Politecnico di Torino

JAFFE Dataset Facial and Emotional Recognition Using Principal Component Analysis

Mathematics in Machine Learning

Professor Francesco Vaccarino

Professor Mauro Gasparini

Farzad Imanpour – Hafez Ghaemi

A.Y. 2020/2021

Abstract

High-dimensional data are becoming increasingly common and are often difficult to interpret. There exist approaches for dimensionality reduction without losing significant information. In this paper, we use principal component analysis (PCA) as a dimensionality reduction technique before applying a kernel support vector machine (SVM) for classification. Also, classifiers with inherent dimensionality-reduction, such as linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA) have been applied to the original and reduced data (the dataset used in this work is the Japanese Female Facial Expression, JAFFE). The use of dimensionality reduction resulted in a performance improvement, especially on the emotion recognition task, compared to non-dimensionality reduction classification. Furthermore, we investigated the potential application of PCA as the simplest form of an autoencoder, and its ability was compared to a simple convolutional neural network autoencoder. Finally, we compared the results of emotional recognition with human-level performance using the available rating data.

Content

1 Theory

1.1 Principal component analysis

1.1.1 Introduction

Principal component analysis (PCA) [1], [2], [3] is one of the most popular multivariate statistical techniques. It has been widely used in the areas of pattern recognition and signal processing and is a statistical method under the broad title of factor analysis.
PCA forms the basis of multivariate data analysis based on projection methods. The most important use of PCA is to represent a multivariate data table as a smaller set of variables (summary indices) in order to observe trends, jumps, clusters and outliers. This overview may uncover the relationships between observations and variables, and among the variables. As the preprocessing step we performed dimensionality reduction in PCA which is an unsupervised learning algorithm, dimensionality reduction is a procedure that helps the as the same as name says to reduce the dimensionality of the dataset. This is related to the curse of dimensionality, that is a phenomenon that happens to high dimensional spaces and it is strictly correlated to the geometrical characteristics of the high dimensional spaces. So we can try to reduce in order to be able to perform well the algorithm. briefly, PCA can alleviate curse of dimentionality and improve classification or regression performance on tasks with small datsets but large feature spaces.PCA is a very flexible tool and allows the analysis of datasets that may contain, for example, multicollinearity, missing values, categorical data, and imprecise measurements. The goal is to extract the important information from the data and to express this information as a set of summary indices called principal components.
Statistically, PCA finds lines, planes, and hyper-planes in the K-dimensional space that approximate the data as well as in the least-squares sense. A line or plane that is the least-squares approximation of a set of data points makes the variance of the coordinates on the line or plane as large as possible (Figure 1).

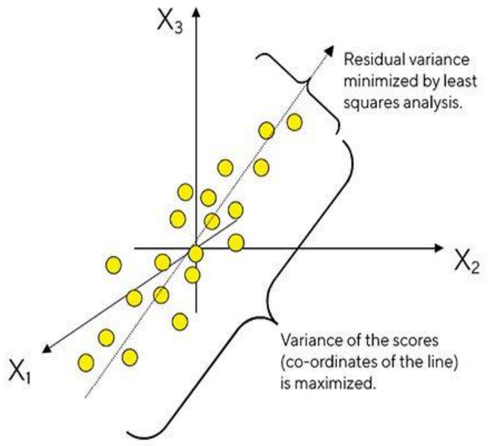

X_3
Residual variance minimized by least squares analysis.
X_2
X_1
Variance of the scores (co-ordinates of the line) is maximized.

Figure 1. PCA creates a visualization of data that minimizes residual variance in theleast-squares sense and maximizes the variance of the projection coordinates.

1.1.2 The PCA procedure

In the previous section, we explained why pre-processing the data for PCA is necessary. Now, let’s take a look at how PCA works, using a geometrical approach. our problem is to find a good number of principal components that allow us to reduce the dimensionality without eliminating too much information from the data and extract the important information from the data and to express this information as a set of summary indices called principal components.

Consider a matrix X with N rows (a.k.a. "observations") and K columns (a.k.a. "variables"). For this matrix, we construct a variable space with as many dimensions as there are variables. Each variable represents one coordinate axis. For each variable, the length has been standardized according to a scaling criterion, normally by scaling to unit variance. In the next step, each observation (row) of the X-matrix is placed in the K-dimensional variable space. Consequently, the rows in the data table form a swarm of points in this space (Figure 2).

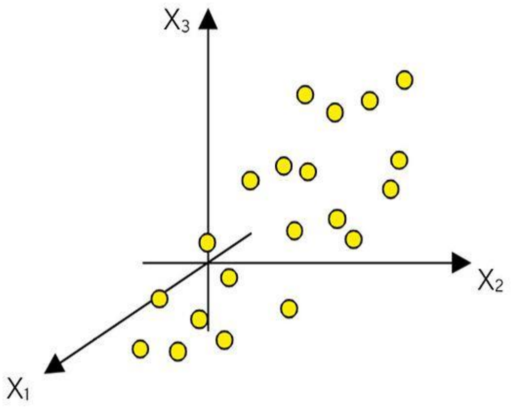


Figure 2. The observations (rows) in the data matrix X can be understood as a swarm of points in the variable space (K-space).

Next, mean-centering involves the subtraction of the variable averages from the data. The vector of averages corresponds to a point in the K-space (Figure 3)

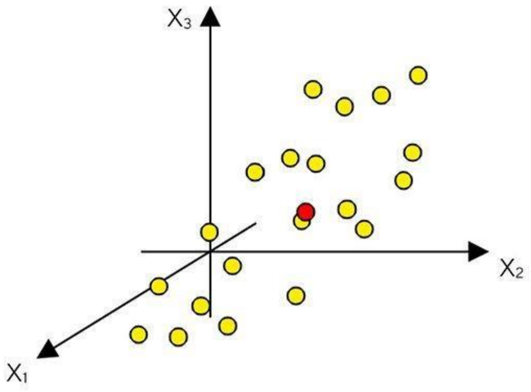


Figure 3. In the mean-centering procedure, you first compute the variable averages. This vector of averages is interpretable as a point (here in red) in space. The point is situated in the middle of the point swarm (at the center of gravity).

The subtraction of the averages from the data corresponds to a re-positioning of the coordinate system, such that the average point now is the origin (Figure 4)

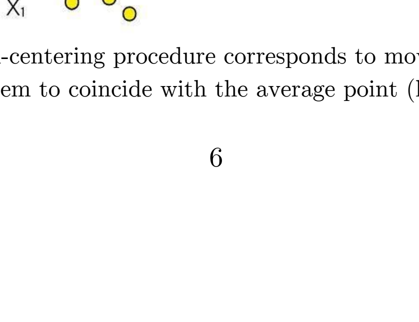


Figure 4. The mean-centering procedure corresponds to moving the origin of the coordinate system to coincide with the average point (here in red).

Given data on K variables:
PCs are denoted by $u_1, u_2, \dots, u_k$.
Since PCs are orthogonal linear transformations of the original vairables there is at most k PCs.
We can choose p based on what percentage of variance of the original data we would like to maintain.
The projection of n (observations), d-dimensional observations on the first PC u_1 is $u_1{}^T$X. We want to project on this first dimension to have maximum variance.

$$\text{var}(u_1{}^T \text{X}) = u_1{}^T \text{SX}$$

where S is the $k \times k$ sample covariance matrix of X.
This means that the variance stated above has no upper limit and so we can not find the maximum.
To solve this problem, we choose u_1 to maximize $u_1{}^T$SX while constraining u_1 to have unit length.
Therefore, we can rewrite the above optimization problem as:

$$\max u_1{}^T \text{SX}$$
$$\text{subject to } u_1{}^T u_1 = 1$$

To solve this optimization problem a Lagrange multiplier λ is introduced:

$$\text{L}(u_1, \lambda) = u_1{}^T \text{ SX} - \lambda(u_1{}^T u_1 - 1)$$

Lagrange multipliers are used to find the maximum or minimum of a function f (x, y) subject to constraint g(x, y) = c

We will have
Differentiating with respect to u_1 gives a set of d equations,

$$\text{S}u_1 = \lambda$$

Premultiplying both sides by $u_1{}^T$ we have:

$$u_1{}^T \text{S} u_1 = \lambda u_1{}^T u_1 = \lambda$$

$u_1{}^T \text{S} u_1$ is maximized if λ is the largest eigenvalue of S.
Clearly λ and u_1 are an eigenvalue and an eigenvector of S.
This shows that the first principal component is given by the eigenvector with the largest associated eigenvalue of the sample covariance matrix S. A similar argument can show that the p dominant eigenvectors of covariance matrix S determine the first p principal components.
Var($u_i{}^T u_i$) = $u_i{}^T S u_i$= λ_i where λ_i is an eigenvalue of the sample covariance matrix S and u_i is its corresponding eigenvector.
Var($u_1{}^T$X) is maximized if u_1 is the eigenvector of S with the corresponding maximum eigenvalue.

Each successive PC can be generated in the above manner by taking the eigenvectors of S that correspond to the eigenvalues:

$$\lambda_1 \geq \ldots \geq \lambda_k$$

such that

$$\mathrm{Var}({u_1}^T \ \mathrm{X}) \geq \ldots \geq \mathrm{Var}({u_k}^T \ \mathrm{X})$$

<u>THE PCA ALGORITHM:</u>

Recover basis (PCs): Calculate $XX^T = \sum_{i=1}^{n} x_i {x_i}^T$ and let U = eigenvectors of XX^T corresponding to the top p eigenvalues.

Encode training data: Y = U^T X where Y is a p × n matrix of encodings of the original data.

Reconstruct training data: $\hat{X}$ = UY = UU^T X.

Encode test example: y = U^T x where y is a p-dimensional encoding of x.

Reconstruct test example: $\hat{x}$= Uy = UU^T x.

the first principal component (u_1) is the line in the K-dimensional variable space that best approximates the data in the least-squares sense. This line goes through the average point. Each observation (yellow dot) may now be projected onto this line in order to get a coordinate value along the PC-line. This new coordinate value is also known as the score (Figure 5).

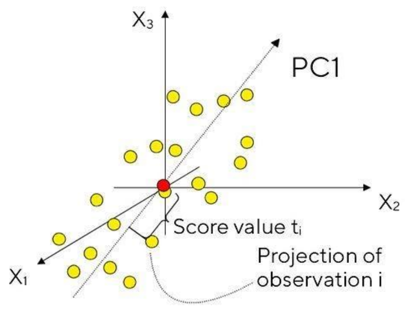


Figure 5. The first principal component (PC1) is the line that best accounts for the shape of the point swarm. It represents the maximum variance direction in the data. Each observation (yellow dot) may be projected onto this line in order to get a coordinate value along the PC-line. This value is known as a score.

Usually, one summary index or principal component is insufficient to model the systematic variation of a data set. Thus, a second summary index – a second principal component (u_2) – is calculated. The second PC is also represented by a line in the K-dimensional variable space, which is orthogonal to the first PC. This line also passes through the average point and improves the approximation of the X-data as much as possible (Figure 6).

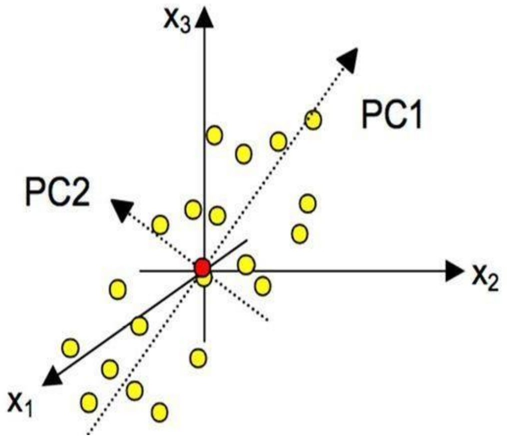


Figure 6. The second principal component (PC2) is oriented such that it reflects the second largest source of variation in the data while being orthogonal to the first PC. PC2 also passes through the average point.

Two principal components define a model plane. When two principal components have been derived, they together define a place, a window into the K-dimensional variable space. By projecting all the observations onto the low-dimensional subspace and plotting the results, it is possible to visualize the structure of the investigated data set. The coordinate values of the observations on this plane are called scores, and hence the plotting of such a projected configuration is known as a score plot (Figure 7).

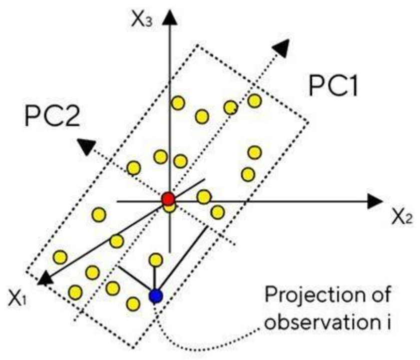


Figure 7. Two PCs form a plane. This plane is a window into the multidimensional space, which can be visualized graphically. Each observation may be projected onto this plane, giving a score for each.

1.1.3 SVD in PCA

A unique solution for matrix U in the above PCA algorithm can be obtained by finding the singular value decomposition of X (We know that for XX^T, SVD is the same as orthogonal and eigenvalue decompositions). For each rank p, we have

$$U_p = eigs(XX^T)$$

which means that the first p columns of U are the eignenvalues of XX^T. Therefor, finding the SVD corresponds to finding the principal components of X.

1.1.4 Kernel PCA

In the field of multivariate statistics, kernel principal component analysis (kernel PCA) is an extension of principal component analysis (PCA) using techniques of kernel methods. Using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space.

