathbf{b}_{\mathrm{opt}}}{\|\mathbf{X}\mathbf{b}_{\mathrm{opt}}\|_2}$$

The equation can be solved by iteratively flipping the bit *b* in order to find the *b* which most negatively contributes to the L_2 projection energy the most in the equation by solving for:

$$\mathbf{b}_{\mathrm{opt}} = \arg \max_{\mathbf{b}\in\{\pm 1\}^N} \mathbf{b}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{b}.$$

In order to find multiple principal components, the L_1 -PCA algorithm works by removing the first L_1 principal components r_1 from the data sample using and updating by solving:

$$\mathbf{x}_i^{(update)} = \mathbf{x}_i - \mathbf{r}_1 \mathbf{r}_1^{\mathrm{T}} \mathbf{x}_i$$

Finally, the procedure repeats until k L_1 prinicpal components $r_1 \ldots r_k$ are found.

The novelty of this paper is how it does predicted classification on test images. It does so by finding an individual L_1-subspace for each class of face image in a dataset. It organizes the data matrix $X^{(j)} = \left[x_1{}^{(j)}, \ldots\ x_n{}^{(j)}\right] \in\ \mathbb{R}^{D\mathrm{x}N}$ with each j = 1, 2, … C corresponding to each class index and point $x_i{}^{(j)}$ refers to a vectorized training image of class j. It then zeros out the mean of each $\mu^{(j)}$ from each column of $X^{(j)}$ and minimizes the reconstruction error and predicting which class x_t is by solving the eqation below for j-hat

$$\widehat{j} = \arg \min_{1 \le j \le C} \left\| (\mathbf{x_t} - \mu^j) - \mathbf{Q}_{L_1}^{(j)} \mathbf{Q}_{L_1}^{\mathrm{T}(j)} (\mathbf{x_t} - \mu^j) \right\|_2$$

This algorithm is the basis for the process used in this paper. Similarly, the algorithm used finds a subspace and a mean for each face class or action class and uses that to classify the new images. The exact process used will be outlined in the methodology section in part 3.

[1] also shows an extension of this method that uses a fixed number of components k across all of the classes and gives an optimal solution to that problem.

Essay [3], “Principal Component Analysis Bases on L_1-Norm Maximization” discusses the differences between PCA based on L_2 norm, L_1 norm, and also goes into detail about a proposed algorithm R1-PCA, which combines parts of L_2-PCA and L_1-PCA. This method is rotational invariant while also suppressing the effect of outliers, but is much prone to variation depending on the dimension m. It is also an iterative method, so it takes a long time to converge in large subspaces.

It brings up the proposed new algorithm it will solve, PCA- L_1 and defines the error functions and continues to then show how to minimize them. It starts explaining the details behind PCA- L_1 and shows how it can find a local maximum and proposes methods to alter it in order for it to find a global maximum. It further extends the algorithm to extract multiple features from the L_1 objective function.

Finally, it talks about the results of the proposed algorithm by testing the proposed method PCA- L_1 and compares it to R1-PCA, and L_2-PCA. It starts comparing them on a fabricated dataset to show how their method is comparable to R1-PCA in suppression of outlier data. It then tried all three methods on various UCI datasets, and found that PCA- L_1 performed better than L_2-PCA and R1-PCA on datasets when the number of extracted features was small. It also took less computational cost in most cases, bar the one with a much larger number of samples.

They then tried these methods with the Yale face database, which had 165 images of 15 face classes. Each class had varying levels of obstruction. They found that PCA- L_2 generally converged must faster than R1-PCA, while still being robust to outliers like R1-PCA is.

Essay [4], "Fast computation of the L_1-Principal Component of Real-Valued Data" brings up [2] and how they offer an optimized solution, but also brings up that it has limitations when large data dimensions D or data N. It suggests that the aforementioned implementation is unsuitable because its exponential size in D complexity. It proposes a greedy single-bit-flipping iterative algorithm to find L_1-principal component of complexity $O(N^3)$.

