Real-Time Facial Expression Recognition - Final Report

Intro

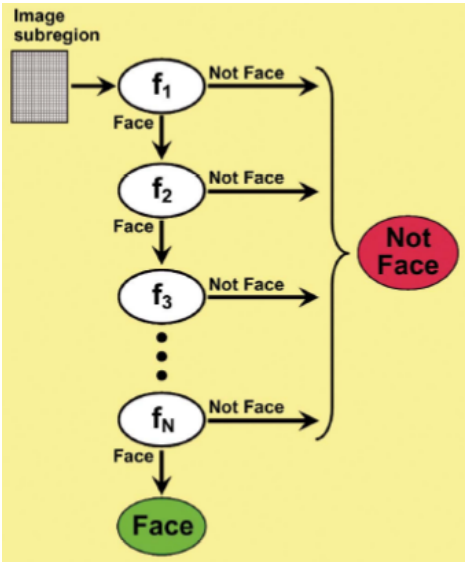
We think that facial expression is among the essential things for human-computer interaction, e.g. smart personal assistants (of course we are talking about artificial intelligence). This kind of personal assistant can threat better if they can recognize emotions of the users. Other application areas are including but not limited to medicine, therapy, education, entertainment, law, marketing… Thus we decided to build an emotion recognizer application.

Abstract

We wanted to build an emotion recognizer application. By utilizing some tools we found by research, this eventually became possible. We took a machine learning based approach and utilized Python programming language and OpenCV library. We utilized Haar-feature based cascade classifiers and “Fisherface” algorithm. Our results gave 90% accuracy when trained and tested on Cohn-Kanade dataset, but results may vary based on different kinds of images, specifically because our dataset was of rather standardized images. In the end, we combined our model with a graphical user interface to recognize emotion real-time with webcams. The result was rather slow due to the complex calculations happening in the background, specifically regarding cascade classifiers. A possible improvement can be usage of a less structured image database to train the model.

Procedure

To train a model that will execute emotion recognition, we first needed a emotion-image database. We acquired dataset of Cohn-Kanade[1]. The number of images were about 500-1000 (depending on the choices on used emotions - we discarded some unclear ones such as anxious and contempt). The images were rather big and the faces held little proportion of the whole image. Thus we decided to specifically extract the faces out of the images. We did Haar-based cascade classification[2], which is a machine learning approach where a cascade function is trained from a lot of positive and negative images.

