FACE RECOGNITION USING NEURAL NETWORKS

Image and Speech Recognition Project:
Final report

Martin Čolja
Lovro Ludvig
Ardita Šalja

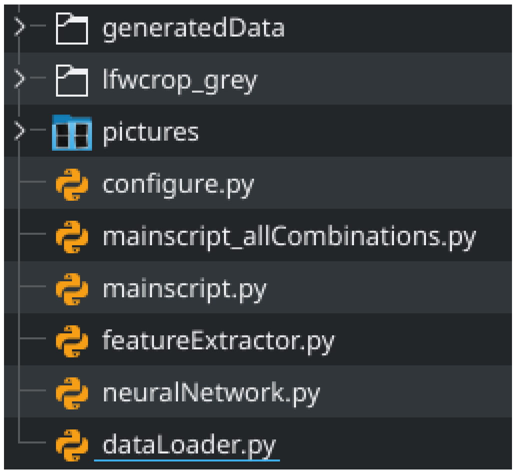
Github: `https://github.com/ledeni56/Face-recognirion-IASR`

1. Introduction

The task of this project was to implement face recognition classifier using Neural network and Eigenfaces. With this paper we will introduce the reader with our approach to that task and look at the results that we got from that approach. Everything was implemented in Python which allowed us to focus on implementation details without worrying about boilerplate code. The classification process was done with 7 classes and the classifier was able to differentiate following classes (we also tried different combinations of classes to see how it behaves, for example with only classifying 5 faces without background and unknown faces):

- Class 1: face of person1
- Class 2: face of person2
- Class 3: face of person3
- Class 4: face of person4
- Class 5: face of person5
- Class 6: unknown faces
- Class 7: background (everything that is not a face)

On the picture 1.1 we can see our project structure. Lets now briefly go over all the files and their usage to later better understand the whole workflow.



Picture 1.1. Project structure

Lets start with the dataLoader.py since it is the first piece of the puzzle. In dataLoader.py there are two classes; DataLoader and DataLoaderHelper. Inside DataLoaderHelper we define methods for saving images (later used to save average face and eigenfaces), loading images and getting image paths. DataLoader class uses those methods to load up all images and save them into NumPy matrices, later we access those images from an instance of DataLoader. All the images being loaded are already cropped and in grayscale, the only preprocessing done by code is adding contrast with PIL library, which is also used for loading and saving images.

Inside neuralNetwork.py there is a class NeuralNetwork, which uses NumPy for working with matrices. NeuralNetwork defines all the necessary methods for working with

Neural network, like training with backpropagation algorithm and classification of a certain input.

The featureExtractor.py file contrains methods for calculating all the necessities for eigenfaces and does it using the NumPy library. FeatureExtractor has to be used only once since it saves generated average face and eigenfaces into .csv files inside /generatedData folder. While generating those matrices it also saves average face and eigenface results as pictures using DataLoaderHelper method to /pictures folder. Besides generating necessary data for eigenfaces it also has methods to prepare testing and training datasets for the Neural network.

Mainscript is used to bring all the pieces of the puzzle together and completes the whole flow of classification. First the DataLoader is initialized to load all the training images. Next all the training images are send to the FeatureExtractor to generate averageFace and eigenfaces. Then ,depending on number of features we want, FeatureExtactor is also used to generate the dataset for training the Neural netwok. After the Neural Network is trained, depending on what set of images we want (train or test images), DataLoader is again used to load them and then pass them to FeatureExtractor to create the dataset for testing the Neural network. After the classification is done we check the results and print out the success rate of classification.

2. Algorithm details

Since we couldn’t afford resources required to construct a deep neural network it was necessary to reduce the input data. Method that saw some success was using principal component analysis or PCA for short. First step is to construct the eigenfaces and to accomplish this we used the NumPy Python library. Since we represented all of the data as a NumPy array, constructing the covariance matrix is almost trivial.

$$\Psi = \frac{1}{N}\sum_{i=1}^{N} X_i \qquad (1)$$

$$\theta_i = X_i - \Psi \qquad (2)$$

All data was collected into a single matrix (equation 1), then translated to a zero-mean state (equation 2), then the covariance matrix was calculated (equation 3).

$$C = \theta^T \cdot \theta \qquad (3)$$

Conveniently NumPy has a built-in function for calculating eigenvalues and eigenvectors, *nupmy.linalg.eigh(covarienceMatrix)* (equation 4).