It discusses the current L_1-maxiumum projection principal component and try to solve one of the optimal solutions to

$$\mathbf{r}_{L_1} = \underset{\mathbf{r}\in\mathbb{R}^{D\times 1},\ \mathbf{r}^T\mathbf{r}=1}{\arg\max} \left\|\mathbf{r}^T\mathbf{X}\right\|_1 .$$

It mentions how [2] created an optimal solution to the problem with a fixed D in time $O(N^{rank(\mathbf{X})})$ but quickly brings up that for moderately large values of $rank(\mathbf{X}) \leq D$ makes for a very impractical implementation. They then suggest a low-complexity algorithm in order to calculate the L_1-principal component which has a computational complexity of $O(N^3)$.

Finally, they test the algorithm. They try it with numerical evaluation, data dimensionality reduction, and direction-of-arrival estimation. They come up with similar conclusions from before, that L_1-principal component performs better with outliers than L_2 principal component in most metrics. However, this paper outlines a substantial improvement to the algorithm in [2] by making it much computationally effective.

In [5], "Human Action Recognition and Prediction: A Survey" takes a different look on computer vision by switching to video recognition and analysis. This paper goes over novel approaches to video action recognition

There are two major topics in video analysis with regard to computer vision. First is action recognition, which focuses on accurately predicting what a human has done after the fact and analyzing an entire video. Next is action prediction is an guess on what the human will do before they have acted, which basically focuses on the future state of the incoming video which is less related to this paper.

Actions are difficult to represent in still images in a video, because different angles, positions, poses, and motion are difficult to capture. It mentions hand-crafted methods to tackle some of these issues. It brings up how MEI is working on encoding dynamic motion into still images, and it can further be expended into videos through "'so-called' optical flow algorithms" [5]. It also mentions analyzing local representations, and feature trajectory to obtain short and long duration information respectively.

It then discusses action and interaction approaches using shallow architecture. Once it recognizes the action, a computer must determine an action classifier, a problem that can be broken up into multiple categories. Direct classification tries to directly recognize actions using already established classifies such as support vector machine, k-NN, etc. Next, a sequential approach can be taken since they treat the video as a composition of temporal segments or

frames. Temporal segments and compressed video frames are the idea behind the action recognition algorithm used in this paper. Current models that do this are hidden Markov models and structured support vector machines. However, these methods do not perform well with much background noise.

[5] also mentions a few other approaches, such as space time approaches that take into account the difference between space and time, but are right now limited to small datasets. Part based approaches take into consideration the action of both the human as a whole as well as the motion of individual body parts. Manifold learning approaches aim to take as much information from the human silhouette of the video as possible and have shown novel high success rates, but are limited because they will not perform well with many real world scenarios. Mid level feature approaches are robust to background noise, but have trouble expressing the level of detail that are needed to distinguish small variations. Finally, feature fusion approaches take multiple features from the same video because the data generated are inter-related, but are very convoluted.

3. Methodology

3a. PCA face recognition

This paper uses the per class individual L_1 -subspace classification method created from [1]. The mathematics of why the approach works is seen above in the related work. However, implementing this mathematical approach required a lot of manipulation to process the data in the correct fashion. This section of the paper looks to explain the process of how the algorithm was implemented.

In order to first get all the information the program needs set up to predict images, it runs the function loadFaces from imageLoader.py. This process roots through the directories to find the files it works with. It then grabs a random selection of images and resizes the data to a smaller format to prevent overengineering of features while also make the computations significantly faster. It flattens the image to get the photo data into a format $X^{(j)} = \left[x_1{}^{(j)}, \ldots\ x_n{}^{(j)}\right] \in \mathbb{R}^{D\mathrm{x}N}$ with each $x_n{}^{(j)}$ a vectorized data sample of class j. It then randomly splits the data into a training data and test data to be saved for later.

It finds the mean of the data, saves the mean, and then zero-means the training data. It then passes the data to runPca() to run through the PCA implementation. This implementation first uses function l1pca() which is an iterative bit flipping algorithm to find b_{opt} as in the equation below:

$$\mathbf{b}_{\text{opt}} = \arg \max_{\mathbf{b} \in \{\pm 1\}^N} \mathbf{b}^{\mathrm{T}} \mathbf{X}^{\mathrm{T}} \mathbf{X} \mathbf{b}.$$