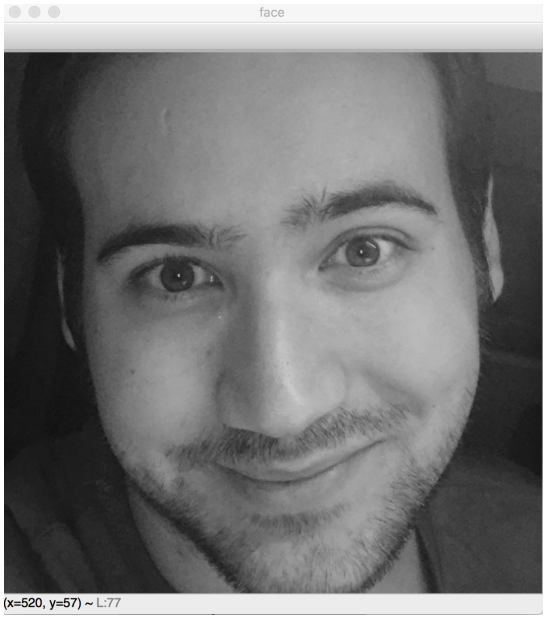


How cascade classifiers work

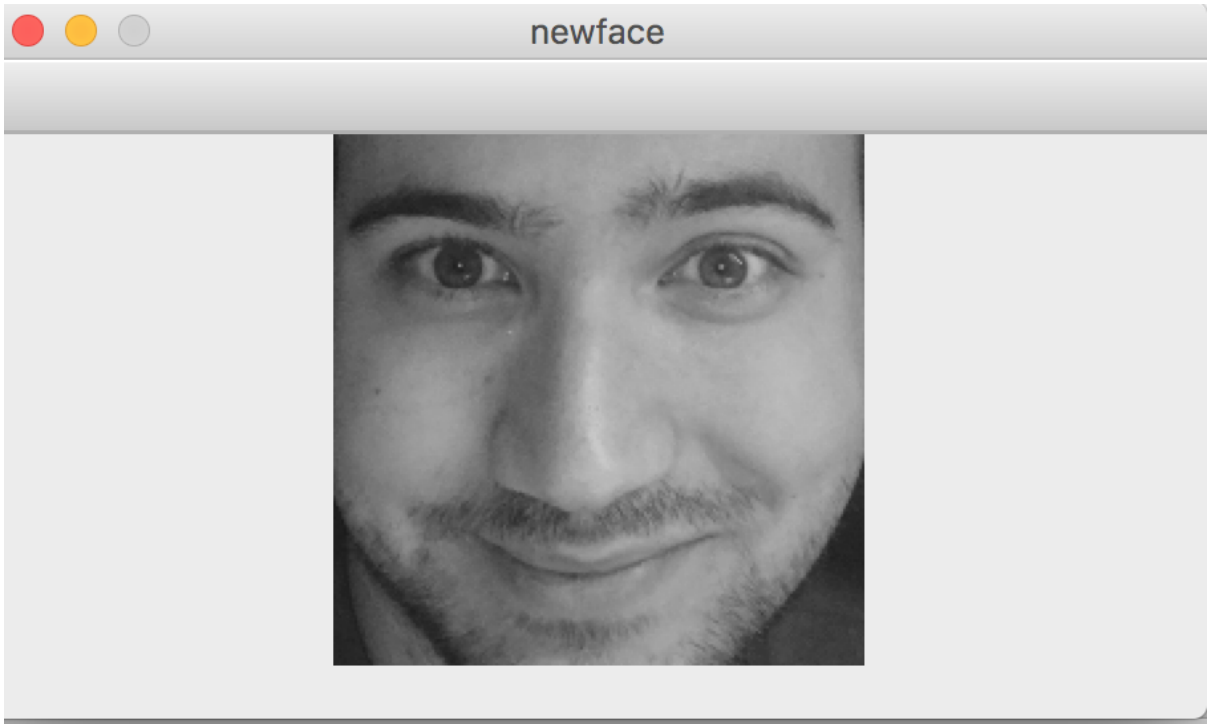
Once the faces were extracted, we further pre-processed them utilizing rotations, scaling, and resizing in order to really create a standard dataset. This process is also called as geometric transformations[3]. Note that we also apply all this extraction and pre-processing in

our real-time program while using/testing our mode, and this is why it doesn't exactly work in real-time, i.e is kind of slow.

A look at pre-processing:



