Learning facial expressions from an image

Bhrugurajsinh Chudasama, Chinmay Duvedi, Jithin Parayil Thomas
{bhrugu, cduvedi, jithinpt}@stanford.edu

1. Introduction

Facial behavior is one of the most important cues for sensing human emotion and intentions among people. As computing becomes more human centered, an automatic system for accurate facial expression analysis is relevant in emerging fields such as interactive games (for instance, the games played using Microsoft Kinect), online education, entertainment, autonomous driving, analysis of viewer reaction to advertisements, etc. For example, the reactions of gamers could be used as feedback to improve the gaming experience on systems like the Microsoft Kinect. Similarly, analysis of facial expressions of drivers would help in determining their stress level and such information could be used to alert drivers if they are stressed and in a state unsafe for driving. With these applications in mind, this report describes our attempt to learn facial expressions from an image using deep learning. We also contrast our deep learning approach with conventional "shallow" learning based approaches and show that a convolutional neural network is far more effective at learning representations of facial expression data.

This report is organized as follows. Section 2 describes the problem in more detail along with the dataset used to train and test our algorithm. Section 3 describes the image preprocessing and feature extraction stages. Section 4 describes the various learning models that were applied to this problem and Section 5 presents our results. We analyze and discuss our results in Section 6 and conclude with future plans in section 7.

2. Problem description and dataset

Figure 1 Example images from the dataset

Any learning algorithm that uses features extracted from data will always be upper-bounded in its accuracy by the expressiveness of the features. As a result, a motivation for deep learning based approaches is that learning algorithms can learn, or design features much better than humans can. With this background, ICML 2013 hosted a facial expression recognition challenge

based on a new dataset assembled using images crawled from the internet [1]. In this project we work on the same problem, to classify an image of a human face into one of 7 expressions.
The data consists of 48x48 pixel grayscale images of faces. The seven categories into which the images are to be classified based on facial expression are Angry, Disgust, Fear, Happy, Sad, Surprise and Neutral. The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples. In our project we used the public test set for validation and the private test set to report test error rate.
The number of examples of each class in the dataset are as listed in Table 1. Some example images from the dataset are shown in Figure 1.

Table 1. Distribution of labels in training, validation and test sets

Label	Training examples	Validation examples	Test examples
Angry	3995	467	491
Disgust	436	56	55
Fear	4097	496	528
Happy	7215	895	879
Sad	4830	653	594
Surprise	3171	415	416
Neutral	4965	607	626

3. Features and preprocessing

3.1 Features for shallow learning models

For our experiments, we used two sets of features. In one case, we provided Eigenfaces (described below) as inputs to SVM and shallow neural networks. In the second set of experiments, we tried improving the accuracy of the SVM and shallow neural net models by providing Gabor-filters as the input features. Gabor-filters are linear filters used for edge detection in image processing and the frequency and orientation representations of Gabor filters are similar to those of the human visual system. They have been found to be particularly appropriate for discrimination.

1. Eigenfaces - We mean-adjusted the images, performed PCA on these and then extracted the top 65 principal components from them. Once this was done, the training/test samples were then mapped to these principal component dimensions.
2. Gabor-filters with Adaboost - We used 72 different Gabor filters (9 frequencies, 8 orientations). The frequency parameter determines the thickness of the detected edge and the orientation determines the angle of the detected edge. This gave us 48*48*72 features for each sample. In order to reduce the dimensionality of this feature space, we ran Adaboost feature-reduction algorithm (with DecisionTree classifier of depth one) on the training set [2,3]. The Adaboost classifier assigned an importance value to each feature and we selected only those features with an importance value greater than 0. This helped reduce the number of features from 48*48*72 to around 1200.

3.

