

Emotion Detector Machine

Weikang Wang, Minghao Li, Jiangyang Peng, Qi Zheng

Dept. of Electrical Engineering

Columbia University

ww2461@columbia.edu; ml4025@columbia.edu; zq2306@columbia.edu; jp3724@columbia.edu;

Abstract—Facial Expression Recognition is an interesting but tough task. Recent researches have put large concentration on this. In our project, we want to design an ‘Emotion Detector Machine’, which can analysis the emotion of the input human face by facial expression recognition. We use three different classifiers, KNN (K- nearest neighbor), SVM (support vector machine) and CNN (convolutional neural network). We also use PCA (principle component analysis) to reduce the dimension of input data. The test accuracy of CNN is 35.8%, the SVM is 27% and the KNN is 20%. The SVM based on Landmark and HOG descriptor with PCA is around 26% and with original data is 48%. The SVM with PCA is 6% faster than the SVM with original data.

Keywords: machine learning, digital signal processing, SVM, CNN, PCA, KNN

I. INTRODUCTION

Face recognition is never a novel word and it has been researched for several years. However, facial expression analysis has just been an interesting area that attracts many people to research. By analyzing the facial expression of an input human face picture, the computer can tell which emotion the human holds. This is really an interesting. However, designing a system to realize this is a tough task. It’s hard to define which features correspond to which emotions or facial expression such as happy, mad or sad, etc. Our project supposed to use different approaches to design a system that can help computer to analysis human’s emotion from picture correctly and efficiently.

II. RELATED WORKS

In paper [1], Wang and Zhang use the K-PCA approach to extract the feature of face and do the recognition. Their method can calculate the basic elements in a high dimensional feature space. Paper [2] proposed a new emotion recognition system, which adopts Local Binary Patterns and

Motion History Histogram as its descriptors and use SVM-2K classifier to unify the generation of descriptors. In [3], Liu, Zhang, and Pan designed a CNN model to process the Facial Expression Recognition problem. Their design has several subnet and reach the accuracy about 62.44%. Paper [4] raised an effective facial landmark detection system. It adopts FEC-CNN as the basic principle to calculate the complex nonlinearity and use face-bounding box to enhance the performance of the landmark localization. Xi, Podolak and Lee proposed a method for face recognition by using SVM subsystem to get to features of facial data and construct the face vectors which are analyzed by another SVM machine.

III. TECHNICAL APPROACH

Our project is aims at building a human emotion detector. In order to realize this object, we need to build a suitable classifier to correctly classify the input images with different emotions. We will also provide feature extractor to help the classifiers work better. In the project, we use the KNN (K-NearestNeighbor), SVM (Support Vector Machine) and CNN (Convolutional Neural Network) as our classifiers and use PCA (Principal Component Analysis) as our feature extractor. We compare different test accuracy by utilizing these methods to evaluate our systems.

3.1 PCA (Principal Component Analysis)

In our project, the original dataset we used contain the pixels’ information of over 30,000 pictures. For each of the data points, it has 2,304 dimensions, which is quite a high dimensional dataset. Analysis based on the original dataset takes up a lot time. Actually, there is something

we can do to prevent that from happening, that is, doing dimensional reduction. By deleting some of the dimensions, which are not very important, we can significantly simplify our further analysis.

The photos we used for our project are 48*48 pixels, and that is where 2304 comes from. For each of the pixels, there is a corresponding dimension, which stands for the grayscale value of that pixel. Because all the photos we used are human faces, obviously those faces cannot take up all the 2,304 pixels. In another word, there will always be some pixels that are of zero value, or noisy value. In such cases, there is no doubt that we can confidently delete those pixels, and this will do no harm to our result.

What we mentioned above is only one case where dimensional reduction is needed, and there are some other cases where dimensional reduction can also be applied. After dimensional reduction, we can simply analysis based on the lower dimensional dataset.