$$(C - \lambda I) \cdot \vec{v} = 0 \quad (4)$$

After that it was just the matter of selecting the best ones. After experimenting with different values, we opted for the best 100 eigenvalues, since our preliminary choice of 20 - 30 best gave unsatisfactory results. We use those eigenfaces to create a transformation matrix and apply it to the matrix created by translating an input image into a zero-mean state, thereby converting it to 100-dimensional face-space (equation 5).

$$\omega = (Z - \Psi) \cdot U^T \qquad (5)$$

For the architecture of the Neural network, after a certain amount of experimentation, we settled on using 100-dimensional face-space for the input layer, with each coordinate representing one feature of the input. Our hidden layer has achieved best results with 100 hidden neurons. The output layer, depending on what we need to classify, can either have 5, 6 or 7 outputs. For example, if we decide to classify only faces our network will have 5 outputs, however, if we also decide to differentiate between faces and some background, the number of outputs will increase to 6. Finally, the learning rate of 0.05 gave the best classification results. These parameters can be modified dynamically so that we can tweak the network for better performance and quickly change it to meet the type of classification criteria.

Before the training procedure can begin, the complete dataset is split into train and test subsets. We opted for the standard 70% train and 30% test split. Training procedure consists of propagating samples from the training set through the network, calculating the error and propagating it back to the hidden layer, all while adjusting weights in the direction of gradient descent. After the network is trained its performance is assessed on the testing set by propagating the samples from it through the network. We decided to classify samples based on the maximum value found in the output layer. In the case of 5 classes, each class would be represented by a number ranging from 0 to 4, if the sample belonged to the 3rd class and the biggest value was found on the 3rd output, the network has been successful in classifying that sample.

3. Data

In this project we will be using an existing dataset LFWcrop Face Dataset which contains 100 images per individual. All of those images are in grayscale and have resolution of 64x64 pixels. Images are already centered, so there is no need to crop and resize it. Examples of images in our dataset are shown on the picture 3.1.

Picture 3.1. Example of images in our dataset

As negatives, we are using grayscale images of original resolution 800x600 pixels. We resized them to resolution of 64x64 pixels in order to have a consistent dataset. Examples of these images are shown on picture 3.2.

Picture 3.2. Examples of negative images in our dataset

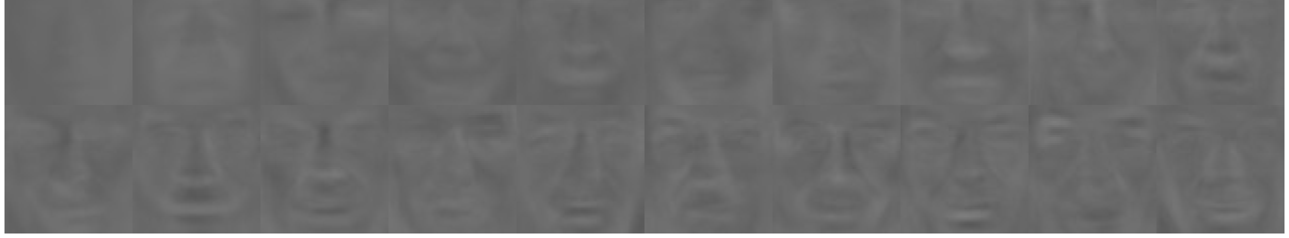
We divide our dataset into training and testing data. Training set contains 70 images per person we want to recognize, 70 images of unknown faces and 70 images of background. In total, that is 490 training images. Testing set contains 30 images per person we want to recognize, 30 images of unknown faces and 30 images of background, resulting in 210 testing images.

In our implementation, we first load these images into NumPy arrays using NumPy library. Then we normalize images by dividing every pixel value with 255. As a result, we get all pixels in range [0, 1]. Furthermore, we enhance contrast of each image using ImageEnhacer by factor 5.0.

Algorithm using eigenfaces is explained in section 2. We first calculate average face by summing up all face images. Calculated average face is shown on picture 3.3.

Picture 3.3. Average face

We feed this algorithm with all faces, including images of known and unknown faces. As a result, we get 4096 eigenfaces as explained in section 2. Out of these 4096 eigenfaces, we can use as many as we want as an input to our neural network. On the picture below, 20 of the best eigenfaces, with largest eigenvalues are shown in picture 3.4.