Once b_{opt} has been found, l1pca() finds the principal component vector r_{L_1} in the implementation) by multiplying the original data vector X by b_{opt} and dividing it by the L_2 norm of the same product, and saves that value to r_{L_1}. This implements the below equation:

$$\mathbf{r}_{L_1} = \frac{\mathbf{X}\mathbf{b}_{\text{opt}}}{\|\mathbf{X}\mathbf{b}_{\text{opt}}\|_2}$$

l1pca() returns r_{L_1} to runPca() as the variable Qprop. Next it removes that principal component's contribution from the test data by taking (Qprop)(Qprop)^T. This product is then multiplied by the data sample, and then it updates the dataset by subtracting the total product from the original dataset. This runPca() implements the following equation into code:

$$\mathbf{x}_i^{(update)} = \mathbf{x}_i - \mathbf{r}_1\mathbf{r}_1^{\mathrm{T}}\mathbf{x}_i$$

It repeats this process until enough components k, or r_1 , …r_k are found and stored in a variable. This whole matrix of values r_1 , …r_k are stored for each class and then put into list QpropList. This process finds the Qprop, class mean and test data for each data class. The process is then run across every class of the data to be tested.

After it runs loadImages(), it gets the list of the organized test data (=testData) as well as the list of class unique principal components (= QpropList) and list of class means (=classMeanList) are returned. From here, it will start to predict the class of each test image by projecting each test image from testData onto the individual class subspaces. This is done in the function imageClassifier(). This function then iteratively applies the following process for a training image across each class:

1. Zeros out the class mean from the test image.
2. Finds out ${Q_{L_1}}^{(j)}{Q_{L_1}}^{T(j)}$. Note: ${Q_{L_1}}^{(j)}$ is $(D\ x\ k)$, with k= number of principal components and D = number of data features. No matter how many principal components K are tested, this product will always yield a $(D\ x\ D)$ matrix.
3. Takes the mean centered data and subtracting from it the product of the Q term ${Q_{L_1}}^{(j)}{Q_{L_1}}^{T(j)}$ with the mean centered data.
4. Next, it takes the L_2-norm of this whole term to find the reconstruction error j-hat for that class.

It then finds this value for each class and it returns the minimum value as the predicted class. This whole process implements the below equation:

$$\widehat{j} = \arg\min_{1\le j\le C} \left\| (\mathbf{x_t} - \mu^j) - \mathbf{Q}_{L_1}^{(j)}\mathbf{Q}_{L_1}^{\mathrm{T}(j)}(\mathbf{x_t} - \mu^j) \right\|_2$$

To be more computationally efficient, this process is repeated across each test data sample for varying levels of k. Each image in testing data is given 6 image predictions, one for each value of k it runs through the above proves. The prediction values j-hat are all returned and stored.

The program takes these predicted class values and checks whether or not the value is correct. For each evaluation, it records how many of the test images are predicted correctly across each component k for a set of training data. It then finds the percentage of test images corrected labeled for each component k.

The program is set up to run this entire process using definable variables for how many K components to use and how many evaluations per component. Each run through it uses a different selection test images, train images, and sometimes classes (different selection of classes were used in datasets the total number of classes was more than number of classes tested on).

3b. Motion History Image Action Recognition

A unique approach to compressing videos is taken in order to solve action recognition problems. The algorithm works by creating a set of motion history images for each action, or class, to represent movement in time from the video in a still image. The program then saves these images and runs the same process as it did for the facial representation but instead on the video images

Figure 1: Sample of MHI images created from the Weizman video Dataset

The motion history image, or MHI depicts an action occurring over time by fading movement through shades of white to black on the image as seen in figure 1. The implementation utilizes motion templates from the Python software OpenCV. The process of creating the MHI can be broken down into 4 steps. First, it takes the grey scale difference between two images. Second, it finds the places the images are different and creates a mask of the movement between the two frames. Next, it updates the motion history of the video by comparing the mask and the old motion history. Finally, it grabs and saves a subset of the motion history over a set amount of frames. The program reads through each frame until the end of a video. The following four steps are used on each video frame after the first.

For step one, take the last two frames of the video and take the absolute value of the pixel values across the entire image. After that, make the difference found greyscale. This is done in order to have one greyscale channel keep track of the difference of the images instead of three RGB channels each keeping different values.