One of the many ways of doing dimensional reduction is called Principal Components Analysis (PCA). We can consider our dataset as is composed by 2,304 components (not necessarily to be the same as those 2,304 dimensions). Each of those components is of different importance. Because the principal components contain most of the information of that data point, we can confidently delete the rest of the components without losing much of the information about that data point. This is the general idea of PCA, and we will provide one way to figure out those principal components.

Assume the original dataset has d dimensions, and we will project the data points in that d dimension space into another k dimension space, which has the minimum difference with the origin data point after the projection. Our goal is to figure out how to make that projection.

Assume that there are n inputs of d dimensions, represented by x_i . The projection matrix is an $n*k$ matrix called Q . The projection matrix Q must have the following form:

$$Q = [q_1, q_2 \dots q_k]$$

where each of the q_i is a normalized column vector, and for any q_i and q_j , there is always $q_i * q_j = 0$, that is, q_i and q_j are always orthogonal. If the inputs are centered at zero, that is, the average of each dimension is zero, we can simply calculate the co-variance of the input matrix X , and q_1, q_2, \dots, q_k are the corresponding eigenvectors of the top k eigenvalues. The proof is omitted here. Figure 3-1 shows the relevant coding for doing so.

```
meanVals = np.mean(dataMat, axis = 0)
meanRemoved = dataMat - meanVals
covMat = np.cov(meanRemoved, rowvar = 0)
eigVals, eigVects = np.linalg.eig(covMat)
eigValInd = np.argsort(eigVals)
eigValInd = eigValInd[:-(topNfeat + 1):-1]
redEigVects = eigVects[:, eigValInd]
lowDDDataMat = np.dot(meanRemoved, redEigVects)
```

Figure 3-1

3.2 Classifiers

Another part in our project is to use machine learning algorithm to analyze the data. Here we adopt KNN, SVM and CNN classifiers.

3.2.1 KNN (K-NearestNeighbor)

The general idea of KNN is that given some d dimension inputs, we can find that the data points are distributed in some d dimension space. Then, if two data points lie closely to each other in that space, it means that those two data points are very similar, and there is no much different between those two data points. Thus, if one of the two points belongs to class "A", it is probable that the other data point also belongs to that class.

In our project, we divided the whole dataset into two parts, training part (about 30,000 data points) and testing part (about 3,000 data points). For each testing data points, we need to calculate the "distance" to every training data points. And the testing data points should be in the same class as the training data point which has the shortest distance to it, and that training data point is call its nearest neighbor. We can also find k nearest neighbors, and then the testing data point will belong to majority of that k nearest neighbors.

In our project, the term "distance" mentioned above is defined as:

$$\sum_{n=1}^{2304} (x_{i,n} - x_{j,n})^2$$

where $x_{i,n}$ represents the n -th dimension of the data point x_i . And for each testing data point x_i , we need to find out the training data point x_j which satisfy the following equation

$$\min_{x_j \in X} \left(\sum_{n=1}^{2304} (x_{i,n} - x_{j,n})^2 \right)$$

where X is the input space, which is consisted of the whole training data points.

3.2.2 SVM (Support Vector Machine)

In SVM part, we also use two other methods to extract features from picture, one is landmark and the other one is Hog feature extraction algorithm. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Landmark and HOG are two kinds of descriptor to extract the useful information.

a. Landmark algorithm

shape_predictor_68_face_landmarks.dat is the dataset file for Dlib's facial landmark detector, which helps to extract the whole face, as well as locate the corners, such as the eyes, ears etc. You can see the demo result in figure 3-2 after landmark below.



Figure 3-2

Here we chose to use Dlib package because the code they implemented is very fast and accurate, and it's very clean and documented.

b. Hog algorithm

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) is used as features.

