Tetiana Prysiazhnyk

Daryna Kosyk

Lilia Kushta

# Project Report

## 1. General topic and the problem

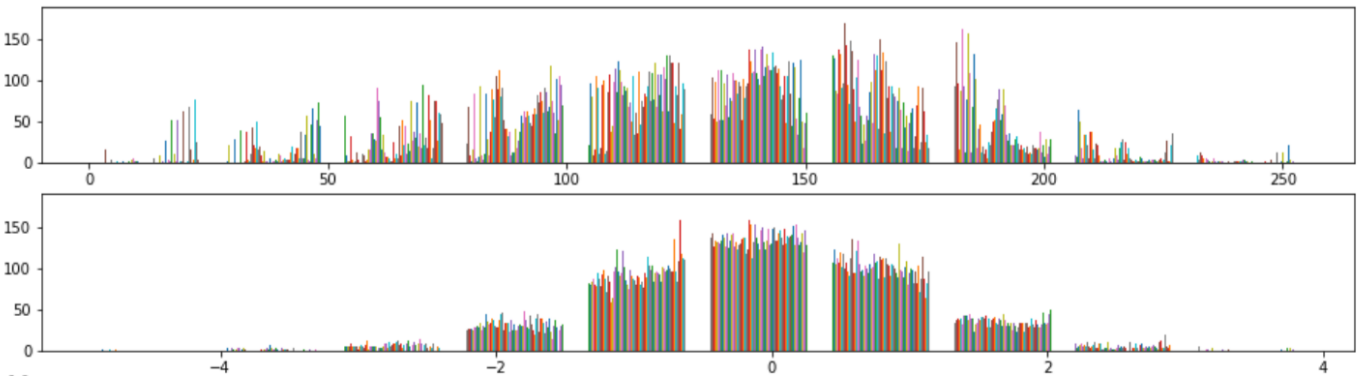
The main idea of our project is to implement a face recognition process. This idea is very suitable for both: our study field as work of the algorithm we use bases on methods of linear algebra, and current situation in Ukraine: on war tools such as surveillance cameras or specific weapons are equipped with face recognition software. To conduct face recognition we have used two methods: deep learning and the computer vision approach. We implement eigenfaces and Siamese neuron network. The aim of this is to compare results and make conclusions on which approach is more effective.

## 2. Dataset

The first idea of our team was to collect our own dataset, which would include photos of us and our family&friens. However, after learning more about the work of the algorithm we understood that it would take a lot of time because for proper work of the algorithm it is necessary to have at least about 60-70 photos. Thus, we took a The Labeled Faces in the Wild dataset. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. But only 8 people had enough photos for the algorithm, so we are predicting only their labels (names). To implement the eigenfaces algorithm and the neuron network we have separated the data into 70%-part on which we have trained the algorithm and 30%-part to test its work and see the accuracy of it.

## 3. Data standarization

After that we conduct standarization of the data. It is a common requirement for many machine learning estimators, because many elements used in the objective function of a learning algorithm assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. On the plots we can observe the results of standarization.




## 4. Eigenfaces

**Eigenfaces algorithm** works in the following way:

- it takes a database of face images, calculates the eigenfaces and determines the face space using all of them
- calculates the set of the weight of a new image
- checks whether the image is face and its resemblance close enough to face space
- Tells whether the image matches any known face in a database

This approach has its own advantages and disadvantages.

**Pros:**

1. the recognition is simple and effective compared to other matching approaches
2. no knowledge of geometry and reflectance of faces is required
3. raw data are directly used for learning and recognition without any processing
4. data compression is achieved by the low-dimensional subspace representation.

**Cons:**

1. its recognition rate decreases for recognition under varying pose and illumination since the eigenface representation is, in a least-squared sense, faithful to the original images
2. the method is very sensitive to scale, therefore, a low-level preprocessing is still necessary for scale normalization
3. though the eigenface approach is shown to be robust when dealing with expression and glasses, these experiments were made only with frontal views. The problem can be far more difficult when there exists extreme change in pose as well as in expression and disguise.
4. another thing is that face images tested in the experiments are taken in the uniform background, while this situation is not suitable in natural scene.