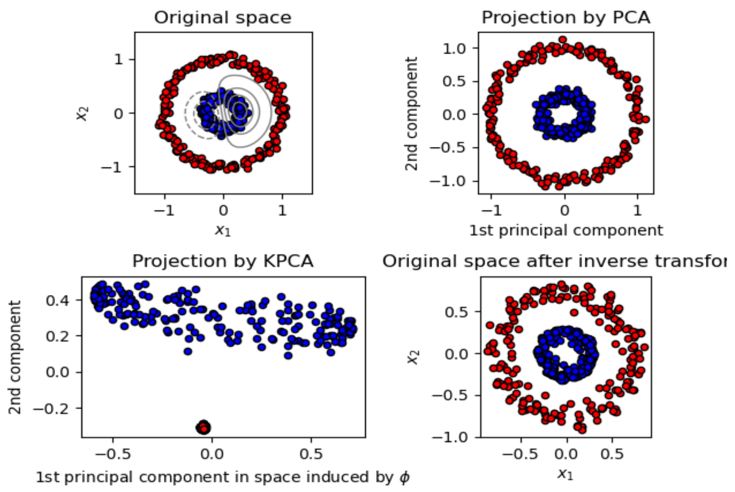


Figure 8. An example of kernel PCA

The example in Figure 8 shows that Kernel PCA is able to find a projection of the data that makes data linearly separable.

To understand the utility of kernel PCA, particularly for clustering, observe that, while N points cannot, in general, be linearly separated in d>N dimensions, they can almost always be linearly separated in N dimensions. That is, given N points X_i , if we map them to an N-dimensional space with

$\Phi(x_i)$ Where $\Phi : R^d \rightarrow R^N$

Moreover, there is the Kernel trick where it came to help when the data is not separable. So we need a non linear map. That goes to N, where N is required to be subset of Hilbert space. This is important because it can help us to calculate this kernel that is defined as the inner product of the features.

This is important because thanks to the Representer theorem is tell us if a function is in a Hilbert space, we can find a solution as a linear combination of the images, and also thanks to the Mercer's condition where it tells us that we can use the Gram matrix so that it is a positive semi-definite matrix to compute these inner product. In this way, we do not actually care about high dimensional space where we are mapping our data, but we just need to compute these inner products, so this is a good thing for our algorithm, but it can also be a disadvantage because in this way it depends on the number of points, so if the number of points is large, the computation of this gram matrix can become difficult.

It is easy to construct a hyperplane that divides the points into arbitrary clusters. Of course, this Φ creates linearly independent vectors, so there is no covariance on which to perform eigendecomposition explicitly as we would in linear PCA.

Instead, in kernel PCA, a non-trivial, arbitrary Φ function is 'chosen' that is never calculated explicitly, allowing the possibility to use very-high-dimensional Φ 's where we never have to actually evaluate the data in that space. Since we generally try to avoid working in the Φ-space, which we will call the 'feature space', we can create the N-by-N kernel

K = K(x,y) = $(\Phi(x)^T\Phi(y))$

which represents the inner product space of the otherwise intractable feature space. Some well-known kernels are given below:

linear: $\langle x, x' \rangle$

polynomial: $(\gamma\langle x, x' \rangle + r)^d$ where d is specified by parameter r, by coef0.

rbf: $\exp(-\gamma\|x - x'\|^2)$ where γ is specified by parameter gamma, must be greater than 0.

sigmoid: $\tanh(\gamma\langle x, x' \rangle + r)$ where r is specified by coef0.

1.2 Support vector machine

1.2.1 Introduction

Support vector machine (SVM) [4], [6] is a machine learning (ML) algorithm as it produces significant accuracy with less computational power than most of other ML methods. the idea behind a support vector machines is that we need to find a hyperplane that separate as well as well as possible, the two classes.

1.2.2 The SVM classification prodedure

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N is the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence (Figure 9).

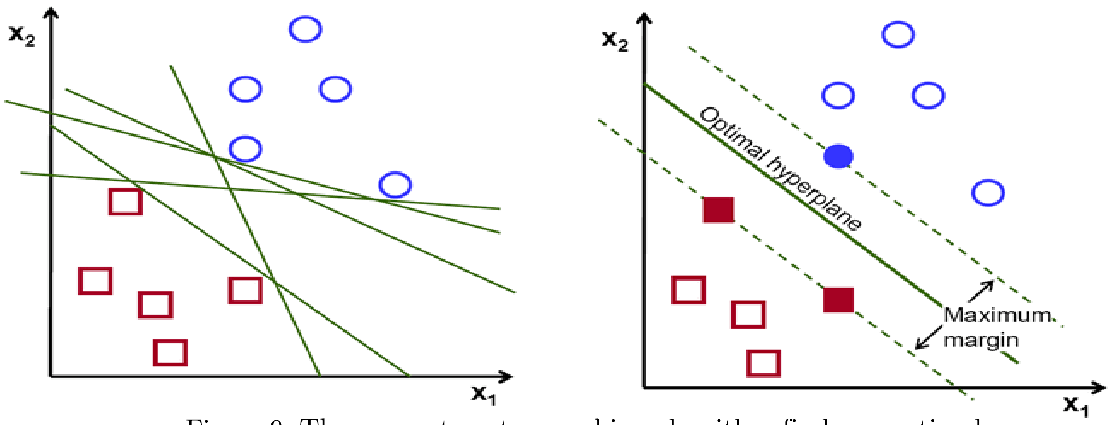


Figure 9. The support vector machine algorithm finds an optimal hyperplane to separate the two classes

1.2.3 The SVM hyperplanes

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane (Figure 10). It becomes difficult to imagine when the number of features exceeds 3.

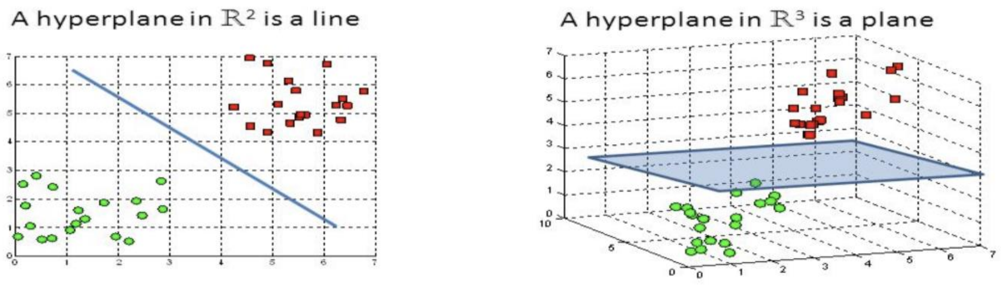


Figure 10. separating hyperplanes in SVM

1.2.4 The support vectors and the margin

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane because these are the points that help us build our SVM.

Hard margin: If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two strict (hard) hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. With a normalized or standardized dataset, these hyperplanes can be described by the equations below:

$$w^T x - b = 1$$

$$(anything\ on\ or\ above\ this\ boundary\ is\ of\ one\ class, with\ label\ 1)$$

and

$$w^T x - b = -1$$

$$(anything\ on\ or\ below\ this\ boundary\ is\ of\ the\ other\ class, with\ label\ -1)$$

where w is the normal vector to the hyperplane.

Geometrically, the distance between these two hyperplanes is $\frac{2}{||W||}$. So to maximize the distance between the planes we want to minimize ||w||. The distance is computed using the distance from a point to a plane equation. We also have to prevent data points from falling into the margin, we add the following constraint: for each i either,

$$(w^T x_i - b) \geq 1\ , if\ y_i = 1,\ (w^T x_i - b) \leq -1\ \ , if\ y_i = -1$$

These constraints state that each data point must lie on the correct side of the margin. This can be rewritten as:

$$y_i(w^T x_i - b) \geq 1, for\ all\ 1\ \leq i\ \leq n$$

We can put this together to get the optimization problem:

Minimize ||w|| subject to $y_i(w^T x_i - b) \geq 1 for\ i = 1, \dots, n.$

The w and b that solve this problem determine our classifier, x→ sgn(x^T x-b) where sgn(.) is the sign function.

An important consequence of this geometric description is that the max-margin hyperplane is completely determined by those x_i that lie nearest to it. These are called support vectors.

The hyperplane is then found by solving this optimization problem.

Soft margin: This idea is based on a simple premise: allow SVM to make a certain number of mistakes and keep margin as wide as possible so that other points can still be classified correctly. This can be done simply by modifying the objective of SVM.
In this new setting, we would aim to minimize the following objective:

$$L = \frac{1}{2}||w||^2 + C.(\#number\ of\ mistakes)$$

This differs from the original objective in the second term. Here, C is a hyperparameter that decides the trade-off between maximizing the margin and minimizing the mistakes. When C is small, classification mistakes are given less importance and focus is more on maximizing the margin, whereas when C is large, the focus is more on avoiding misclassification at the expense of keeping the margin small. At this point, we should note, however, that not all mistakes are equal. Data points that are far away on the wrong side of the decision boundary should incur more penalty as compared to the ones that are closer.

The idea is: for every data point x_i, we introduce a slack variable ξ_i. The value of ξ_i is the distance of x_i from the corresponding class's margin if x_i is on the wrong side of the margin, otherwise zero. Thus the points that are far away from the margin on the wrong side would get more penalty.

With this idea, each data point x_i needs to satisfy the following constraint:

$$y_i(w^T x_i - b) \geq 1 - \xi_i$$

Here, the left-hand side of the inequality could be thought of like the confidence of classification. Confidence score ≥ 1 suggests that classifier has classified the point correctly. However, if confidence score ≤ 1, it means that classifier did not classify the point correctly and incurring a linear penalty of ξ_i.

Given these constraints, our objective is to minimize the following function:

$$L = \frac{1}{2}||w||^2 + C\textstyle\sum_i \xi_i + \displaystyle\sum i\, \lambda_i(y_i(w.x_i + b) - 1 + \xi_i)$$

where the parameter λ determines the trade-off between increasing the margin size and ensuring that the x_i lie on the correct side of the margin. Thus, for sufficiently small values of λ, the second term in the loss function will become negligible, hence, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

Figure 11 is a graphical comparison between hard and soft margins.

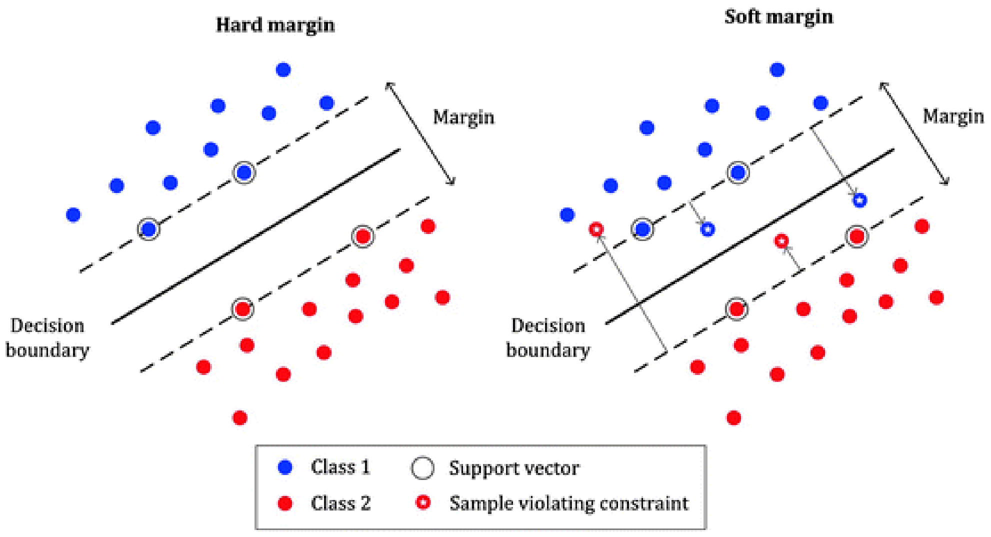


Figure 11. SVM hard and soft margins

1.3 Kernel SVM

The original maximum-margin hyperplane algorithm constructed a linear classifier. But to create nonlinear classifiers we can apply the kernel trick to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high-dimensional; although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.

SVM algorithms can use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions, for example, linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. Introduce Kernel functions for sequence data, graphs, text, images, as well as vectors. The most used type of kernel function is RBF because localized and finite response along the entire x-axis. The kernel functions return the inner product between two points in a suitable feature space by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

The kernel function can be any of the following:

linear: $\langle x, x' \rangle$

polynomial: $(\gamma\langle x, x' \rangle + r)^d$ where d is specified by parameter r, by coef0.

rbf: $\exp(-\gamma\|x - x'\|^2)$ where γ is specified by parameter gamma, must be greater than 0.

sigmoid: $\tanh(\gamma\langle x, x' \rangle + r)$ where r is specified by coef0.

1.4 Linear and quadratic discriminant analysis

Linear Discriminant Analysis (LDA) [2], [5] and Quadratic Discriminant Analysis (QDA) [5] are two classic classifiers, with, as their names suggest, a linear and a quadratic decision surface, respectively).

These classifiers are attractive because they have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice, and have no hyperparameters to tune.

The plot in Figure 12 shows decision boundaries for Linear Discriminant Analysis and Quadratic Discriminant Analysis. The bottom row demonstrates that Linear Discriminant Analysis can only learn linear boundaries, while Quadratic Discriminant Analysis can learn quadratic boundaries and is therefore more flexible.