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

Besides, we build up the OpenCV (Open Source Computer Vision) virtual environment to satisfy the package needs, which is a library of programming functions mainly aimed at real-time computer vision. And the reason we chose OpenCV is that it provides facial recognition system including landmarks, Hog algorithms, as well as SVM machine learning library for us to use easily. Another very practical reason is that when we are using OpenCV packages to process the pixels and images, it speeds up 100 times faster than MATLAB or natural python packages like Pillow.

For SVM, it's a support vector machine which constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space and be used for classification, regression, or other tasks like outlier's detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. We can see the mathematical meaning of SVM in Figure 3-3.

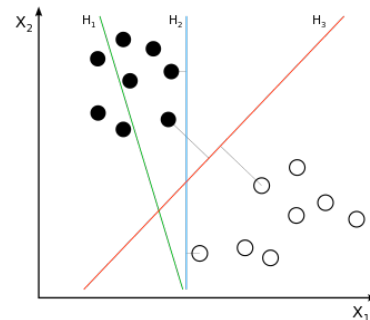


Figure 3-3

H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximum margin.

The basic techniques of the code from GitHub:

<https://github.com/amineHorseman/facial-expression-recognition-svm.git>

3.2.3 CNN (Convolutional Neural Network)

CNN (Convolutional Neural Network) is invented from the MLP (Multiple Layer Perceptron) and is typically designed for classifying image inputs. It consists of different layers such as convolutional layers, pooling layers and fully connected layers. Its biggest advantage is that we need not choose feature extractor manual and just use the original image as the input. The CNN can itself extract proper features of the input and use them to classify efficiently.

Our CNN classifier has 4 convolutional layers, each with a pooling layer behind; 3 fully connected layers and an output with dimension 7, which implies seven different emotion labels. The parameters of filters in each convolutional layer and each pooling layer are shown in the Table 3-1 and Table 3-2 separately.

	1 st conv layer	2 nd conv layer	3 rd conv layer	4 th conv layer
Filter size	5*5	5*5	5*5	5*5
Filter number	48	128	192	192

Table 3-1

	1 st pooling layer	2 nd pooling layer	3 rd pooling layer	4 th pooling layer
Filter size	2*2	2*2	2*2	2*2
Stride	2	2	2	2

Table 3-2

The input and output size of each layer are shown in Table 3-3. Notice that we combine each convolutional layer and its corresponding pooling layer together as a layer.

	1 st layer	2 nd layer	3 rd layer
Input	48*48*3	24*24*48	12*12*128
Output	24*24*48	12*12*128	6*6*192
	4 th layer	5 th layer	6 th layer
Input	6*6*192	3*3*192	1*1*1728
Output	3*3*192	1*1*1728	1*1*1728
	7 th layer		
Input	1*1*1728		
Output	7		

Table 3-3

IV. EXPERIMENTS

4.1 KNN

First, we will analyze the performance of KNN classifier.

It is obvious that KNN is not a very good classifier, because it took up the longest time and gave the lowest accuracy. In our project, there are over 30,000 training data points and more than 3,000 testing data points, which means that we must loop over 100 million times to find out its nearest neighbors. And because our inputs have 2304 dimensions, this makes it nearly impossible to apply KNN classifier on the whole dataset. Instead, we only applied KNN on a smaller dataset, which contains 500 training data points and 200 testing data points. Figure 4-1 shows the outcome of KNN classifier.

```
Jingyangs-MacBook-Pro:Project pengjy$ python3 knn.py
Evaluation time:
100
We set k to be:
1
Evaluation accuracy:
0.2
Jingyangs-MacBook-Pro:Project pengjy$ python3 knn.py
Evaluation time:
96
We set k to be:
5
Evaluation accuracy:
0.215
```

Figure 4-1

Setting different values of k will change the evaluation accuracy. Compared with SVM, it has lower accuracy and takes longer time. Remember, the 103 seconds shown in the figure above is only based on a tiny part of the whole dataset!