There are also some additional problems due to “appearance-based” nature of the algorithm:

1. learning is very time-consuming, so that makes it difficult to update the face database.
2. recognition is efficient only when the number of face classes is larger than the dimensions of the face space; if it is not like this, the projection of an unknown image requires pixel-by-pixel multiplication of the input image by all eigenfaces

This approach uses dimensionality reduction and linear algebra concepts to recognize faces. The main algorithm of eigenfaces is **Principal Component Analysis**. PCA is a dimensionality reduction technique, it reduces the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally lessen the accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because it is easier to work with smaller data sets and to analyze data. Such data is faster for machine learning algorithms without extraneous variables to process.

Let's Consider a set of $m$ images of dimension $n * n$ (training images). First of all we need to convert each image in the vector of size $n^2$. So, we get $m$ vectors: $\{x_1, x_2, ..., x_m\}$. The next step is to calculate the average of all these face vectors and subtract it from each vector:

$avg = 1/m \sum_{k=1}^{m}(x_k)$ - formula to calculate average.

$a_i = x_i - avg$, $a_i$ is a vector, representing an averaged face.

Then we need to form a matrix A, where the columns are vectors $a_1, ..., a_m$, the dimension of this matrix is $n^2 * m$, as we have $m$ vectors of the size $n * 1$

$A = [a_1 a_2 * * * a_m]$

The next step is to find a covariance matrix - $C$. It is calculated from the product of $A$ and $A^T$. It is clear that the dimension of the covariance matrix is $n^2 * n^2$. This matrix consists of $n^2$ eigenvectors of size $n^2$. You can notice, that if we calculate the covariance matrix as the product of $A^T$ and $A$, the size would be $m * m$, which is much more effective to compute, assuming $m < n^2$.

$Cov = A^T A$ - matrix consists of $m$ eigenvectors of size $m$.

At this step, it is important to show that we calculate eigenvalues and eigenvectors of covariance matrix as:

$A^T A v_i = \lambda_i v_i$

Multiplying by $A$ we get:

$AA^T A v_i = \lambda_i v_i$

$C' u_i = \lambda_i u_i$

Where,

$\lambda_i - eigenvalue$

$v_i - eigenvector$

$C' = AA^T$ and $u_i = A v_i$

We see that $C'$ and $C$ have the same eigenvalues and eigenvectors. So, the $m$ eigenvalues of the covariance matrix $C$ gives $m$ largest eigenvalues and eigenvectors of $C'$.

Now we calculate Eigenvector and Eigenvalues of this reduced covariance matrix $C's$ and map them into the by using the formula:

$u_i = Av_i$

We need to select $k$ eigenvectors of $C'$ corresponding to the $k$ largest eigenvalues, given that $k < m$. The size of each eigenvector is $n^2$.

The next step is to normalize training faces (vector representing face - vector representing average face) $x_i$ and represent each face vector as a linear combination of $k$ eigenvectors.

$$x_i - avg = \sum_{j=1}^{k} w_j u_j$$

$u_j - EigenFacesvector$

Then we take the coefficient of eigenfaces and represent the training faces in the form of a vector of those coefficients:

$x_i = [w_1^i w_2^i w_3^i ... w_k^i]$

The next crucial pert is to test the algorithm

We take a random image with a face. First step is to represent it as a vector of the same dimension as the vector training face. Then we subtract this vector from the average face($avg$):

$\phi = y - avg$

The next step is to project the normalized vector into eigenspace to obtain the linear combination of eigenfaces.

$$\phi = \sum_{j=1}^{k} w_j u_j$$

Then we need to form the vector of coefficients:

$\Omega = [w_1^i w_2^i w_3^i ... w_k^i]$

We take the vector generated in the above step and subtract it from the training image to get the minimum distance between the training vectors and testing vectors:

$e_r = min_l ||\Omega - \Omega_l||$

If $e_r$ is below tolerance level $T_r$, then it is recognised with $l$ face from the training image, else the face is not matched from any faces in the training set.\

After that we apply an **support vector machine** classification model and fit data to the training set. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. (In our case features are labels of people). 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. 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. (Which we have decreased with PCA to ease the work). Using these support vectors, we maximize the margin of the classifier.