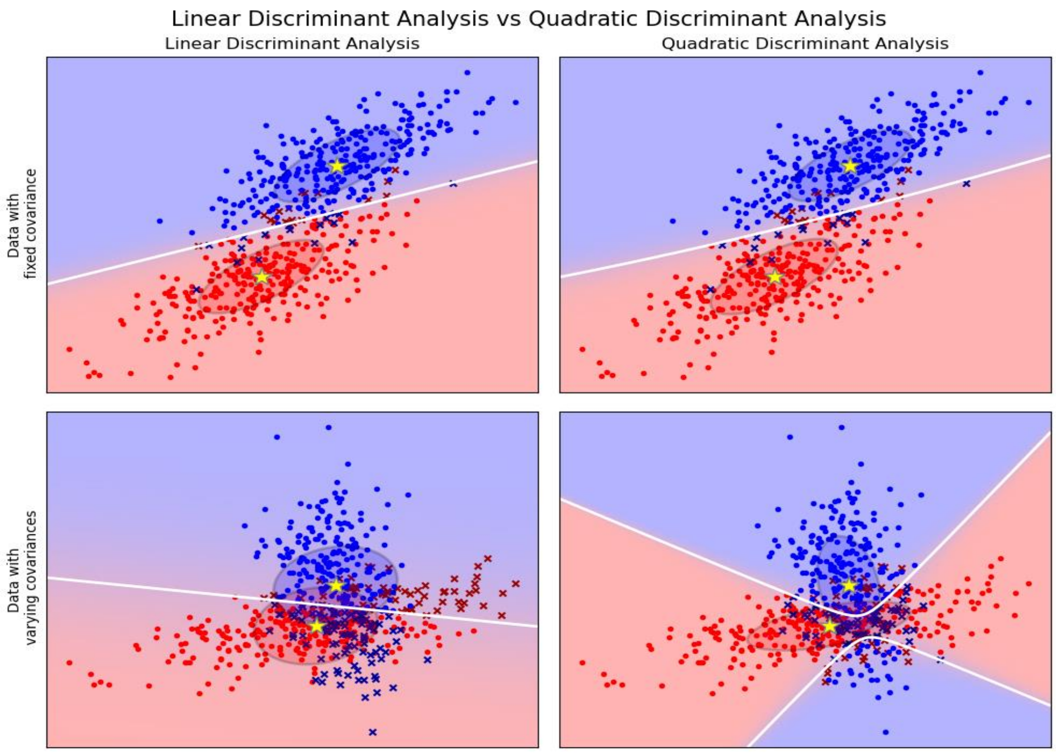


Figure 12. LDA and QDA decision boundaries

1.4.1 Dimensionality reduction using LDA and QDA

Linear Discriminant Analysis can be used to perform supervised dimensionality reduction, by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes (in a precise sense discussed in the mathematical section below). The dimension of the output is necessarily less than the number of classes, so this is in general a rather strong dimensionality reduction, and only makes sense in a multi-class setting.

Both LDA and QDA can be derived from simple probabilistic models which model the class conditional distribution of the data P(X|y = k) for each class k. Predictions can then be obtained by using Bayes' rule, for each training sample $x \in R^d$:

$$P(y=k|x) = \frac{P(x|y=k)P(y=k)}{P(x)} = \frac{P(x|y=k)P(y=k)}{\sum_l P(x|y=l) \cdot P(y=l)}$$

and we select the class k which maximizes this posterior probability. More specifically, for linear and quadratic discriminant analysis, P(x|y) is modeled as a multivariate Gaussian distribution with density:

$$P(x|y=k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^t \Sigma_k^{-1}(x-\mu_k)\right)$$

where d is the number of features.

1.4.2 Mathematical formulation of the QDA

According to the model above, the log of the posterior is:

$$\log P(y=k|x) \quad = \log P(x|y=k) + \log P(y=k) + Cst$$

$$= -\frac{1}{2}\log |\Sigma_k| - \frac{1}{2}(x-\mu_k)^t \Sigma_k^{-1}(x-\mu_k) + \log P(y=k) + Cst$$

where the constant term Cst corresponds to the denominator $P(x)$, in addition to other constant terms from the Gaussian. The predicted class is the one that maximises this log-posterior.

1.4.3 Mathematical formulation of the LDA

LDA is a special case of QDA, where the Gaussians for each class are assumed to share the same covariance matrix$\Sigma_k = \Sigma$ for all . This reduces the log posterior to:

$$\log P(y=k|x) = -\frac{1}{2}(x-\mu_k)^t \Sigma^{-1}(x-\mu_k) + \log P(y=k) + Cst.$$

The term $(x-\mu_k)^t \Sigma^{-1}(x-\mu_k)$ corresponds to the Mahalanobis Distance between the sample x and the mean μ_k . The Mahalanobis distance tells how close x is from μ_k, while also accounting for the variance of each feature. We can thus interpret LDA as assigning to the class whose mean is the closest in terms of Mahalanobis distance, while also accounting for the class prior probabilities.

The log-posterior of LDA can also be written as:

$$\log P(y=k|x) = \omega_k^t x + \omega_{k0} + Cst.$$

where $\omega_k = \Sigma^{-1}\mu_k$ and $\omega_{k0} = -\frac{1}{2}\mu_k^t \Sigma^{-1}\mu_k + \log P(y=k)$. From the above formula, it is clear that LDA has a linear decision surface. In the case of QDA, there are no assumptions on the covariance matrices of the Gaussians, leading to quadratic decision surfaces.

As mentioned above, we can interpret LDA as assigning x to the class whose mean μ_k is the closest in terms of Mahalanobis distance, while also accounting for the class prior probabilities. Alternatively, LDA is equivalent to first sphering the data so that the covariance matrix is the identity, and then assigning to the closest mean in terms of Euclidean distance (still accounting for the class priors).

Note that the K means of μ_k are vectors in $\mathcal{R}^d$, and they lie in an affine subspace H of dimension at most $K-1$ (2 points lie on a line, 3 points lie on a plane, etc.).

Computing Euclidean distances in this d-dimensional space is equivalent to first projecting the data points into H, and computing the distances there (since the other dimensions will contribute equally to each class in terms of distance). In other words, if x is closest to μ_k in the original space, it will also be the case in H. This shows that, implicit in the LDA classifier, there is a dimensionality reduction by linear projection onto a $K-1$ dimensional space.

We can reduce the dimension even more, to a chosen L, by projecting onto the linear subspace H_L which maximizes the variance of μ_k^* the after projection (in effect, we are doing a form of PCA for the transformed class means μ_k^*).

2 Image Reconstruction, and Facial and Emotion Recognition

2.1 Dataset

The JAFFE dataset [8] contains 213 images of seven facial expressions (six basic facial expressions including happiness, fear, anger, disgust, sadness, and surprise, plus one neutral expression) posed by 10 Japanese female models (The number of classes in the facial recognition task). The dataset is almost balanced, both in terms of face and emotion labels. So, thres is no need for data augmentation and balancing the dataset. Alongside the image data, we have ratings where each image has been rated on the six basic emotion on a scale of 1 to 5 by 60 Japanese students (these rating data will be later used to perform a statistical analysis and comparison between human and machine emotion recognition performance). The database was planned and assembled by Michael Lyons, Miyuki Kamachi, and Jiro Gyoba. Each image has a corresponding label related to the person represented in it and the posed emotion. We can use this dataset for image

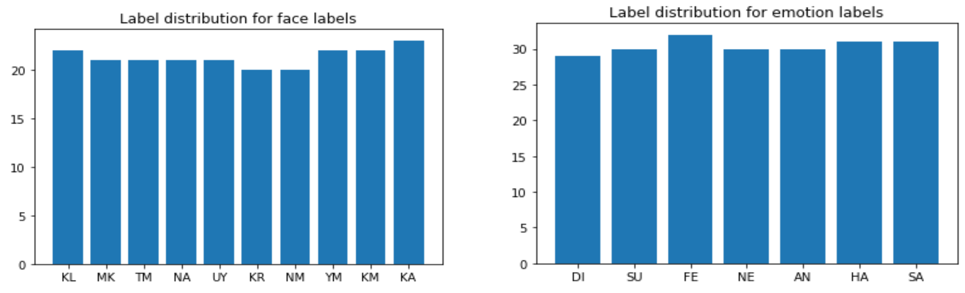


Figure 13. Label distributions for face and emotion labels

reconstruction, and facial and emotional classification. Three sample images from the dataset are given in Figure 14.

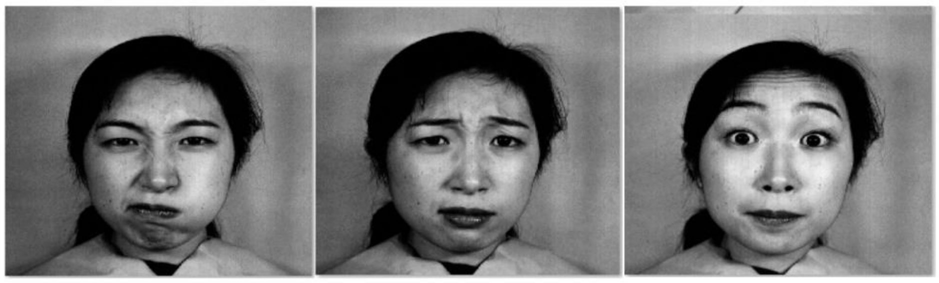


Figure 14. Three sample images from the JAFFE dataset

2.2 Applying PCA to the JAFFE

In this section, we transform our dataset in a lower-dimensional dataset using the transform method of the PCA() module in Scikit-Learn [9].

We performed PCA with and without z-normalization of the dataset. However, the image reconstruction, and face and emotion recognition performance of the dataset did not differ. Also, we performed kernel PCA with different kernels. However, kernel PCA did not improve the performance on image reconstruction, and facial and emotional recognition. This may be attributed to the curse of dimentionality. The feature vector for images is large enough, and applying kernel PCA increases the number of features even more, which may in turn exacerbate the curse of dimentionality. Therefore, from now on, only the non-normalized PCA results will be presented. Applying the PCA without specifying the number of components or the explained variance ratio will yield all of the principal components (in our case the minimum of the number of features (213) and the number of samples (65536)). The plot in the Figure 15 shows the explained variance ratio of the dataset covered by each principal component (PC). We can see the importance of the first PCs as the explained variance quickly drops after the first principal components. Due to this analysis we can comprehend that all PCs are not necessary to represent the dataset.

We can select 90% of the total variance, corresponding to 44 PCs. Therefore, doing so, we can go from 213 PCs to 44 PCs, showing a huge improvement in terms of computational cost. Figure 16 represents the plot of the selected PCA with 44 principal components.

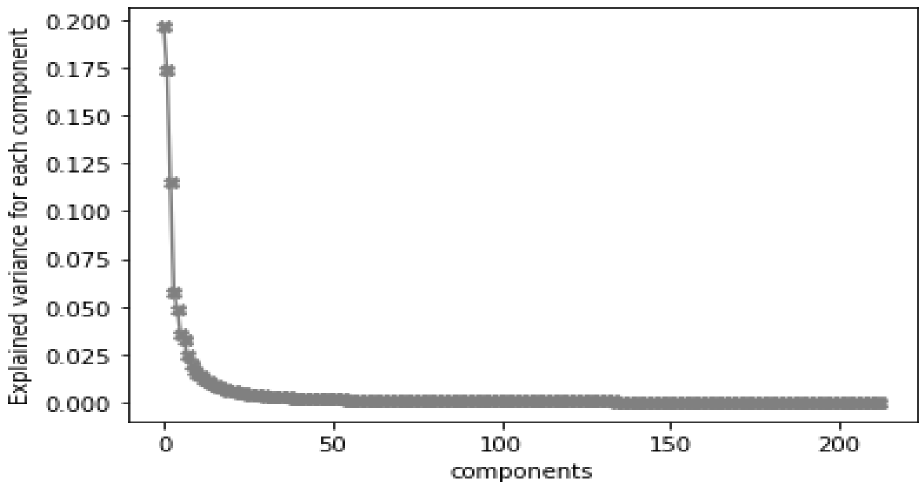


Figure 15. Explained variance ration covered by each PC

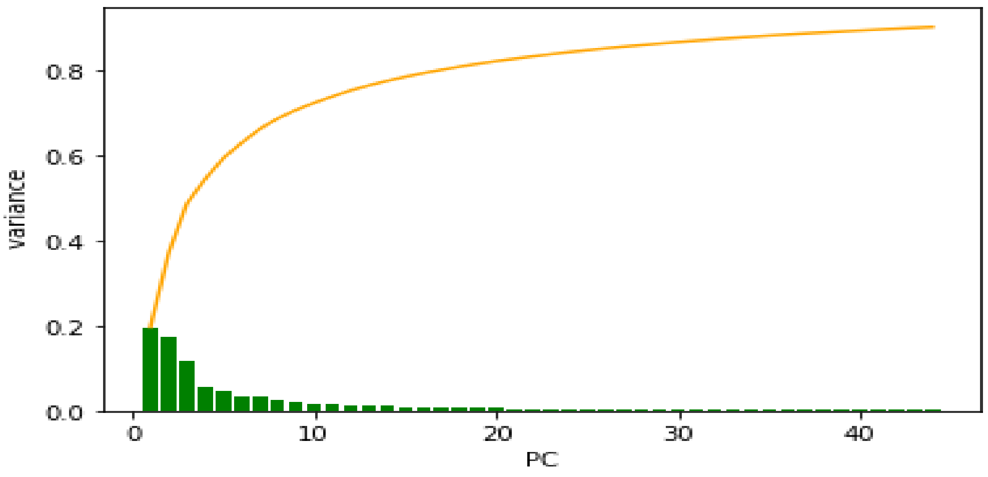


Figure 16. Pareto chart for explained variance ratio up to 44 PCs

It is observable that we reach the 90% of cumulative explained variance with only 44 PCs. If we want to clarify about the role of each PC, we can say that since our case is just images and pixels, we cannot examine the percentage of features represented by each PC, as done in tabular data reduction. Instead, here each PC can be interpreted as indicating different parts and textures present in the face. The PCs resulted from image data are sometimes called eigenfaces. Some of these eigenfaces can be observed in Figure 17. Naturally, the first PCs represent richer representations and more information related to faces. The last principal component almost looks like a noisy image. Also, in Figure 18, we can observe the difference of the first PC with the original image.