And, there is another significant weakness of KNN. Consider the case where in the 4 nearest neighbors of a given testing data point. Two of them belong to class A, and the other two belongs to class B. Then the algorithm is unable to tell which class that testing data point belongs to, and just randomly give one of the classes, which lower the accuracy of KNN classification.

4.2 SVM

Moreover, we will process the data in three dimensions: the origin pictures with 48×48 pixels, that is 2304 dimensions; the dimension-reduced data with only 1000 and 1500 dimensions, using the Principal Components Analysis (PAC) algorithm.

Here we will show the result accuracy for each dimension.

a. Original picture (2304 dimensions) with HOG + Landmark descriptor:

```
[farscode@927 ~]$ cat /etc/conda/activate.d/conda_activate.sh | grep python | xargs python train.py --trainsets - evaluateuoyes
Building Model...
Start training...
Verbal: rfm
Decision Function: evr
Max Epochs: 10000
Training samples: 3280
Validation samples: 3280
/home/farscode/.conda/condaenvs/farscode/python/lib/python3.7/site-packages/sklearn/base.py:1228: UserWarning: Solver terminated early (max_iter=10000). Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(msg, category=UserWarning)
training time = 707.3 sec
early model...
evaluating...
validation accuracy = 48.1
loading dataset Fer2013...
Start evaluation mode...
training preparation mode...
validation samples: 3280
test samples: 3280
evaluating...
validation accuracy = 48.1
text accuracy = 48.1
evaluation time = 27.4 sec
```

Figure 4-2

b. 1000-dimension data with HOG + Landmark descriptor:

```

$ ./fncall-expression-recognition-sw-master$ python train.py --train-yes --evaluate-yes
[WARNING] dataset newdata...
building model...
start waiting...

kernel: x86
decision function: our
max epochs: 10000

training samples: 28789
validation samples: 1794

/home/luqian/virtualenv/fncall-master$ python2.7/site-packages/sleemos/svm/base.py:218: ConvergenceWarning: Solver terminated early (Max
iterations = 100), consider increasing your data with StandardScaler or RVMStandardScaler
(training time = 692.3 sec)
ConvergenceWarning: Solver terminated early (Max iterations = 100), consider increasing your data with StandardScaler or RVMStandardScaler
loading...
validation accuracy = 26.1
validation dataset newdata...
start evaluation...
loading pretrained model...

validation samples: 1794
test samples: 1795

evaluating...
validation accuracy = 26.5
test accuracy = 26.1

```

Figure 4-3

c. 1000-dimension data with HOG descriptor:

```
ster$ python train.py --train=yes --evaluate=yes
loading dataset newdata...
building model...
start training...
..
kernel: rbf
decision function: ovr
max epochs: 10000
..
Training samples: 28709
Validation samples: 1794
..
training time = 246.4 sec
saving model...
evaluating...
- validation accuracy = 26.0
loading dataset newdata...
start evaluation...
loading pretrained model...
..
Validation samples: 1794
test samples: 1795
..
evaluating...
- validation accuracy = 26.0
- test accuracy = 23.8
- evaluation time = 13.5 sec
```

Figure 4-4

d. 1500-dimension data with HOG + Landmark descriptor:

[illegible]

Figure 4-5

e. 1500-dimension data with HOG descriptor:

```
[facecourse-py2] mingham@ubuntu:~/dlab-19.6/faceai-expression-recognition-svm-net$
ster$ python train.py --train=yes --evaluate=yes
loading dataset newdata...
building model...
start training...

kernel: rbf
decision function: ovr
max epochs: 10000
--
Training samples: 28709
Validation samples: 1795
--
training time = 266.5 sec
saving model...
evaluating...
- validation accuracy = 26.0
loading dataset newdata...
start evaluation...
loading pretrained model...
--
Validation samples: 1795
Test samples: 1794
--
evaluating...
- validation accuracy = 26.0
- test accuracy = 23.9
- evaluation time = 13.1 sec
```