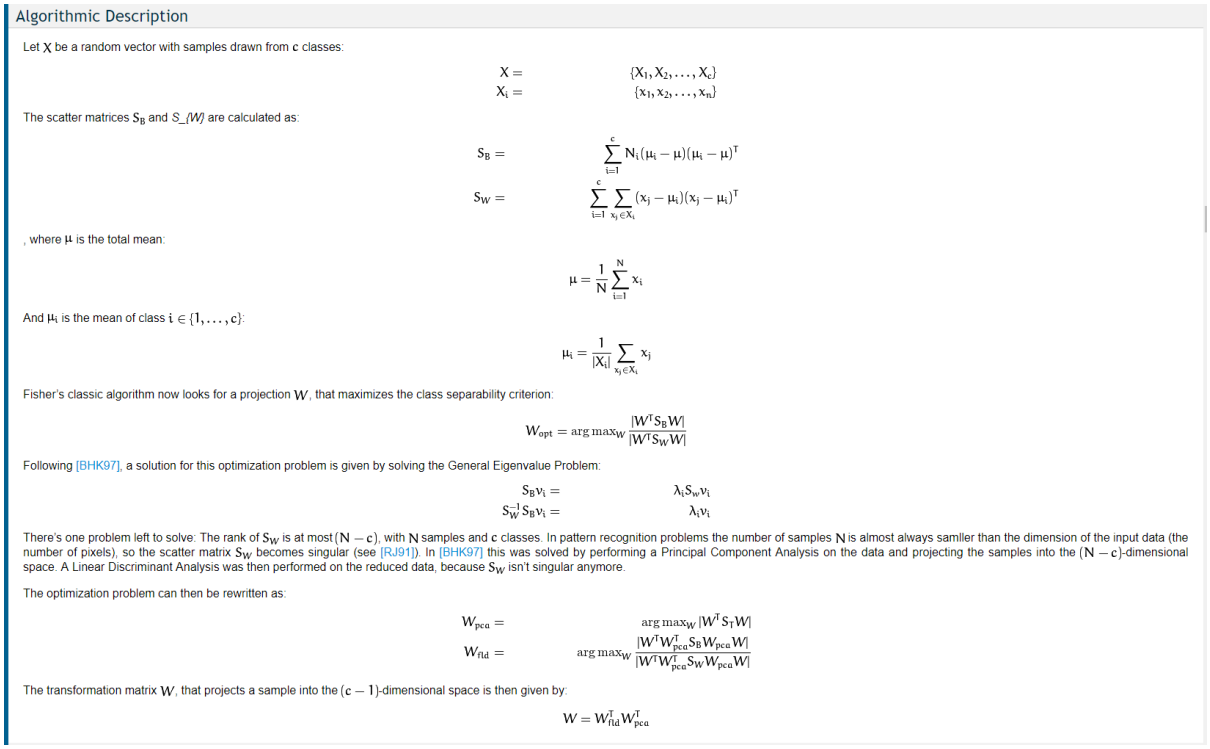
original



rotated, resized (cropped) image

Lastly, we trained a model based on "Fisherfaces" algorithm to recognize emotions. In the algorithm, linear discriminant analysis is used to find the subspace representation of a set of face images, the resulting basis vectors defining that space are known as Fisherfaces. We additionally tried Eigenfaces, and Local Binary Patterns Histograms(LBPH) algorithms as

well, but Fisherfaces gave the best results with ~90% accuracy on our randomly chosen training/test sets in our CK database.



Algorithmic Description

Let X be a random vector with samples drawn from c classes:

$$X = \{X_1, X_2, \ldots, X_c\}$$
$$X_i = \{x_1, x_2, \ldots, x_n\}$$

The scatter matrices S_B and S_{W} are calculated as:

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$
$$S_W = \sum_{i=1}^{c} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^T$$

, where μ is the total mean:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

And μ_i is the mean of class $i \in \{1, \ldots, c\}$:

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j$$

Fisher's classic algorithm now looks for a projection W, that maximizes the class separability criterion:

$$W_{opt} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

Following [BHK97], a solution for this optimization problem is given by solving the General Eigenvalue Problem:

$$S_B v_i = \lambda_i S_w v_i$$
$$S_W^{-1} S_B v_i = \lambda_i v_i$$

There's one problem left to solve: The rank of S_W is at most $(N - c)$, with N samples and c classes. In pattern recognition problems the number of samples N is almost always samller than the dimension of the input data (the number of pixels), so the scatter matrix S_W becomes singular (see [RJ91]). In [BHK97] this was solved by performing a Principal Component Analysis on the data and projecting the samples into the $(N - c)$-dimensional space. A Linear Discriminant Analysis was then performed on the reduced data, because S_W isn't singular anymore.

The optimization problem can then be rewritten as:

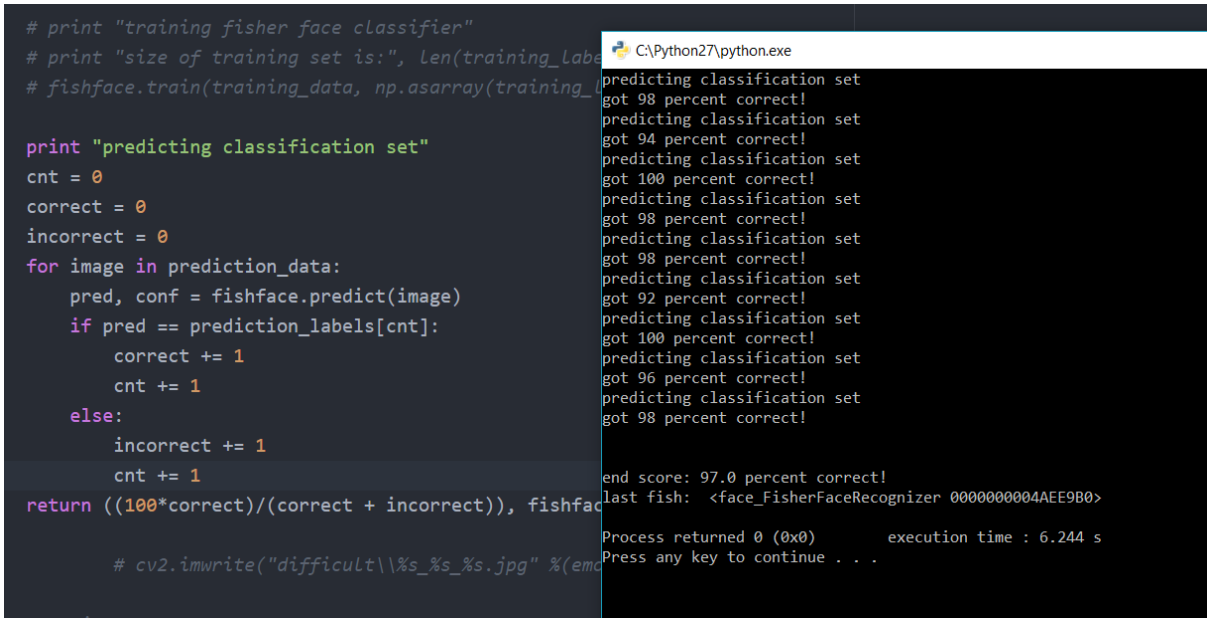
$$W_{pca} = \arg\max_W |W^T S_T W|$$
$$W_{fld} = \arg\max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}$$

The transformation matrix W, that projects a sample into the $(c - 1)$-dimensional space is then given by:

$$W = W_{fld}^T W_{pca}^T$$

Fisherfaces Algorithm[4]

Our results within our tests were satisfying with 90%+ accuracy.



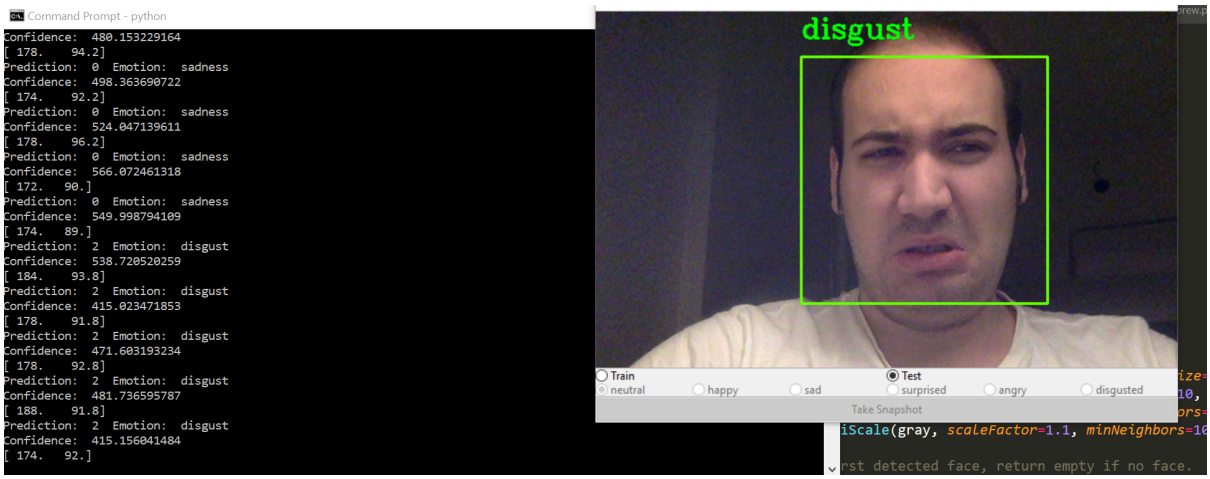
```
# print "training fisher face classifier"
# print "size of training set is:", len(training_labe
# fishface.train(training_data, np.asarray(training_

print "predicting classification set"
cnt = 0
correct = 0
incorrect = 0
for image in prediction_data:
    pred, conf = fishface.predict(image)
    if pred == prediction_labels[cnt]:
        correct += 1
        cnt += 1
    else:
        incorrect += 1
        cnt += 1
return ((100*correct)/(correct + incorrect)), fishfac

        # cv2.imwrite("difficult\\%s_%s_%s.jpg" %(emo
```