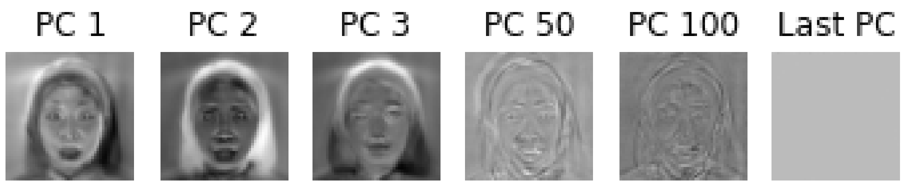


Figure 17. The eigenfaces

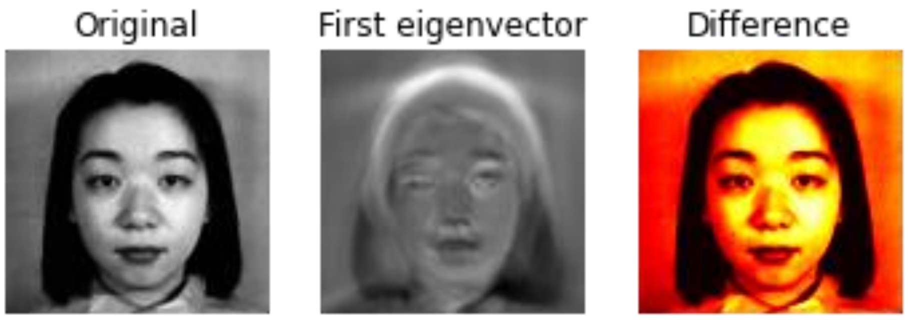


Figure 18. The difference between the original image and the first eigenface

2.3 Face Reconstruction

Mathematically, PCA reduces the initial size of the flattended image feature vector from 1 × n to 1×p. We can then multiply the resulting reduced feature vector by the p×n matrix containing the PCs of the PCA, and the result will be a 1 × n feature vector, which can be reshaped to reconstruct the original image.

And as a rule of thumb, we know that we must keep between to 70 percent and 90 percent of the variance explained and we kept 90%.

Here, we will represent effect of using different number of PCs in image reconstruction. Figure 19 shows the reconstructed images using different number of PCs. We can see that as we increase the number of PCs, we will have a higher-quality reconstruction, and using all principal components results in the original image. However, the goal of image reconstruction is also image compression. Therefore, it is not reasonable to use all PCs for image reconstruction, as it produces the same iamge. Within the 44th PC (90 percent cumulative explained variance ratio), we can see almost accurate faces, and after that there is not much difference with the original image (see Figure 18). From now on, we will use the 90-percent cumulative explained variance as the default mode for image reconstruction.

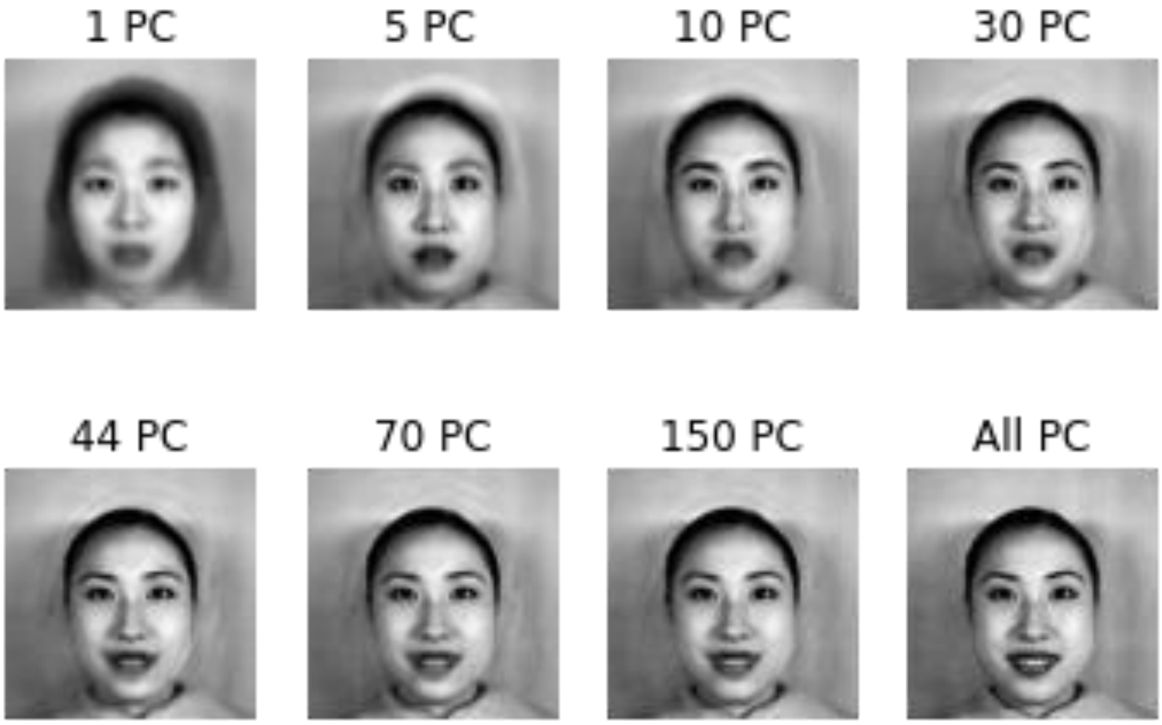


Figure 19. Reconstructed images using different number of principal components

Finally, in Figure 20, we can see the difference between reconstructed images using PCs up to 90 percent explained variance with original the ones and their difference. The difference is almost negligible.

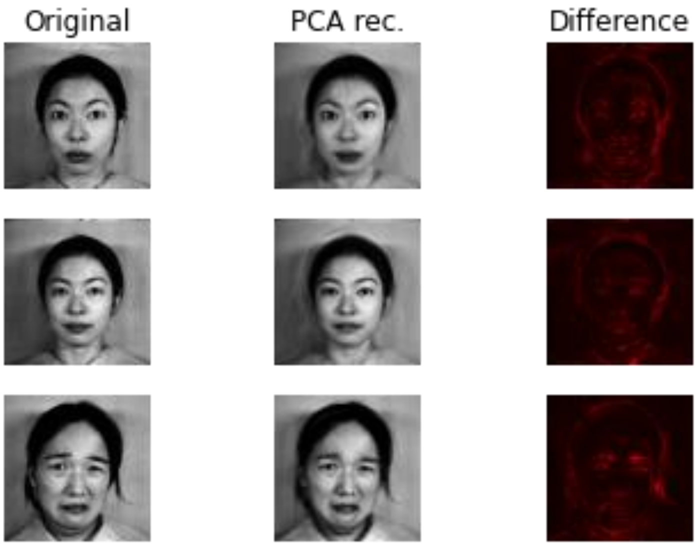


Figure 20. Comparting reconstruction of images using PCs with the original images and their differnce

2.4 Visualizing the first principal components with labels

If we want to predict how well a classifier may perform on the data reduced by PCA, for both emotion and facial recognition, we can see how well-seperated the first principal components are with respect to the class labels. In other words, we want to see how the two principal components are able to distinguish among different classes on the two tasks of facial and emotion recognition. Therefore, in Figures 21 and 22, on the x-axis we set the 1st PC, while on the y-axis we set the 2nd PC. The points with different colours show different classes. Figure 21 is the representation of the two PCs based on the facial recognition task, and Figure 22 is the representation based on the emotion detection task.

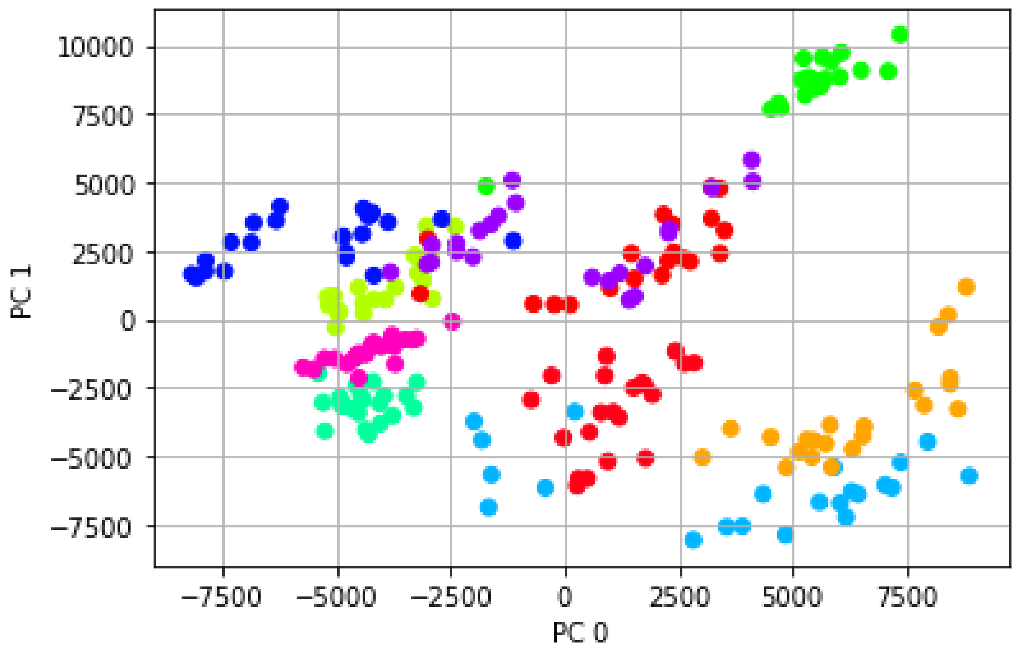


Figure 21. The first principal components colored based on facial recognition labels for each sample

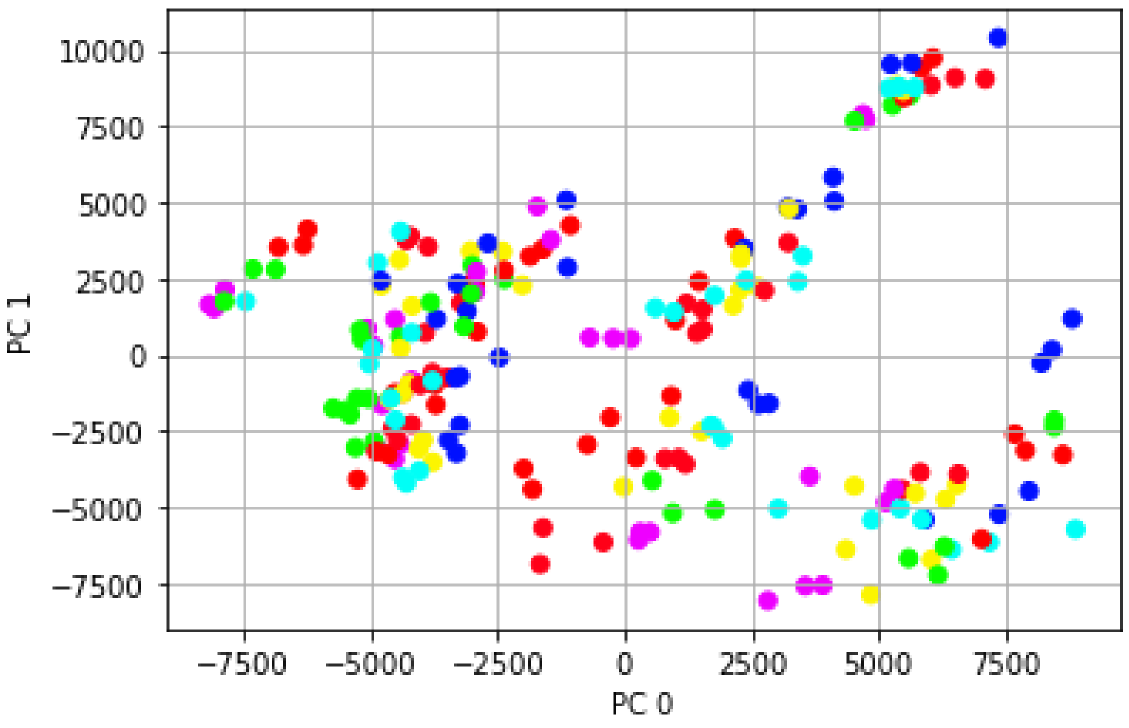


Figure 22. The first principal components colored based on emotion detection labels for each sample

These figures show that we should expect a much harder task when classifying emotions on this dataset, as the features related to this task are hard to distinguish.