The second step creates a motion mask of the difference in the videos. For this step, the program uses the function cv2.threshold(), which is a function that uses a defined threshold in order to mask an image into black and white channels. It takes the greyscale difference and

changes all of the pixels to either maxval, or white, if they are above the threshold limit or black if they are not. It uses the below equation to then save this motion mask to a destination dst:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Third, the program updates the stored motion history of the video by using cv2.updatemotionhistory() and the newest motion mask. The function keeps track of the different changes over time and saves it. It takes a silhouette, or mask image, and updates the motion history in the following manner:

$$\texttt{mhi}(x,y) = \begin{cases} \texttt{timestamp} & \text{if } \texttt{silhouette}(x,y) \neq 0 \\ 0 & \text{if } \texttt{silhouette}(x,y) = 0 \text{ and } \texttt{mhi} < (\texttt{timestamp} - \texttt{duration}) \\ \texttt{mhi}(x,y) & \text{otherwise} \end{cases}$$

The three cases do the following:

1. The stored motion history adds the current mask to the image
2. The motion history fades parts of the image if enough time has passed
3. Stays the same

Finally, the fourth step involves saving the MHI. A section of the motion history is copied, clipped, and then saved into a finished MHI as seen in the pictures above. This entire method is used in order to track the changes over time and compress them into still images. Once the MHI is created, it is saved into the a folder in the directory, and the same L_1-PCA mentioned above is used to classify the images.

4. Experimental Studies

4a. L_1pca Face Recognition Results

The L_1 PCA face recognition algorithm presented was tested on the CroppedYale[9] dataset as well as the ORL dataset[10]

Yale CroppedYale Dataset:

The aforementioned L_1 PCA algorithm was tested on the CroppedYale Dataset. 40 Experiments were run and the results for each component were the combined average of all 40 random experiments. 8 classes were randomly selected from the 39 possible classes and tested on each experiment. Extremely dark images which could not be recognized by a human were manually removed from the dataset for testing. The results of the test can be seen in figure 2 the exact data can be found in figure 3.

The results from testing this dataset shows how when more principal components are used to more accurately create a subspace for the class, accuracy improved. The average accuracy for a component increases as the number of components increases for every value K=1, …K=6. The highest

ORL Database:
The same method was applied to the ORL database photos. 40 experiments were run and the results for each component were the combined average over all 40 experiments. 8 classes were randomly selected from the 10 possible classes. No images were removed for testing. The results can be seen in figure 2 and figure 4.

Similarly to the CroppedYale dataset, as the number of principal components went up, the average accuracy for testing went up. While the movement was less pronounced, this is consistent with the work from [1] as well as the previous CroppedYale Dataset.