We have a training dataset of $n$ points of the form $(x_1, y_1), ..., (x_n, y_n)$, where $y_i$ indicates the class to which $x_i$ belongs. Each $x_i$ is a $p$-dimensional real vector. 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 $w^T x - b = 1$, (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 (not necessarily normalized) 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 $w^T x_i - b \geqslant 1$, if $y_i = 1$, or $w^T x_i - b \leqslant -1$, 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 $y_i(w^T x_i - b) \geqslant 1$, for all 1. We can put this together to get the optimization problem:

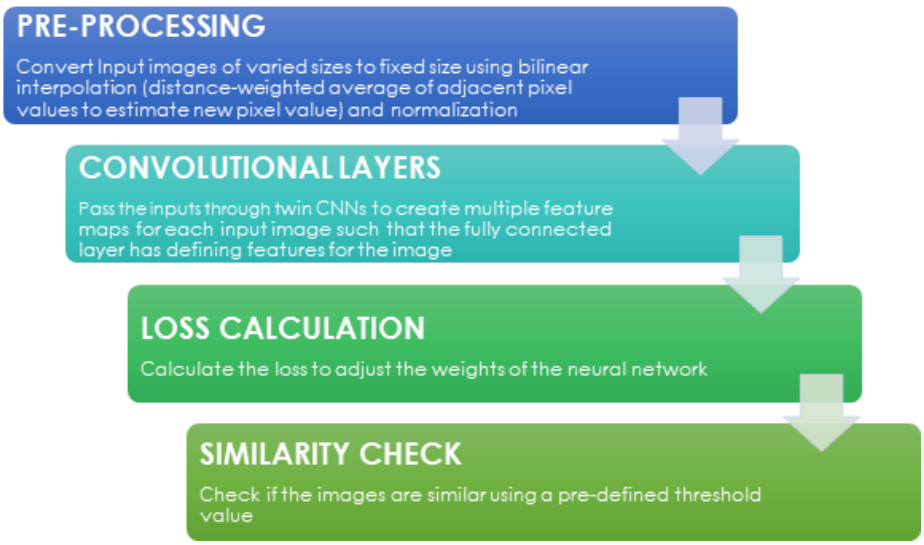
"Minimize $||w||$ subject to $y_i(w^T x_i - b) \geqslant 1$ for $i = 1, ..., n$".

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

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

## 4. Neuron network

**Siamese neuron networks**' main feature is that are built on two or more identical subnetworks, meaning that these networks have the same weights and parameters. They do not need a lot of data to train on comparing to deep neural networks. This is explained by their structure: they are trained to identify classes of data with the help of One-Shot Learning, given just a few images per class.



### Pros:

more robust to class imbalance: with the aid of one-shot learning, given a few images per class is sufficient for Siamese Networks to recognize those images in the future nice to an ensemble with the best classifier: given that its learning mechanism is somewhat different from classification, simple averaging of it with a classifier can do much better than average 2 correlated supervised models learning from Semantic Similarity: Siamese focuses on learning embeddings that place the same classes/concepts close together.

### Cons:

Needs more training time than normal networks: Since Siamese Networks involves quadratic pairs to learn from (to see all information available) it is slower than normal classification type of learning it won't output the probabilities of the prediction, but the distance from each class as training involves pairwise learning.