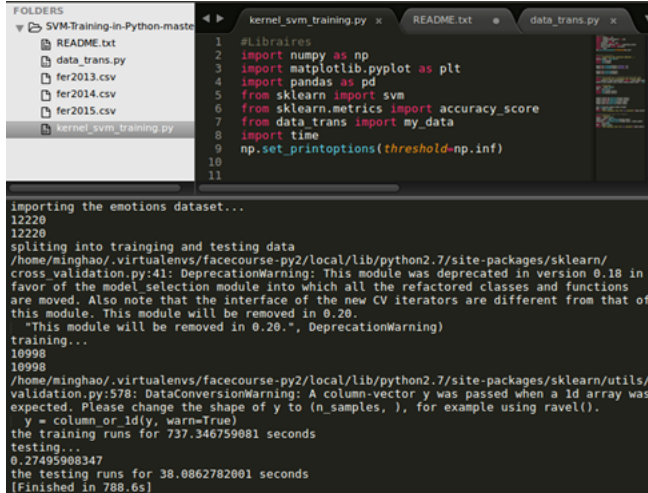
Figure 4-6

From the screen shots above, we can easily find out that as the dimension of the picture increases, the training time is longer, but there's no huge difference in accuracy rate for the cases b and c, our guess is that the descriptor function is so powerful that it can cover the functions of dimension-variation.

But there's one basic variety: the accuracy for origin picture is pretty high to reach nearly 50%. Our explanation is that before reduction, there no useful message missing, even after the functions of two descriptors, it only stores the valuable features such as the landmarks. But the dimension-reduced function PCA is not that

powerful as descriptors to recognize the useful information when detecting faces, which leads to the lower accuracy.

After that, we developed our own kernel SVM algorithm to make a further comparison. The result is list below.



```

FOLDERS
  SVM-Training-in-Python-master
    README.txt
    data_trans.py
    fer2013.csv
    fer2014.csv
    fer2015.csv
    kernel_svm_training.py

kernel_svm_training.py
1 #Libraires
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from sklearn import svm
6 from sklearn.metrics import accuracy_score
7 from data_trans import my_data
8 import time
9 np.set_printoptions(threshold=np.inf)
10
11
importing the emotions dataset...
12220
12220
splitting into training and testing data
/home/minghao/.virtualenvs/facecourse-py2/local/lib/python2.7/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
training...
10998
10998
/home/minghao/.virtualenvs/facecourse-py2/local/lib/python2.7/site-packages/sklearn/utils/validation.py:578: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
the training runs for 737.346759881 seconds
testing...
0.27495908347
the testing runs for 38.0862782001 seconds
[Finished in 788.6s]

```

Figure 4-7

Trained by the pure SVM classifier without any descriptors or dimension-reduced functions, we can see the effect is not only very slow (737s for 12220 pieces of data, 1/3 of the original speed), but not so accurate since many unrelated picture regions are introduced and poorly influence the correctness and recognition.

4.3 CNN

We use the first 30000 samples as the training data and the following 5000 samples as the testing data. We use the TensorFlow as our platform to write the code and use GCP (Google Cloud Platform) to run our code.

Experiment:

The following Figure 4-8 is the picture of loss, training accuracy and test accuracy of training process separately of 750 iterations.