2.5 Facial and emotion recognition using SVM

The JFFAE dataset has both facial and emotion labels of the individuals represented in the images. Therefore, we can perform classification on both of these labeled sets and evaluate the averge accuracy of different predictors.

Here, we perform classification using SVM with different setups and hyperparameters for both facial and emotion recognition with and without applying the PCA. Since the dataset is small, for each classifier, we perform a K-fold cross validation with K = 20. For the PCA-based classifiers, this cross validation was performed for each number of PCs used from 1 up to 213 to find the best number of PCs to be used for the final classifier. Afterwards, a hold-out train/test split was performed to evaluate the performance of this classifier, and more importantly, per-class test accuracies. Since the dataset is balanced, only the accuracy is sufficient for evaluating classifiers. The classification was performed using hard-margin and soft-margin (kernel) SVM with and without PCA applied to the data. The kernel used in the kernel version of SVM is rbf.

Table 1 represents the SVM algorithm classification results without applying PCA. It is clear that the accuracy for facial recognition for Hard SVM is almost 100 percent whether we use a kernel or not. But for emotion recognition there is a huge differnce between hard-margin and soft-margin SVM results. For the soft-margin SVM, the penalty term was chosen as $C = 1000$ after perfoming a grid search.

Table 1. The SVM classification results without applying PCA

SVM margin	Kernel	Recognition task	Accuracy
Hard	✗	Facial	99%
Hard	✗	Emotion	33%
Soft	✗	Facial	99%
Soft	✗	Emotion	92%
Soft	✓	Facial	99%
Soft	✓	Emotion	92%

As described above, we also used SVM on the dataset after dimentionality reduction with PCA. The accuracy per number of PCs used is given in Figures 23 to 25. The confusion matrix for SVM emotion and facial recognition with PCA using the best number of PCs chosen after cross-validation (Figures 23 to 25) is plotted in Figures 26 to 28. Furthermore, Figure 29 compares the training time for soft and hard-margin SVM as we increase the number of PCs used for classification. As expected the training time increases with the number of PCs. Also, the soft-margin SVM requires more time for training, which is also expected because the optimization problem needed to be solved with soft-margin is harder.

We can discuss the results obtained using the different SVM classifiers with and without PCA. As discussed in the previous section, the facial recognition task is easier than emotion detection, and therefore almost all of the classifeirs achieve perfect performance. However, this is not the case for the emotion recognition task. The best performance is obtained using the soft SVM and kernel soft SVM. Generally, soft SVM performs better than hard SVM because the penalty term allows non-linear separation. This is especially effective in our case where we have a huge feature space and an image dataset. Also, SVM is known to alleviate curse of dimentionality [2], and using the soft version of SVM can improve this issue even more by allowing more explanation in the data space. On the other hand, kernel SVM does not show any improvement with respect to soft SVM because the use of a high penalty term (C=1000) adds enough non-linear separation, and a kernel is not effective anymore.

Figures 30, and 31 show the true and predicted labels of the best-performing data models (soft-margin SVM after PCA with the optimum number of PCs) on a portion of the test set.

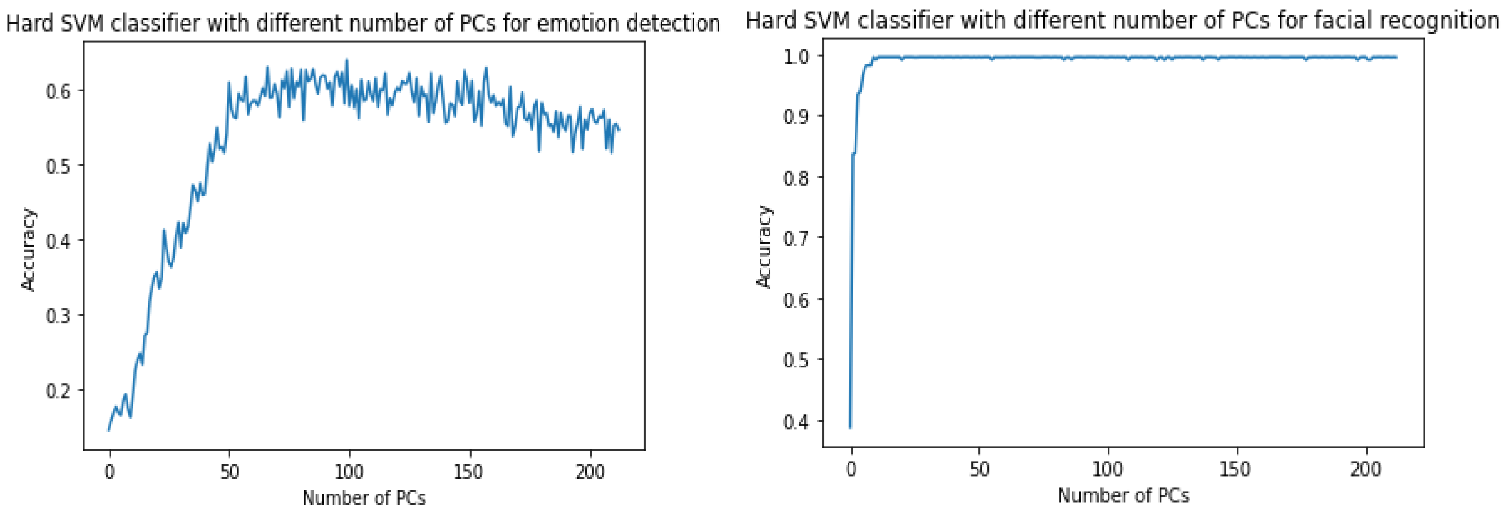


Figure 23. Classification performance of hard SVM with PCA for facial and emotion recognition as the number of principal components increases

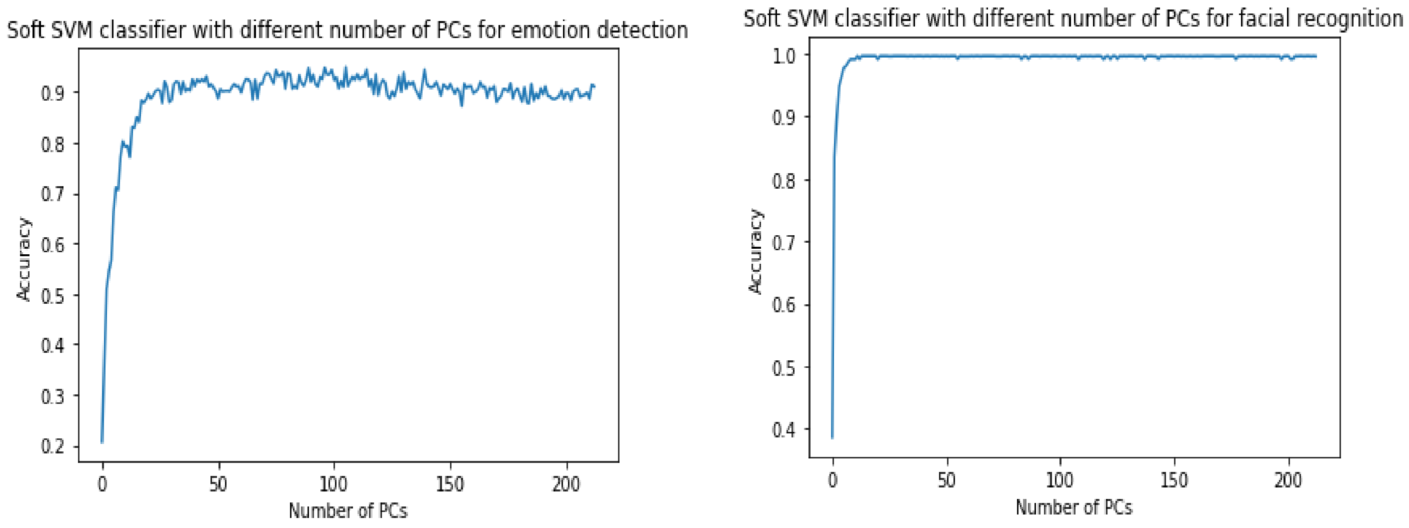


Figure 24. Classification performance of soft SVM with PCA for facial and emotion recognition as the number of principal components increases

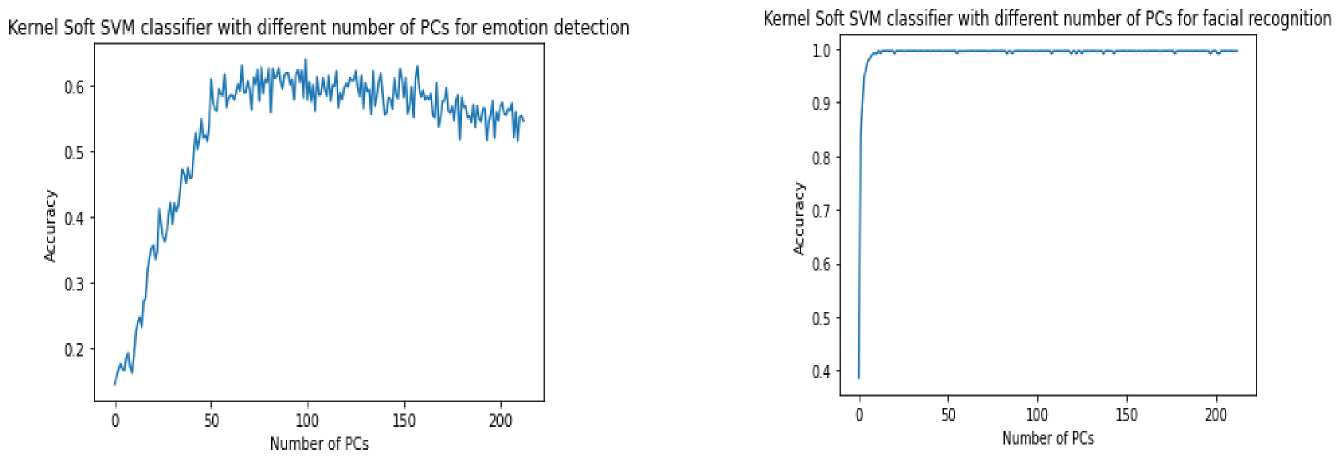


Figure 25. Classification performance of kernel SVM with PCA for facial and emotion recognition as the number of principal components increases

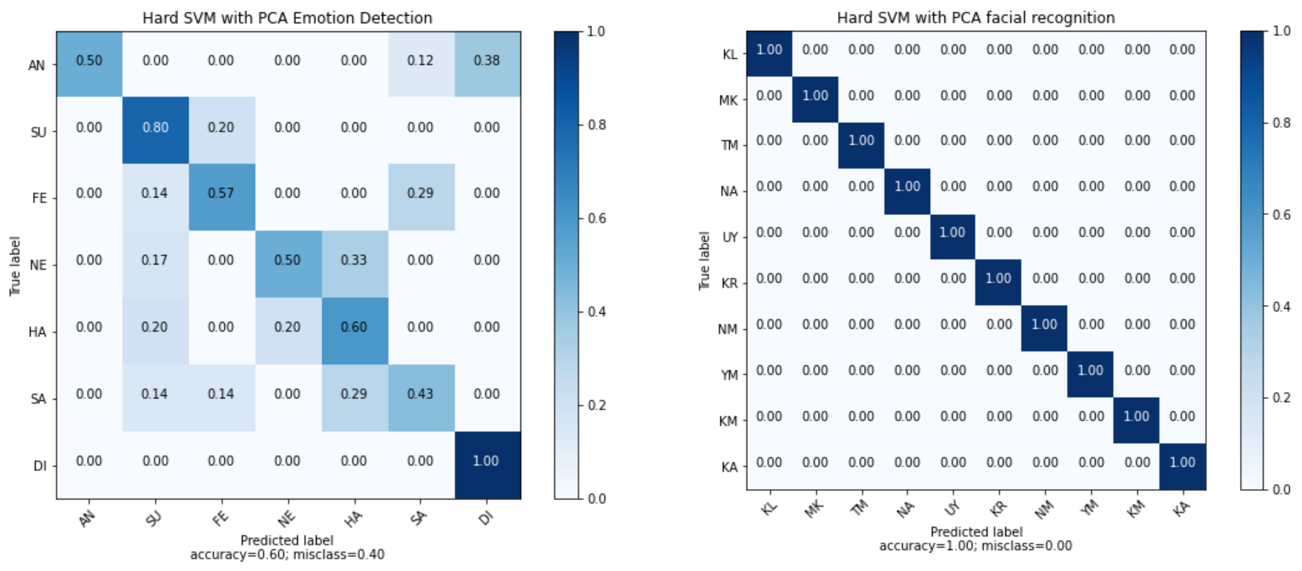


Figure 26. Confusion matrix of hard svm with PCA for emotion and facial recognition