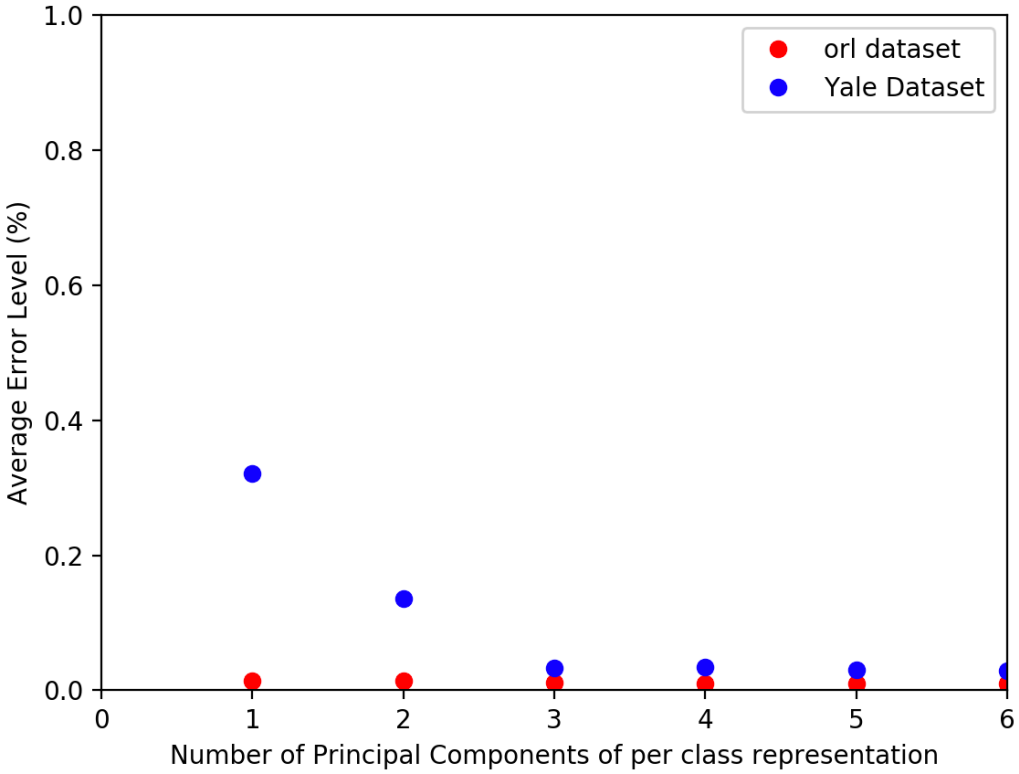


Figure 2: Testing results for L_1 PCA on Yale and ORL datasets

Number of components k:	Accuracy:
1	0.67891
2	0.86484
3	0.96641
4	0.96602
5	0.96953
6	0.97149

Figure 3: CroppedYale exact results table

Number of components k:	0.98542
1	0.98542
2	0.98542
3	0.99062
4	0.99062
5	0.99062
6	

Figure 4: ORL exact results table

4b.Motion History Image Action Recognition Results

The MHI action recognition was tested on videos from the Weizman[6] and KTH[7] datasets

Weizman Dataset:

MHI images were extracted and saved from all of the videos from the Weizman dataset. To create these images, various hyperparameters were tested in order to see how they affected accuracy. The variables tested were MHI_Duration and threshold. MHI_Duration keeps track of how many frames back to remember in the video history. Threshold determines how much movement is needed between frames to be considered a change. Examples of the same frame with various MHI_Duration and threshold levels is seen in figure 5. Frames only started being extracted from the videos once a full MHI_DURATION duration had passed. From there, MHIs were saved every frames that were equal to the MHI_DURATION apart from one another to avoid overlap from the photos.

Testing classes for each experiment all came from the MHIs with the same MHI_Duration and threshold. Frames were grabbed from every video in the dataset until the video ended or 7 different MHIs were saved from a video. 40 experiments were run and there were 6 different action classes. Various levels of testing can be seen in figures 6, 7, and 8.