Picture 3.4. First 20 eigenfaces

4. Results

Our model has several hyperparameters that we can set before training our network. These hyperparameters are learning rate, number of epochs, number of neurons in hidden layer and number of eigenfaces. We tried to find the hyperparameters that work the best by exploring learning rates in range [0.05, 0.2], epochs in range [100, 1000], neurons in range [10,100] and eigenfaces in range [10,100]. We experimentally concluded that combination of learning rate of 0.05, number of epochs of 420, number of neurons of 100 and number of eigenfaces 100 are the optimal hyperparameters.

We evaluate our model of neural network by trying to classify different combinations of datasets to find for which combination our network works best: classifying only five known faces, five known faces and unknown faces, five known faces and background images, five known faces and unknown faces and background.

Since values of weights of the neural network are assigned randomly in the beginning, we evaluate our model ten times in order to get an average success rate. Results for success rated are shown on tables 1-4.

5 faces											
num of ite	1	2	3	4	5	6	7	8	9	10	**AVG**
training	1	1	1	1	1	1	1	1	1	1	**1**
testing	0.77	0.79	0.79	0.79	0.79	0.78	0.813	0.8	0.78	0.8	**0.7903**

Table 1. Results for classifying five known faces

5 faces and unknown											
num of ite	1	2	3	4	5	6	7	8	9	10	**AVG**
training	1	1	1	1	1	1	1	1	1	1	**1**
testing	0.825	0.791	0.783	0.8	0.825	0.825	0.817	0.8	0.816	0.817	**0.8099**

Table 2. Results for classifying five known faces and unknown faces

5 faces and background											
num of ite	1	2	3	4	5	6	7	8	9	10	**AVG**
training	1	1	1	1	1	1	1	1	1	1	**1**
testing	0.783	0.766	0.766	0.783	0.775	0.783	0.783	0.792	0.783	0.792	**0.7806**

Table 3. Results for classifying five known faces and background

5 faces, background and unknwon faces											
num of ite	1	2	3	4	5	6	7	8	9	10	**AVG**
training	0.995	0.993	0.997	0.996	0.998	0.996	1	0.995	0.995	0.998	**0.9963**
testing	0.733	0.728	0.7	0.719	0.7096	0.7096	0.705	0.714	0.742	0.728	**0.71882**

Table 4. Results for classifying five known faces, unknown faces and background

From tables 1-4 we can see that when we try to classify only five faces, our model gives predictions with 79.03% of success rate. When we add one more class, unknown faces, our model works similarly and gives results with 80.99%. If we choose background as an additional class instead of unknown faces, our model still works well, with 78.06% of success rate. Unfortunately, if we try to classify five known faces, unknown faces and background images, our model gives predictions with 71.89% success rate.

From results above, we can conclude that our model works the best if we try to classify five known faces and unknown faces. The training of the neural network on Intel processor i5-8250U@1.6GHz lasted approximately 50 seconds.

5. Analysis and conclusion

We created a face recognition project based on feature extraction using eigenfaces and classifying data by giving it to trained neural network. In our project we first calculated eigenfaces. Then we pull images from our dataset through these eigenfaces to extract features. Those features are then passed as an input to our neural network which gives prediction as an output.

We evaluated our model on different combinations of data, and succeeded to get a success rate of 80.99% when classifying five known faces and unknown faces. Other combinations gave quite satisfying results considering the size and variety of our dataset. The combination with 7 classes (one for each known face, unknown faces and background) seems to work poorly, with 71.82% of success rate. This might be due to the fact that our model recognizes the face in background image, even though there are no faces in it. Furthermore, eigenfaces represent “generic” faces of our dataset, hence, extracting features from background images by pulling them through eigenfaces may produce confusing results. This may be solved by adding face detection first, and then extracting features by using eigenfaces.

There are several things that could improve our model. One of them is more consisted dataset. Our dataset contained pictures taken frontally and in semi-profile. Semi-profile images might be the reason we don’t have better results. It is very hard to recognize a face accurately from all angles. Furthermore, our training dataset contained only 420 face images which may be too small, so a larger dataset could improve our results as well. Some additional preprocessing techniques might also be considered.

Moreover, architecture-wise, our model might work better with more hidden layers and activation functions. We could also consider using deep convolutional networks for face recognition problem.