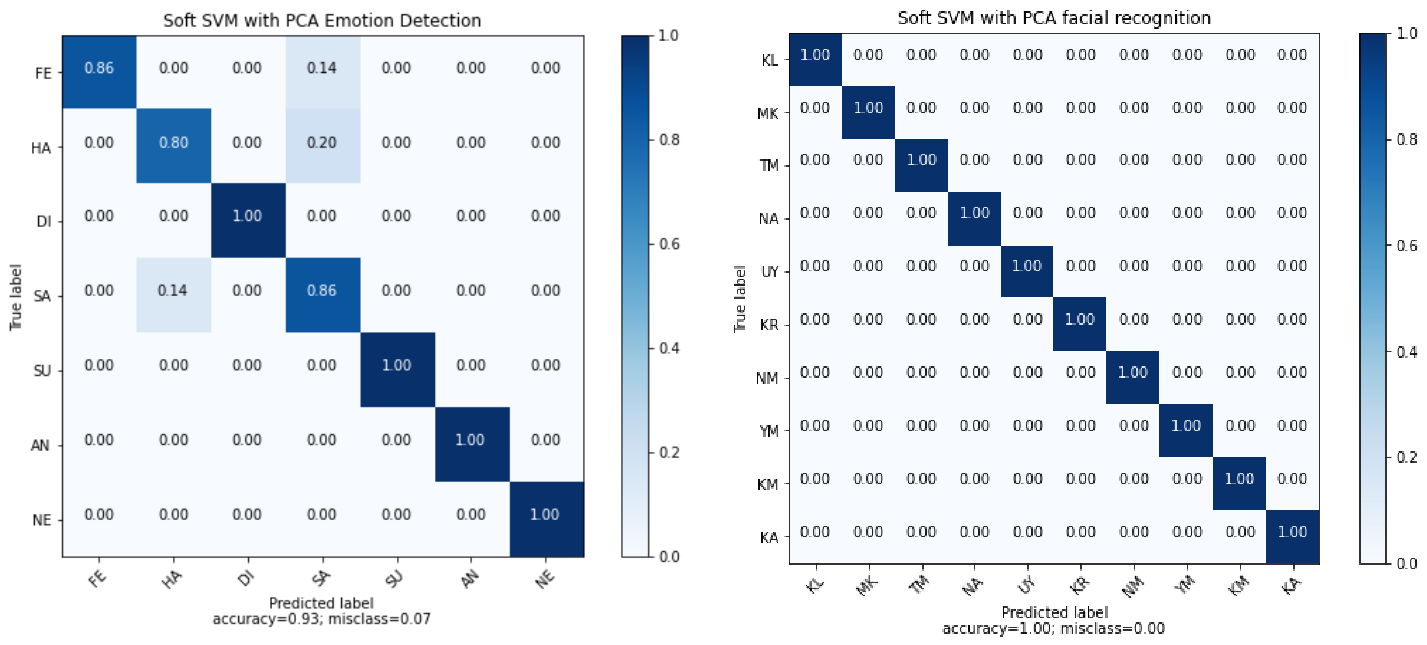


Figure 27. Confusion matrix of soft SVM with PCA for emotion and facial recognition

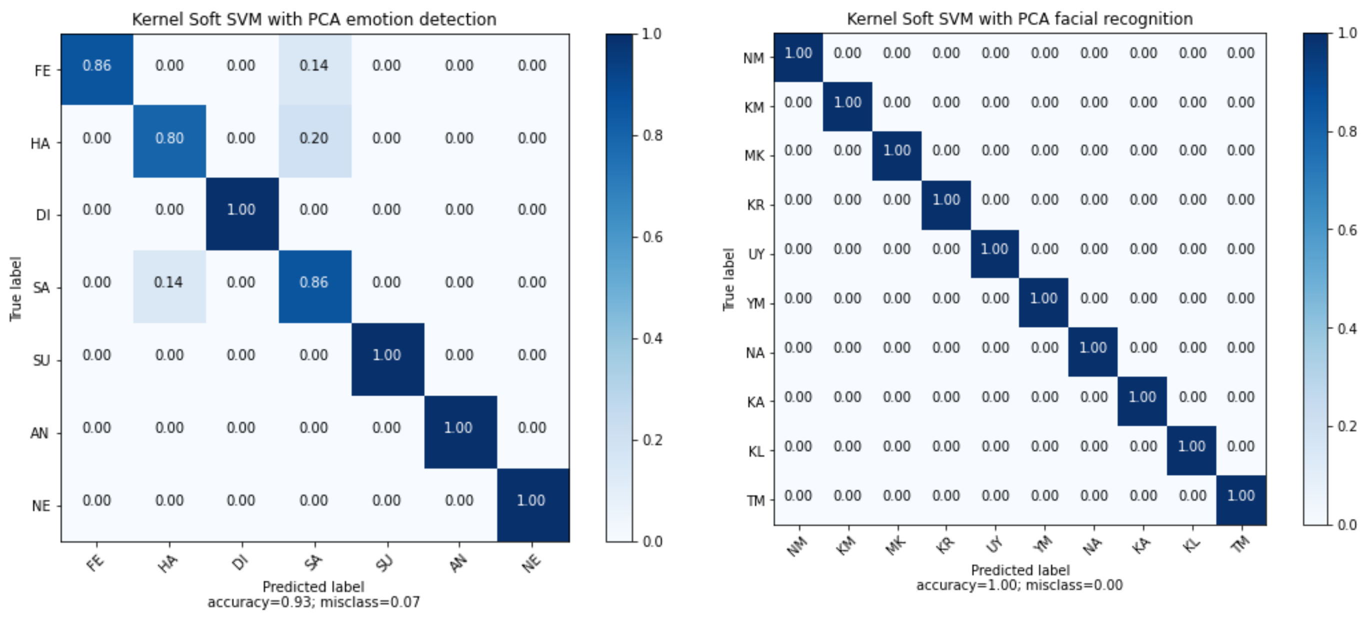


Figure 28. Confusion matrix of hard svm with PCA for emotion and facial recognition

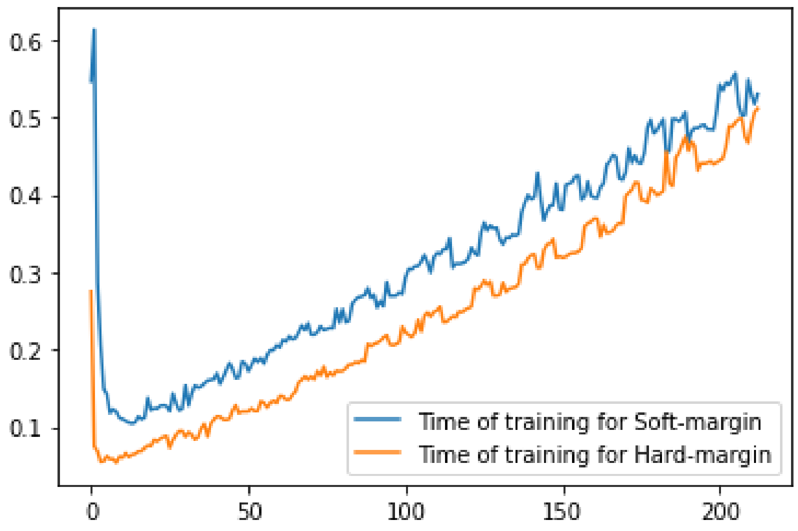


Figure 29. Comparing trainning time of soft and hard SVM for different number of PCs

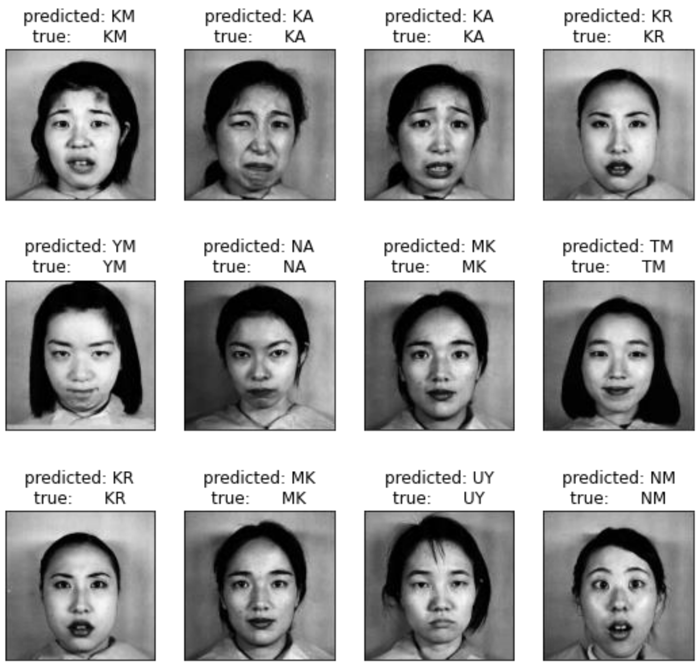


Figure 30. Results of soft SVM facial recognition on a portion of the test set

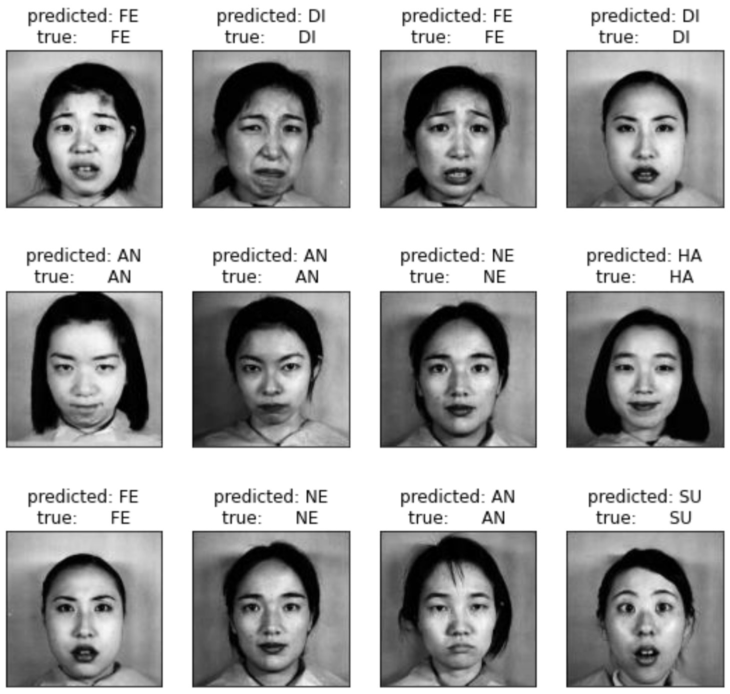


Figure 31. Results of soft SVM emotion recognition on a portion of the test set

2.6 Facial and emotion recognition using LDA and QDA

The LDA and QDA classifiers (before and after PCA) have been applied to both facial and emotion recognition tasks. The results of cross-validation (k=20) LDA before applying PCA was 77.95, and 100 percent for emotion and facial recognition tasks respectively. The results of cross-validation (k=20) QDA before applying PCA was 17.45, and 100 percent for emotion and facial recognition tasks respectively. The similar procedure for finding the optimum number of PCs for classification after applying PCA has been performed for LDA and QDA, and the results are given in Figures 32 to 35.

These results can be discussed. Although LDA classifier has an inherent dimentionality reduction, it does not perform well when the dimention of the data grows too large [7]. This can explain the lower performance of LDA and QDA before applying PCA, and also the performance drop in Figures 33 and 35 as we increase the number of components. Furthermore, the general lower performance of QDA with respect to LDA can be explained. Three examples are offered in pages 152 to 153 of [10] where LDA beats QDA. From these examples we can infer that the added flexibility of QDA would result in higher estimation variance due to the need for estimating some additional parameters. In relatively small datasets, such as ours, this cost will outweigh the lower model bias due to QDA offering a better approximation to the data generation process (DGP) than LDA.