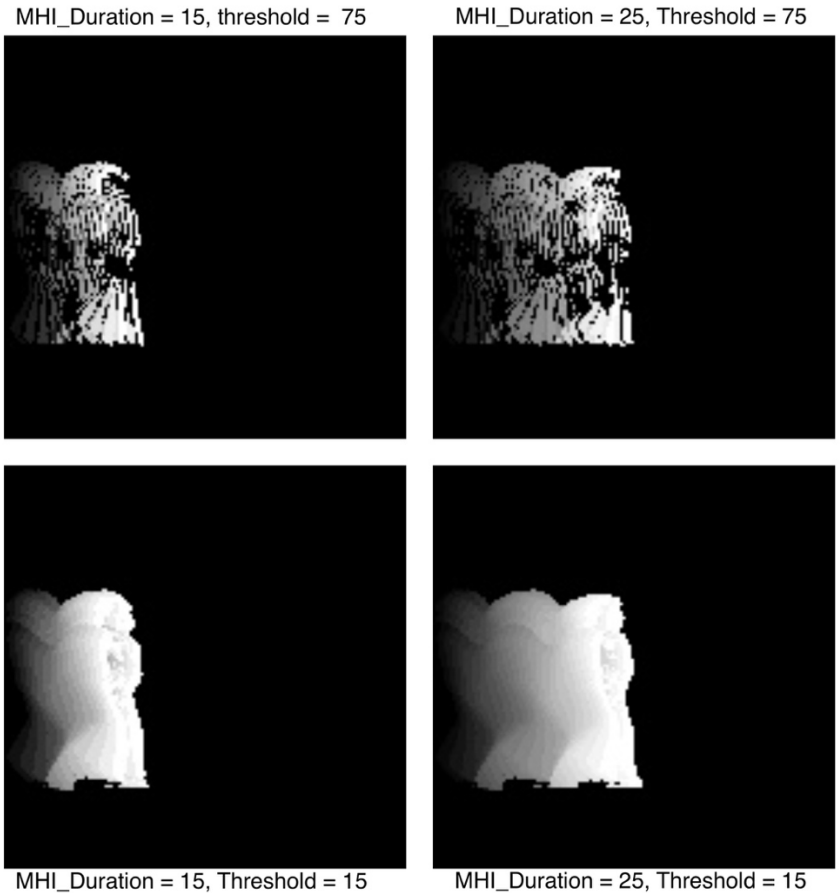
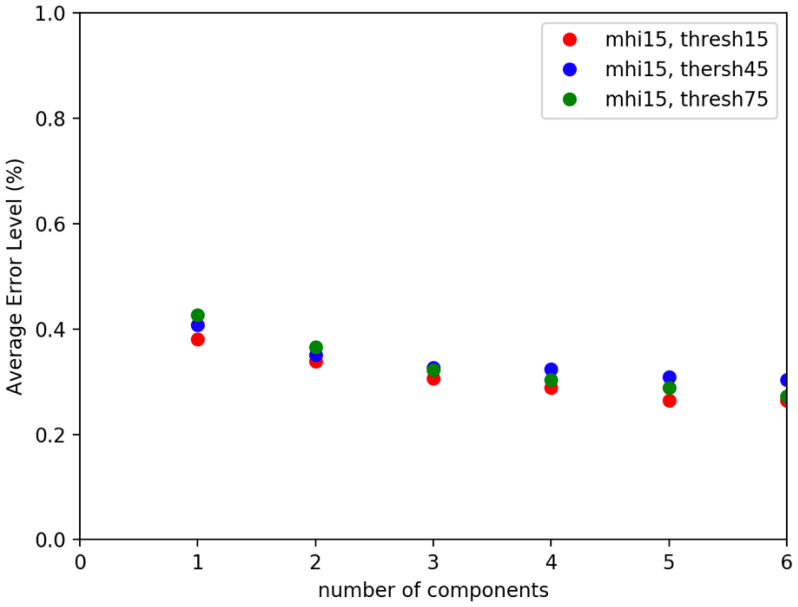


Figure 5: Various MHIs with different MHI_Duration and Threshold.
Taken from the same video from the Weizman dataset

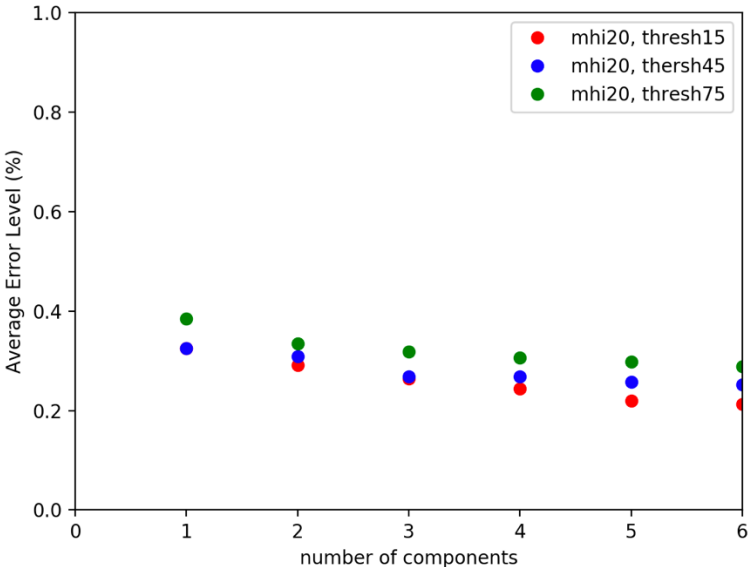


MHI_Duration = 15, threshold = 15	
Number of components k:	Accuracy
1	0.5916685
2	0.66131225
3	0.693455
4	0.7119065
5	0.7357145
6	0.735715

MHI_Duration = 15, threshold = 45	
Number of components k:	Accuracy
1	0.5916685
2	0.648812
3	0.673217
4	0.67678775
5	0.6916685
6	0.69702625