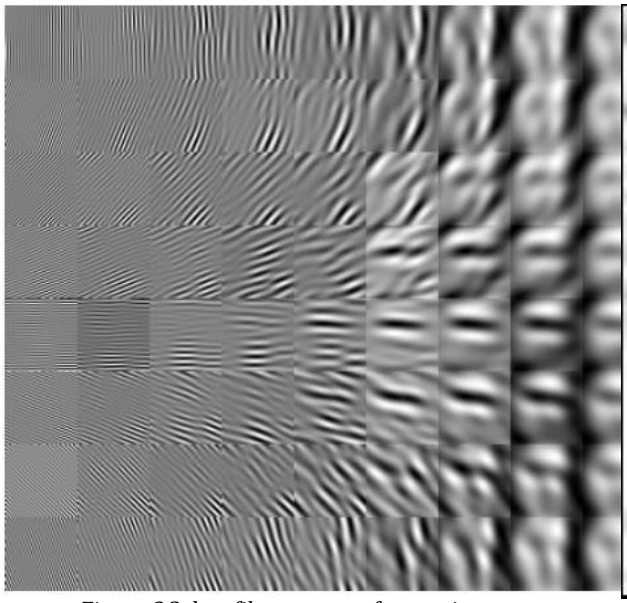

Figure 2Gabor filter outputs for one image

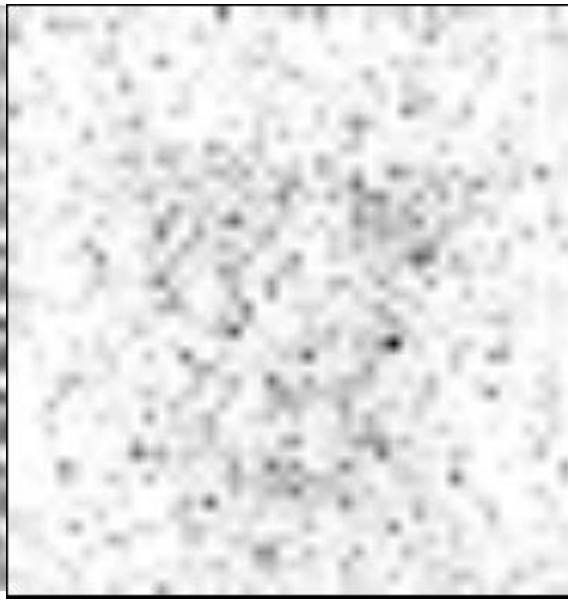

Figure 3 Adaboost results applied to a 48*48 image. Darker pixels indicate higher importance.

3.2 Preprocessing for the deep learning model

We first subtract the mean of each image from each pixel in the image. Next, for each pixel, the mean value of the pixel across all images is subtracted from the pixel and it is divided by its variance. This is the only preprocessing done on the data before it is fed to our convolutional neural network. The convolutional neural network learns the appropriate features from this normalized data.

4. Learning Models

Our roadmap was to benchmark deep learning against conventional supervised learning techniques

4.1 Shallow learning models.

We chose Support Vector Machines (SVM's) and conventional "shallow" neural networks and trained them using the extracted features.

1. *SVM* - We implemented a SVM using the scikit-learn [4] library in Python. The library presents an easy API for training, testing and cross validation for our SVM. Since we have a multi class problem, the library implements a "one against one" approach as described in [6]. We did a grid search over various values of the SVM parameters to obtain the best possible accuracy from our feature set.

2. *Shallow neural network* - We implemented a standard three layered neural network using the Theano [5] library in Python. The fully connected hidden layer has

3723 neurons and an output layer has 7 neurons, one corresponding to each class of labels. The number of hidden layer neurons is chosen to be three times the number of neurons in the input layer. The hidden layer implements a $tanh$ activation function. Since the neural network takes a long time to train, we ran the training on NVIDIA Tesla C2070 GPU's.

4.2 Deep learning model

Next, we implemented Convolutional Neural Network [7], [8] for our problem. The network architecture is loosely based on the approach described in [9]. The network has an input layer, three convolution - max pooling layers, one fully connected hidden layer and an output layer with 7 neurons, one corresponding to each class (similar to the shallow net model). The number of convolution kernels in the three hidden layers are: 32, 32 and 64 respectively and the size of the convolution kernel in each layer is 5x5.

5. Results

In this section we compare the relative performance of the three implemented learning models on our dataset. Figure 4 shows the training error and validation error of the different models on the dataset.