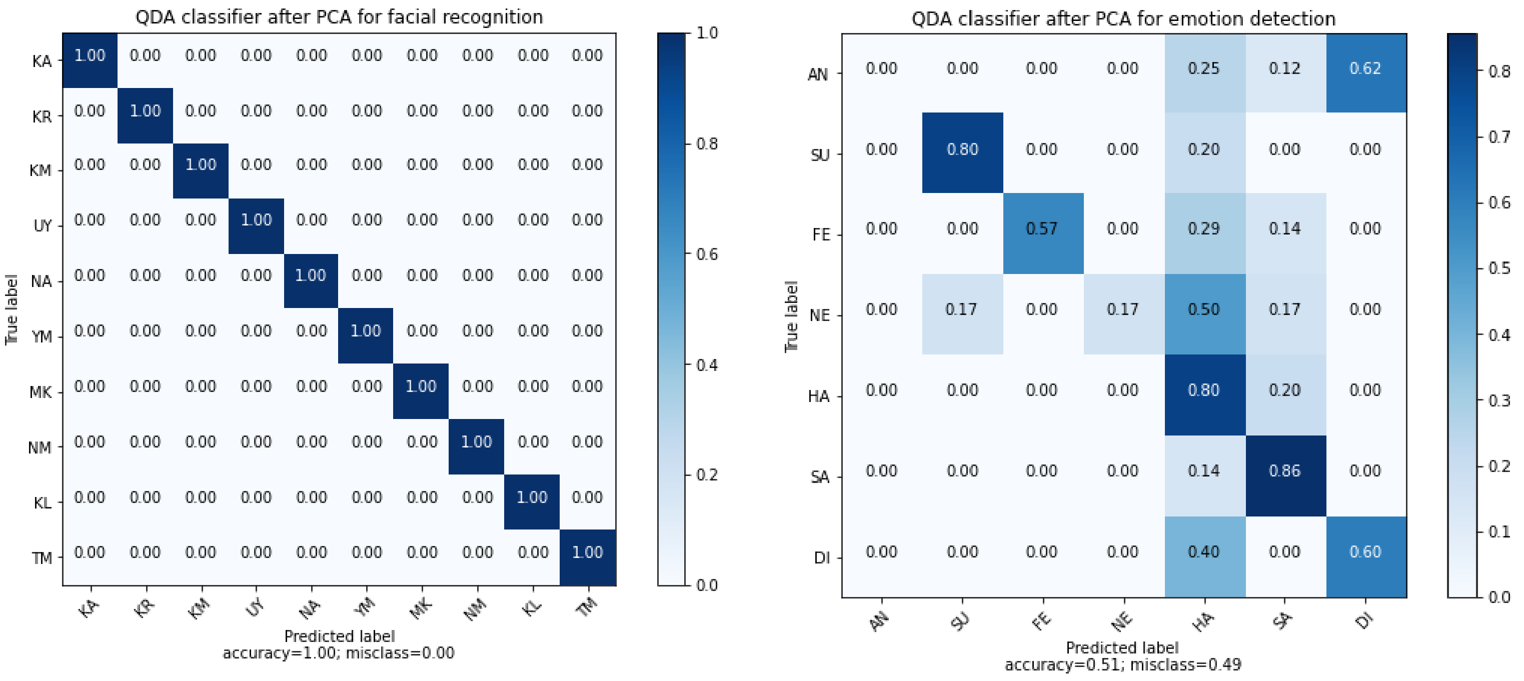


Figure 32. Confusion matrix of QDA with PCA for emotion and facial recognition

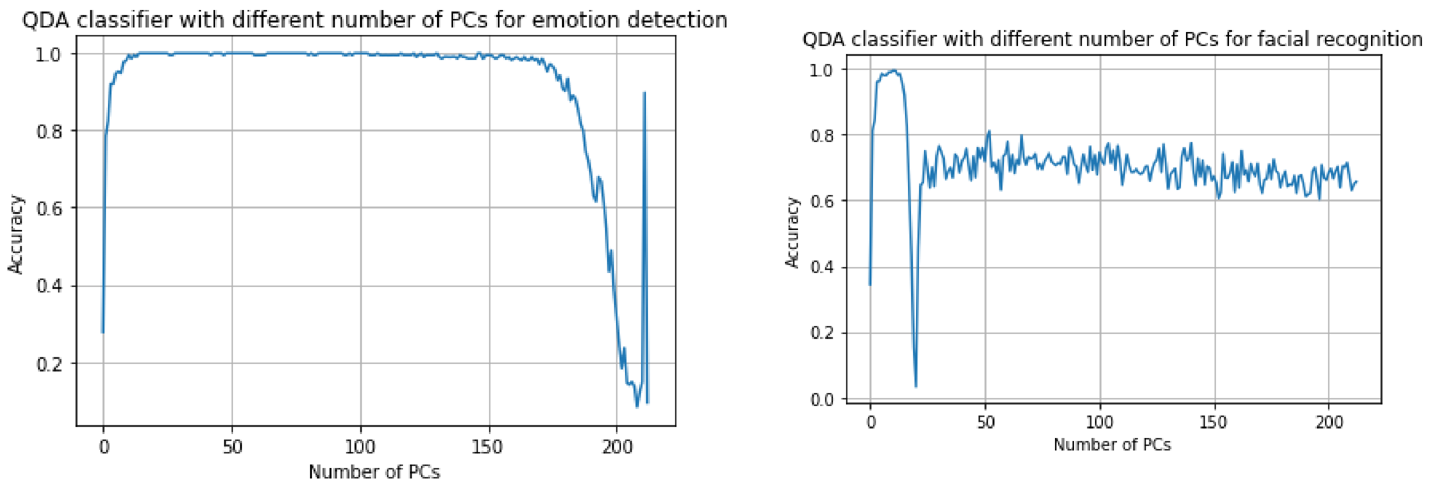


Figure 33. Classification performance of QDA with PCA for facial and emotion recognition as the number of principal components increases

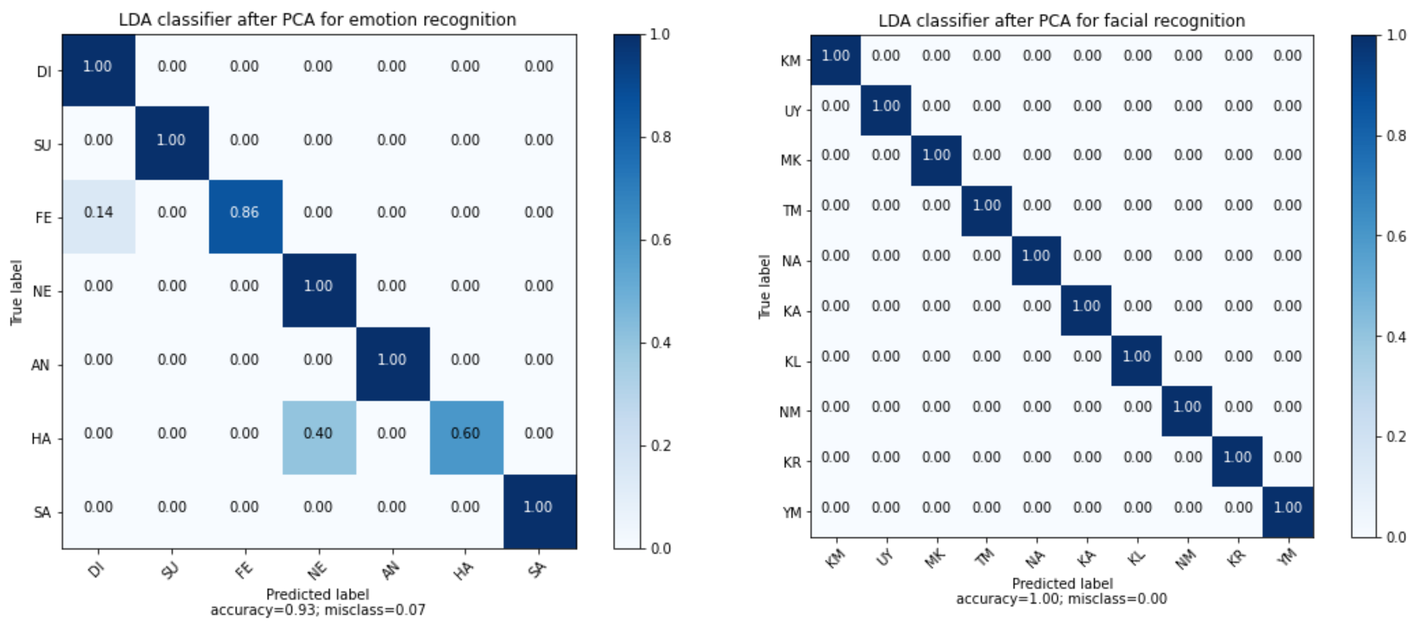


Figure 34. Confusion matrix of LDA with PCA for emotion and facial recognition

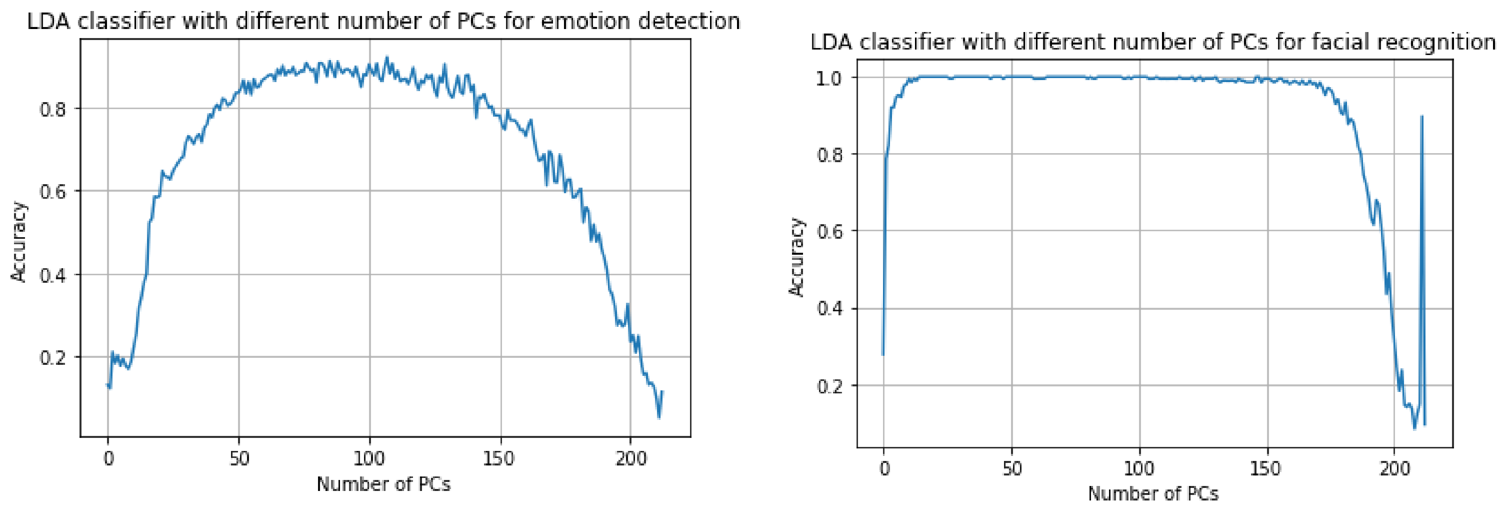


Figure 35. Classification performance of LDA with PCA for facial and emotion recognition as the number of principal components increases

3 Comparing ML and Human Performance on Emotion Recognition

As explained in section 2.1, the dataset comes with semantic ratings of psychological experiments performed on 60 Japanese students, where each subject was asked to rate each one of the six basic facial expressions (emotions) on each image. Some of these data are visualized in Figures 36 to 38. The first two figures show the distribution of two emotions given ratings. Naturally these ratings are almost normal. Figure 38 plots the boxplots for all of the six emotion ratings in a descending order. After analyzing these data, the human performance in recognizing emotions can be calculated by taking the average of each emotion's image ratings. The results of this analysis are given in Table 2.

From this table, and by comparing the per-class and average accuracies of the emotion recognition task with the best-performing machine learning method (the soft-margin PCA-based SVM), we can see that except for happiness, the machine performance is better in detecting the emotions from faces in all other emotions and in terms of weighted average performance.

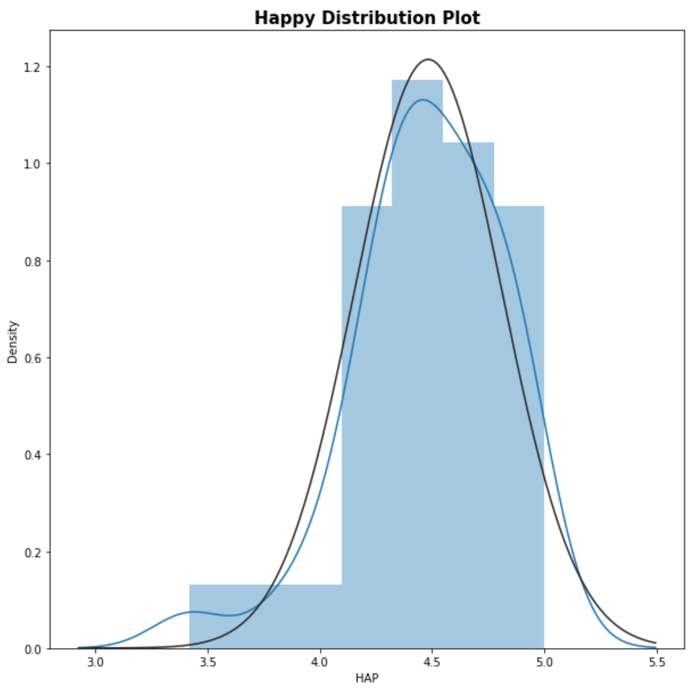


Figure 36. The happiness ratings distribution (along its normal fit)

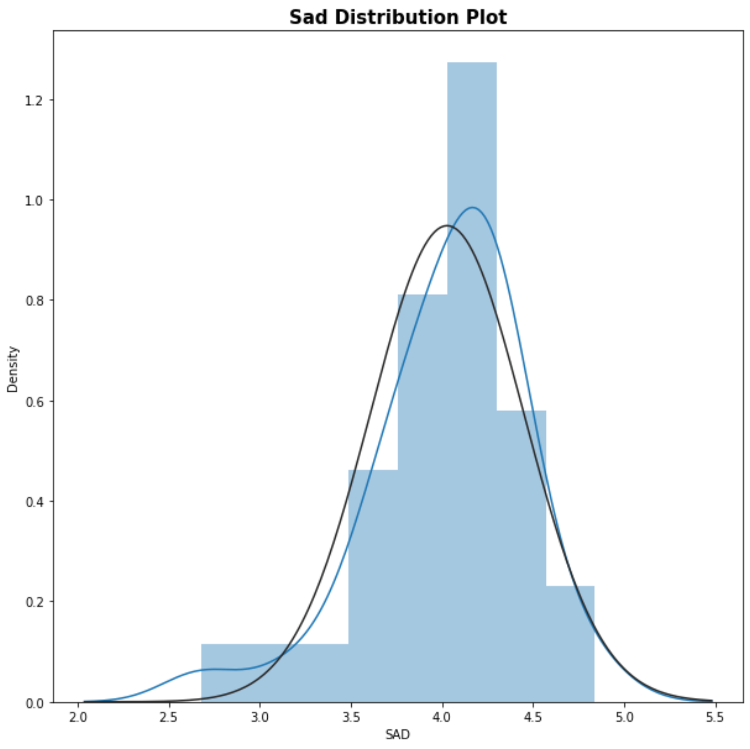


Figure 37. The sadness ratings distribution (along its normal fit)