MHI_Duration = 15, threshold = 75	
Number of components k:	Accuracy
1	0.67500225
2	0.70893075
3	0.7351185
4	0.75654625
5	0.77975975
6	0.7869025

Figure 6: Testing results from the Weizman MHIs
with MHI_Duration =15 and threshold = 15, 45, 75

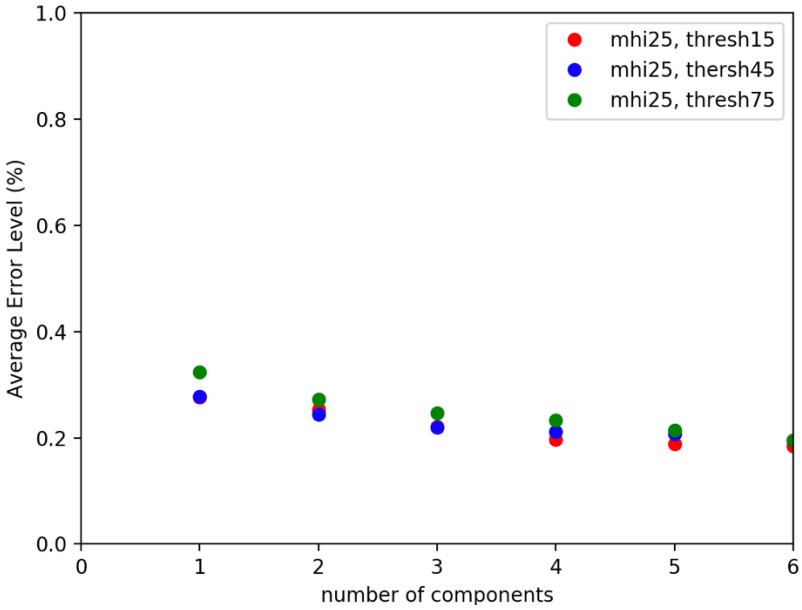


MHI_Duration = 20, threshold = 15	
Number of components k:	Accuracy
1	0.67500225
2	0.70893075
3	0.7351185
4	0.75654625
5	0.77975975
6	0.7869025

MHI_Duration = 20, threshold = 45	
Number of components k:	Accuracy
1	0.67440675
2	0.69047825
3	0.73095325
4	0.732144
5	0.74285825
6	0.7482145

MHI_Duration = 20, threshold = 75	
Number of components k:	Accuracy
1	0.614883
2	0.6654785
3	0.68095525
4	0.69404925
5	0.70178775
6	0.711311

Figure 7: Testing results from the Weizman MHIs
with MHI_Duration =20 and threshold = 15, 45, 75



MHI_Duration = 25, threshold = 15	
Number of components k:	Accuracy
1	0.7238095
2	0.74702375
3	0.77856975
4	0.8029745
5	0.81190275
6	0.816069

MHI_Duration = 25, threshold = 45	
Number of components k:	Accuracy
1	0.7220245
2	0.75595125
3	0.780355
4	0.78868875
5	0.79285425
6	0.804164

MHI_Duration = 25, threshold = 75	
Number of components k:	Accuracy
1	0.6761925
2	0.727977
3	0.75357125
4	0.76666625
5	0.7857125
6	0.804165

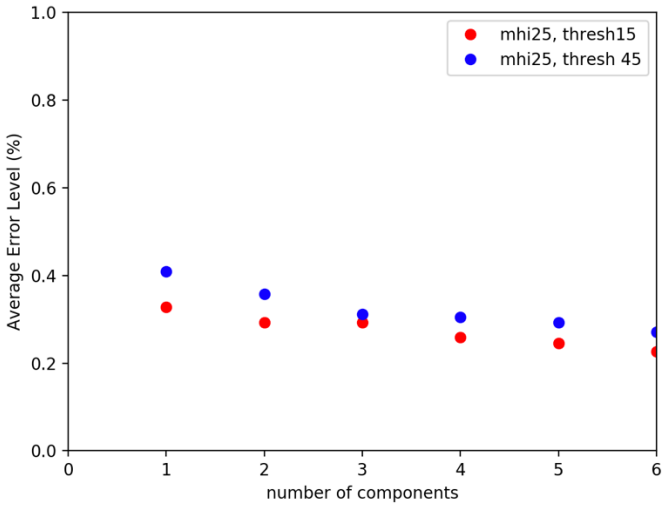
Figure 8: Testing results from the Weizman MHIs with MHI_Duration =25 and threshold = 15, 45, 75

From this testing, the best results came from testing k = 6 with a MHI_Duration of 25 and threshold of 15, which achieved an average accuracy of 81.6%. For every test, as the number of components increased, the accuracy increased, which is consistent with the testing with the faces datasets.