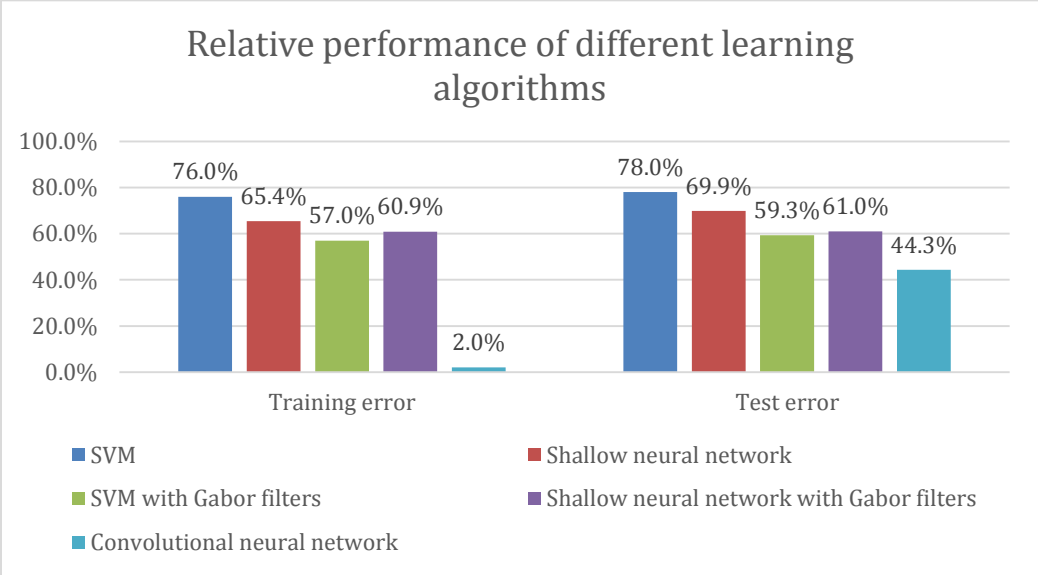

Figure 4 Training and test error rates for different learning models

Clearly, convolutional neural networks outperform shallow learning approaches.

6. Discussion and improvements in the model

The low training error on the convolutional network compared to the high test error indicates that the model is suffering from over-fitting of the data points. To gain more insight into this issue, we derived a confusion matrix, shown in Table 2. Each row in Table 2 corresponds to an expected class label, and each column corresponds to the predicted class label. Thus the entry 24.0747 in the second row, first column indicates that the class "Disgust" is misclassified as the class "Angry" with 24.0747% probability. Based on the analysis of this data, we arrived at the conclusion that the maximum confusion occurred among the classification of the negative emotions (fear, disgust, anger, sad). There was also some confusion in the classification of the positive emotions (happy, surprised, and neutral). To overcome this issue we came up with a two stage hierarchical classification approach:

Stage 1: classify between positive and negative emotions

Stage 2: classify among the emotions in each category

With this, we are getting around 70-80% accuracy for positive classes and around 50-60% accuracy for negative classes, which is already around 10% improvement over the single-stage approach. In spite of the relatively inaccurate binary classifier, this takes us very close to state-of-the art results, which is 69% for the entire set.

Table 2. Confusion matrix for deep learning

	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Angry	49.78	0.22	10.31	9.87	15.79	3.07	10.96
Disgust	24.07	38.89	11.11	3.7	14.81	0	7.41
Fear	13.5	0.61	36.4	7.77	20.86	10.43	10.43
Happy	3.89	0	2.4	76.91	6.17	2.97	7.66
Sad	16.88	0	12.26	13.06	38.85	2.87	16.08
Surprise	4.94	0	6.17	5.68	3.95	74.07	5.19
Neutral	12.31	0.17	7.42	13.66	16.36	2.36	47.72

7. Future plans

We were resource limited in our shallow learning approaches. Mainly, we could not run Adaboost on all our training data. In the future, we want to run adaboost on all training samples, maybe implement the adaboost algorithm on the GPU. Also, the stage 1 binary classifier used in the 2 stage hierarchical classification approach mentioned above needs to be fine-tuned. If we can make this classifier reliable, with the accuracy we are getting for the classifiers in the second stage, we would be able to get the same or higher accuracy than the best results achieved on this dataset. Our ultimate goal is to develop an end-to-end system out of our model and take it to the mobile domain. Mobile devices have already started shipping with GPU's. If we can have the convolutional neural network on the mobile device predict a facial expression from a snapshot of the user's face in soft real time, this could be useful in a wide range of applications.

References

[1] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[2] Bartlett, Marian Stewart, et al. "Recognizing facial expression: machine learning and application to spontaneous behavior." *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE, 2005.

[3] Zhu, Ji, et al. "Multi-class adaboost." *Statistics and Its* (2009).

[4] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *The Journal of Machine Learning Research* 12 (2011): 2825-2830.

[5] Bergstra, James, et al. "Theano: Deep learning on gpus with python." *NIPS 2011, BigLearning Workshop, Granada, Spain*. 2011.

[6] Knerr, Stefan, Léon Personnaz, and Gérard Dreyfus. "Single-layer learning revisited: a stepwise procedure for building and training a neural network." *Neurocomputing*. Springer Berlin Heidelberg, 1990. 41-50.

[7] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks* 3361 (1995).

[8] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[9] Tang, Yichuan. "Deep learning using linear support vector machines." *Workshop on Challenges in Representation Learning, ICML*. 2013.