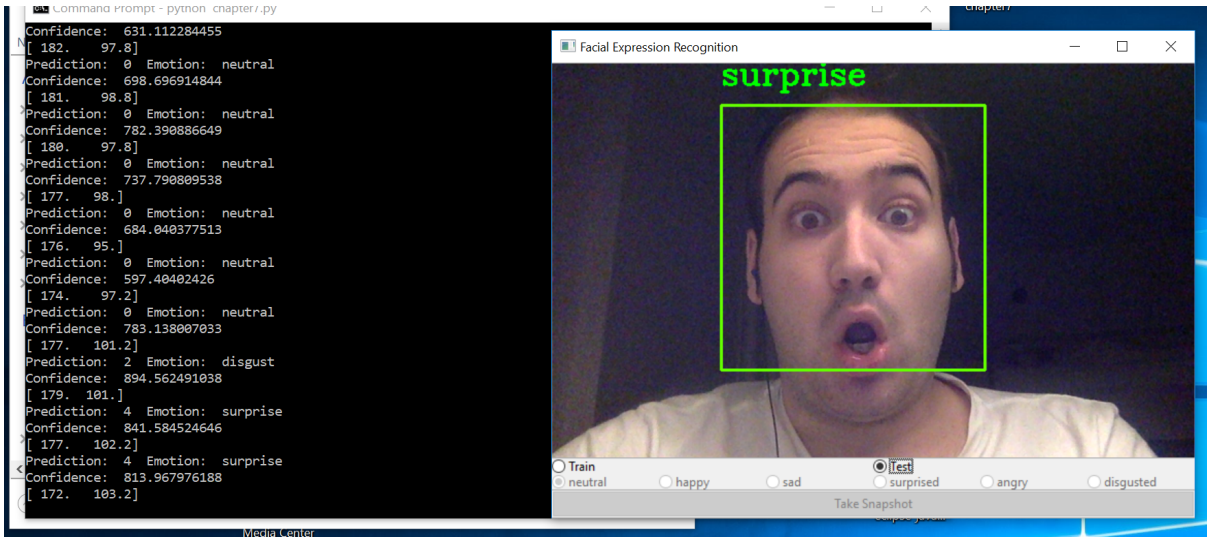
```
C:\Python27\python.exe
predicting classification set
got 98 percent correct!
predicting classification set
got 94 percent correct!
predicting classification set
got 100 percent correct!
predicting classification set
got 98 percent correct!
predicting classification set
got 98 percent correct!
predicting classification set
got 92 percent correct!
predicting classification set
got 100 percent correct!
predicting classification set
got 96 percent correct!
predicting classification set
got 98 percent correct!


end score: 97.0 percent correct!
last fish:  <face_FisherFaceRecognizer 0000000004AEE9B0>

Process returned 0 (0x0)        execution time : 6.244 s
Press any key to continue . . .
```

However, these results are within tests of standard dataset as seen below

```python
def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("dataset\\%s\\*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction
```

And here are the real-time results with GUI application.





Conclusion

Application of haar-based cascade classifiers and fisherfaces algorithm over a standardized dataset gives really great results with more than 90% accuracy. However, when tested in real-time with random webcam images, our result were not perfect due to differences in lighting and posing conditions. Usage of a less structured dataset can improve our real-time results, however as it stands our results are still interesting and encouraging.

References

1 - http://www.consortium.ri.cmu.edu/ckagree/
2 - https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html
3 - https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html
4 - https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#id16