Various levels of MHI_Duration and threshold were tested before running 40 tests. This range seemed to yield the best results before testing, but testing further levels of MHI_Duration could improve accuracy further. Note: Too high of a MHI_Duration would not populate enough frames into the dataset to run the tests.

KHI Dataset:

Frames were grabbed from every video in the dataset until the video ended or 7 different MHIs were saved from a video. 10 experiments were run on this dataset with a MHI_Duration of 25, and threshold of 15 and 25. There were 5 classes tested and a total of 10 images per class run on each of these tests. Figure 9 shows the results of the experiment.



MHI_Duration = 25, threshold = 15	
Number of components k:	Accuracy
1	0.671431
2	0.707145
3	0.707145
4	0.740476
5	0.754761
6	0.773808

MHI_Duration = 25, threshold = 45	
Number of components k:	Accuracy
1	0.590478
2	0.642859
3	0.688098
4	0.69524
5	0.707145
6	0.728574

Figure 9: Testing results from the KTH MHIs
with MHI_Duration = 25 and threshold = 15, 45

Just like the Weizman dataset and the face recognition, the accuracy improved as the number of components improved. A lower threshold level also led to a higher accuracy, similar to the Weizman dataset. The best result of 77.3% was found with a threshold a MHI_Duration of 25 and threshold of 15.

5. Conclusions

L_1-PCA is a robust and effective method for facial recognition, and has shown decent accuracy with action recognition from MHIs. On clean datasets such as the ORL dataset, the classifier was extremely accurate with prediction, and even on a dataset with varying lighting and angles such as the CroppedYale dataset, it was still able to achieve high accuracy once more principal components were chosen.

The L_1-PCA was also able to perform well with the MHIs from the video datasets. Only a select number of MHI_Duration and threshold were selected in order to be able to run the test enough times (in this case, 40 times) to see accurate results. A further area of study would be to try even more levels of MHI_Duration and threshold. Tests could even be run with some classes in with certain values of MHI_Duration and threshold and other classes with different values for those same variables. An interesting concept if one has enough computing power and time would be to write a script to test certain MHI_Duration and threshold levels, find the best levels, and adaptively select and test new levels that achieve the highest accuracy

References:

[1] F. Maritato, Y. Liu, D. A. Pados, and S. Colonnese, "Face recognition with L_1-norm subspaces," in *Proc. SPIE Commercial + Scientific Sensing and Imaging*, Baltimore, MD, Apr. 2016.

[2] P. P. Markopoulos, G. Karystinos, and D. A. Pados, "Optimal algorithms for L-subspace signal processing," IEEE Trans. Signal Process., vol. 62, pp. 5046-5058, Oct. 2014.

[3] N. Kwak, "Principal component analysis based on L_1-norm maximization," IEEE Trans. Pattern Anal. Mach. Intell., vol. 30, pp. 1672-1680, Sep. 2008.

[4] S. Kundu, P. P. Markopoulos, and D. A. Pados, "Fast computation of the L_1-principal component of real- valued data," in Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process. (ICASSP), Florence, Italy, May 2014, pp. 8028-8032.

[5] Y. Kong and Y. Fu, "Human action recognition and prediction: a survey," *arXiv preprint arXiv:1806.11230* (2018).

[6] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," in *Proc. ICCV*, 2005. http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html - Classification Database

[7] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *IEEE ICPR*, 2004. http://www.nada.kth.se/cvap/actions/

[8] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Proc. ICCV VS-PETS*, 2005.

[9] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," IEEE Trans. Pattern Anal. Mach. Intell, vol. 23, pp. 643-660, June 2001.

[10] F. S. Samaria, A. C. Harter, "Parameterisation of a stochastic model for human face identification," in Proc. IEEE Workshop Applications of Computer Vision, Sarasota, FL, Dec. 1994, pp. 138-142.