A siamese neural network is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors. It consists of twin networks which accept distinct inputs but are joined by an energy function at the top. This function computes some metric between the highestlevel feature representation on each side. The parameters between the twin networks are tied. Weight tying guarantees that two extremely similar images could not possibly be mapped by their respective networks to very different locations in feature space because each network computes the same function. Also, the network is symmetric, so that whenever we present two distinct images to the twin networks, the top conjoining layer will compute the same metric as if we were to we present the same two images but to the opposite twins.

Several factors make convolutional networks especially appealing. Local connectivity can greatly reduce the number of parameters in the model, which inherently provides some form of built-in regularization, although convolutional layers are computationally more expensive than standard nonlinearities.

Also the convolution operation used in these networks has a direct filtering interpretation, where each feature map is convolved against input features to identify patterns as groupings of pixels. Thus, the outputs of each convolutional layer correspond to important spatial features in the original input space and offer some robustness to simple transforms. Thus, the outputs of each convolutional layer correspond to important spatial features in the original input space and offer some robustness to simple transforms.

For the neuron network we take the same dataset to compare results of eigenfaces algorithm and this one. We take images from directories,  scale and resize them. Then we make labelled dataset with triplets (anchor, positive/negative, 1/0). Where anchor is a photo of a person, positive/negative shows whether on the next photo is another photo of the same person or another, and 1/0 shows whether on these photos is the same person. We split dataset to 70%-pert to train the neuron system and 30%-part to test it. The next step is to transform each image matrix into model with a vector of features. The model consists of a sequence of convolutional layers, each of which uses a single channel with filters of varying size and a fixed stride of 1. The number of convolutional filters is specified as a multiple of 16 to optimize performance. The network applies a ReLU activation function to the output feature maps, optionally followed by maxpooling with a filter size and stride of 2. Thus the kth filter map in each layer takes the following form:

$$a^{(k)}_{1,m} = max - pool(max(0, W^{(k)}_{l-1,l} \star h_{1,(l-1)} + b_l), 2)$$

$$a^{(k)}_{2,m} = max - pool(max(0, W^{(k)}_{l-1,l} \star h_{2,(l-1)} + b_l), 2)$$

Where $W^{(k)}_{l-1,l}$ is the 3-dimensional tensor representing the feature maps for layer $l$ and we have taken $\star$ to be the valid convolutional operation corresponding to returning only those output units which were the result of complete overlap between each convolutional filter and the input feature maps. The units in the final convolutional layer are flattened into a single vector. This convolutional layer is followed by a fully-connected layer, and then one more layer computing the induced distance metric between each siamese twin, which is given to a single sigmoidal output unit. More precisely, the prediction vector is given as $p = \sigma(\sum_j a_j |h^{(j)}_{1,L-1} - |h^{(j)}_{2,L-1}|))$, where $\sigma$ is the sigmoidal activation function. This final layer induces a metric on the learned feature space of the $(L-1)$th hidden layer and scores the similarity between the two feature vectors. The $a_j$ are additional parameters that are learned by the model during training, weighting the importance of the component-wise distance. This defines a final $L$th fully-connected layer for the network which joins the two siamese twins.

Then we train the model, import Precision and Recall to evaluate results. (Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that were retrieved). After testing our neural network several times, we can see that Siamese neural network and deep learning approach in general works not well with small amounts of data. As we have only 38 photos in positive and anchor folders, the network did not have enough data to train and work good enough and accuracy of recognition was not high. As a result, we got errors in facial recognition quite often. From this we can conclude that Eigenfaces approach works much better on small amounts of data.

## 5. Conclusion

After conducting face recognition with eigenfaces algorithm and with Siamese neurak network, we can conclude that the first method is better. There are several factors which help us to make such conclusion: eigenfaces algorithm works better on smaller dataset, needs less photos to identify a person (about 60 photos is enough) unlike neural networks (which work not really good with such amount of data). Also neural networks need separate trainings for each person, unlike eigenfaces algorithm Moreover eigenfaces algorithm is easier and faster in both: realization and training. However, preprocessing is much more important in the eigenfaces approach.