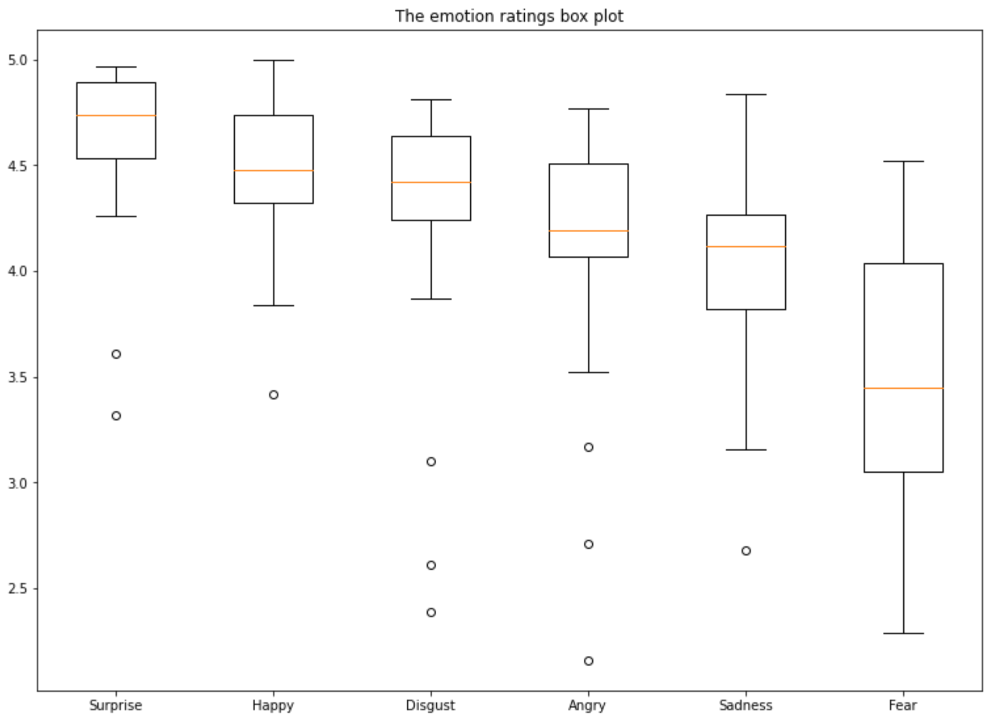


Figure 38. Box plot for different emotions ratings

Table 2. The mean rating of the emotions

Emotion	Mean rating	Number of samples	Accuracy (percentage)
Surprise	4.64	30	92.8
Happiness	4.48	31	89.6
Disgust	4.30	29	86.0
Anger	4.13	30	82.6
Sadness	4.03	31	80.6
Fear	3.50	32	70.0
Weighted Average Accuracy	4.17	183	83.4

4 PCA versus Convolutional Autoencoders

4.1 Introduction to autoencoders

Autoencoders are a specific type of feedforward or convolutional neural networks where the task is to regenerate the input at the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact “summary” or “compression” of the input, also called the latent-space representation. One of the best known use cases of autoencoders is image denoising. An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code (Figure 39).

To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target. We will explore these in the next section.

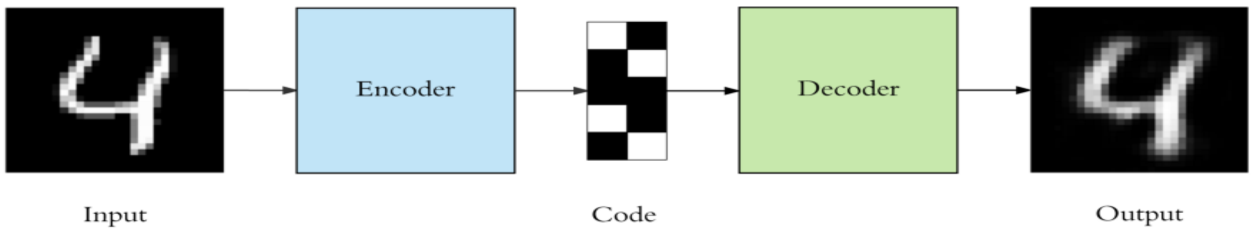


Figure 39. The graphical representaion of an autoencoder

Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. So we can't expect an autoencoder trained on handwritten digits to compress landscape photos.
- Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression they are not the way to go.
- Unsupervised: To train an autoencoder we don't need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.

Architecture

Both the encoder and decoder can be fully-connected feedforward neural networks or ANNs (Figure 40). However, we if the images are high-dimensional, the memory, and computational cost of training feed-forward neural networks will be high. Therefore, we can use convolutional neural networks that are computationally efficient. These autoencoders, use convolutional layers in the encoder section, and upsampling in the decoder section (Figure 41).

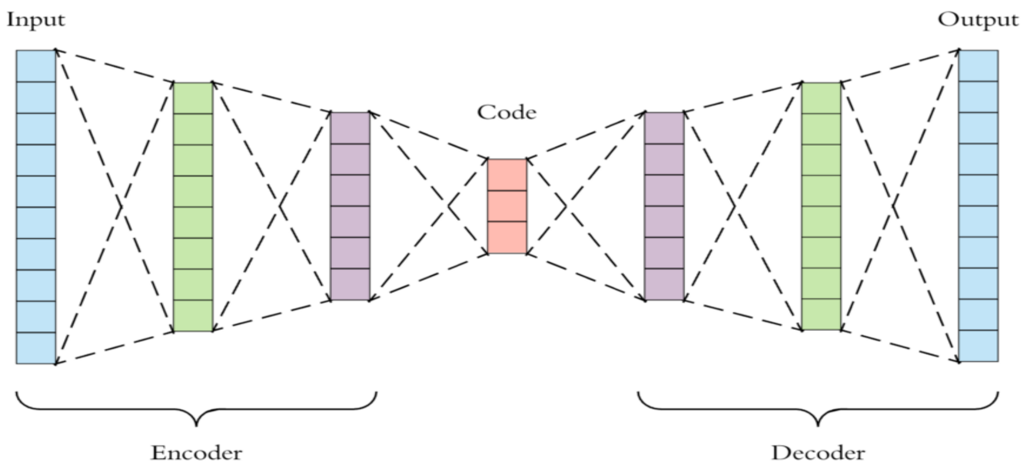


Figure 40. Graphical representaion of a feed-forward autoencoders

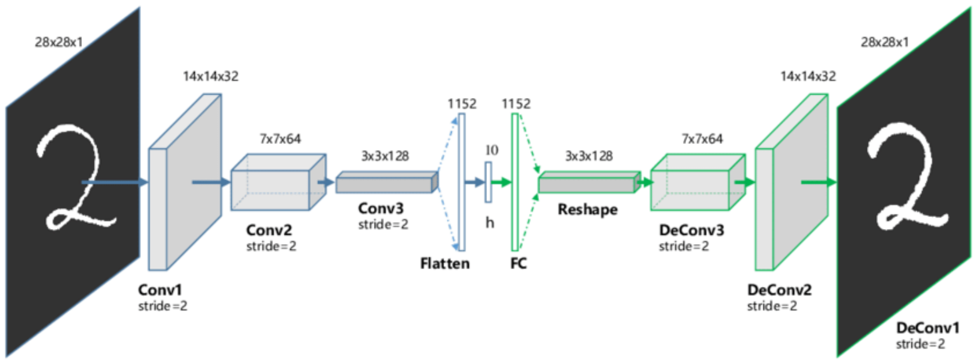
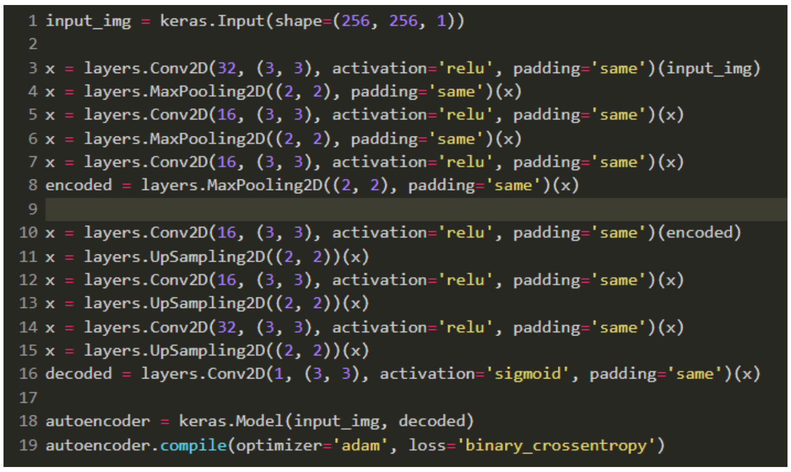


Figure 41. Graphical representation of a convolutional neural network

4.2 PCA image reconstruction compared to autoencoders

The simplest autoencoder is a feed-forward neural network with one hidden layer and a linear activation function. Since, we only have one hidden layer and a linear activation function, this neural network, acts similar to PCA as it first uses a linear transformation to reduce the dimension of the input, and then performs another linear transformation (as the inverse transform in PCA) to reconstruct the data from this low-dimensional representation.

In our case, since the images are 256 by 256 and high-dimensional (compared to 28 by 28 MNIST images), we will compare the performance of PCA reconstruction with a convolutional autoencoder written using Keras and with architecture given in Figure 42.



```
input_img = keras.Input(shape=(256, 256, 1))

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

Figure 42. The architecture of the convolutional autoencoder

Normally, we would expect a neural network to perform better than a conventional algorithm. However, deep learning-based approaches require a lot of training data to perform well. In our case, the dataset is small, and we observe that using PCA with 90 percent explained variance to reconstruct images shows a relatively better performance than the convolutional autoencoder (Figures 43 and 44).

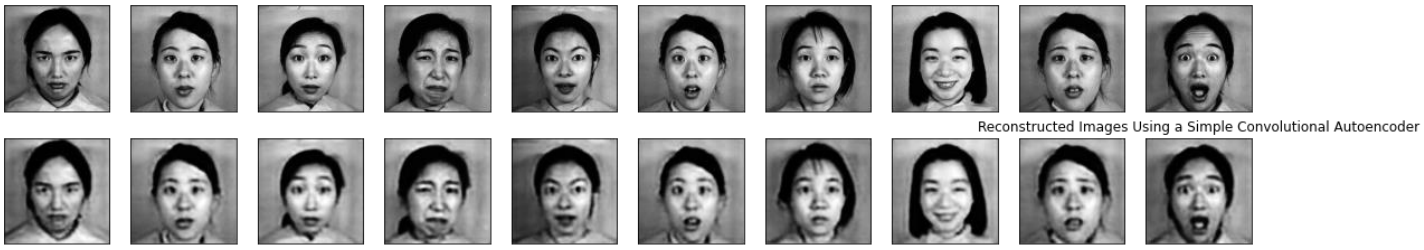

Reconstructed Images Using a Simple Convolutional Autoencoder

Figure43. Image reconstruction with the convolutional autoencoder

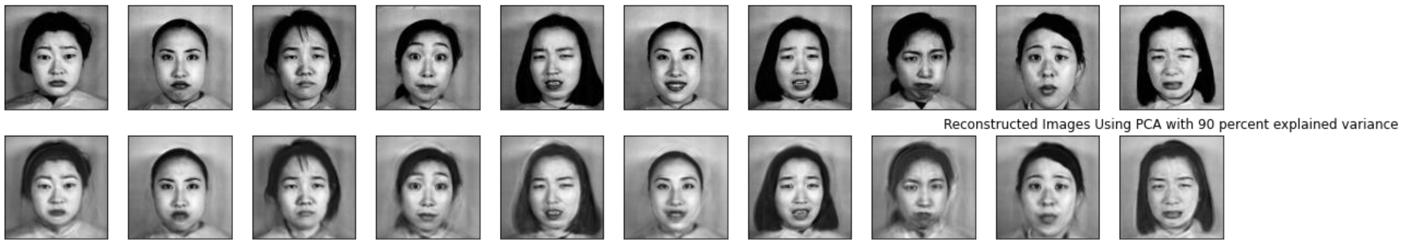

Reconstructed Images Using PCA with 90 percent explained variance

Figure 44. Image reconstruction with 90 percent explained variance PCA

The take-home massage from this analysis is that although we are living in the deep-learning era, our neural networks have a greed for data. When we do not have that data, just like our case, we should resort to the traditional algortihms such as principal component analysis.

References

[1] Jolliffe, Ian T. "Principal component analysis: a beginner's guide—I. Introduction and application." Weather 45.10 (1990): 375-382.
[2] Shalev-Shwartz, Shai, and Shai Ben-David. Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.
[3] Holland, Steven M. "Principal components analysis (PCA)." Department of Geology, University of Georgia, Athens, GA (2008): 30602-2501.
[4] Cristianini, Nello, and John Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. Cambridge university press, 2000.
[5] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. "The elements of statistical learnin." Cited on (2009): 33.
[6] Kroese, Dirk P., et al. Data science and machine learning: Mathematical and statistical methods. CRC Press, 2019.
[7] Yu, Hua, and Jie Yang. "A direct LDA algorithm for high-dimensional data—with application to face recognition." Pattern recognition 34.10 (2001): 2067-2070.
[8] https://zenodo.org/record/3451524#.YPRaXegzZPY
[9] API design for machine learning software: experiences from the scikit-learn project, Buitinck *et al.*, 2013.
[10] James, Gareth, et al. An introduction to statistical learning. Vol. 112. New York: springer, 2013.