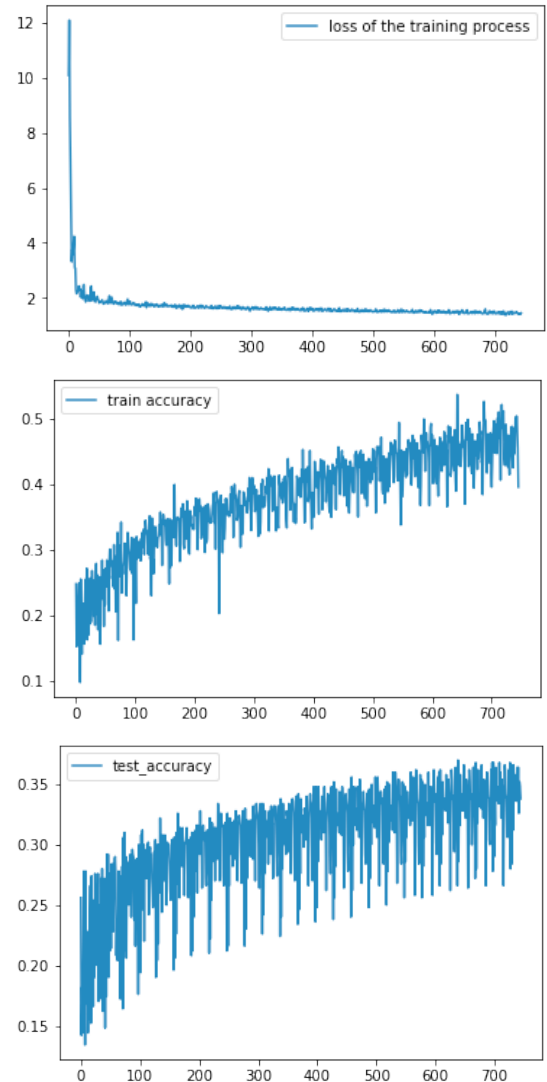


Figure 4-8

Though the training accuracy plot and testing accuracy plot have lots of noise, we can still see that they rise as the iteration grows. As we can see, the training accuracy is not 100% and the loss is not 0. Also, the test accuracy is rising nearly linear. So the training process should be continued. But due to time limited, we have no time to do the training and just get the final test accuracy of 35.8%. We can believe that after enough training, the test accuracy will have a big progress.

V. DISCUSSION

As we can see from the experiment result, the CNN classifier is better than the SVM classifier

and the KNN classifier performs worst. This is because CNN has the strongest ability to represent the function that can describe the input image. It can approximate the distribution of how the input space with different label should like and so have good performance on the test data. The SVM classifier is a linear classifier and it can separate only linear-separable data perfectly. Because data in nature environment is always non-linear-separable, so SVM classifier cannot perform perfectly. As for KNN classifier, its design principle is very easy and it will cause the classifier cannot take enough information from the training data so causing bad performance on test data.

As for PCA method, it reduces the input dimension so it can reduce the computation complexity of the algorithm. In our experiment, we use the SVM with HOG and landmark descriptor as the classifier to test the efficiency of

PCA. The result is it can truly reduce the execution time of the code, but cause a big loss in the test accuracy. We think this is because PCA will filter lots of information of the input data (face picture) and this will cause a big influence on the classifier performance.

REFERENCES

- [1] Yanmei Wang, Yanzhu Zhang, "Facial Recognition Based on Kernel PCA", , vol. 00, no. , pp. 88-91, 2010, doi:10.1109/ICINIS.2010.88
- [2] "Emotion recognition by two view SVM_2K classifier on dynamic facial expression features", , vol. 00, no. , pp. 854-859, 2011, doi:10.1109/FG.2011.5771362
- [3] Kuang Liu, Mingmin Zhang, Zhigeng Pan, "Facial Expression Recognition with CNN Ensemble", , vol. 00, no. , pp. 163-166, 2016, doi:10.1109/CW.2016.34
- [4] Zhenliang He, Jie Zhang, Meina Kan, Shiguang Shan, Xilin Chen, "Robust FEC-CNN: A High Accuracy Facial Landmark Detection System", , vol. 00, no. , pp. 2044-2050, 2017, doi:10.1109/CVPRW.2017.255
- [5] Dihua Xi, Igor T. Podolak, Seong-Whan Lee, "Facial Component Extraction and Face Recognition with Support Vector Machines", , vol. 00, no. , pp. 0083, 2002, doi:10.1109/AFGR.2002.1004