Facial Emotion Recognition using various Classification Algorithms

Onzali Suba, Nitya Pydipati, Sarvari Ventrapragada

Abstract

Computers need to go beyond understanding human speech and text to recognizing the emotions that accompany their interactions to effectively communicate with humans. In this project, we have described methods for various machine learning algorithms like SVM, Neural Nets and Convolutional Neural Nets to recognize human emotions from a database of images. We explore various pre-processing methods to improve the classification accuracy. We observe the best results amongst all the methods described above with a Convolutional Neural Network with an accuracy of 97.2% on the testing set.

Introduction

As artificial intelligence systems are playing an increasingly important role in our everyday lives, we believe the next frontier in Human Computer Interaction is the ability for machines to understand the emotional state of individuals. Human beings express emotions explicitly and implicitly in various ways, through body language, voice intonation and facial expressions. Comprehending these nonverbal cues can lend machines more context to seemingly bizarre situations and enable a more natural way of interacting with humans. Facial expression Recognition (FEA) is one of the most practical ways to identify the underlying emotional state. Facial Emotion Recognition has found profound use in many fields including Medicine, E-learning, Marketing, Entertainment, Driver-safety, Law, etc.

Goal & Methodology

The goal is to determine the emotion that a person's face emotes with the maximum accuracy of at least over 80%. We use the seminal Cohn Kanade database [1] to detect emotions from expressions of people displaying the emotions by making use of various feature extraction and classification techniques.

Dataset

The CMU-Pittsburgh AU-Coded Facial Expression or the Kanade, Cohn, & Tian, 2000 as it is popularly known has been widely used in the research for facial emotion recognition. It has been one of the base databases to examine various features and from that extract expressions that denote emotion of the person at that instant.

For the CK+ distribution, we have further used an augmented dataset that includes 593 sequences from 123 subjects. The image sequence varies in duration (i.e. 10 to 60 frames) and incorporate the onset (which is also the neutral frame) to peak formation of the facial expressions. Participants were 18 to 50 years of age, 69% female, 81%, Euro-American, 13% Afro-American, and 6% other groups. Participants were instructed by an experimenter to perform a series of 23 facial displays; these included single action units and combinations of action units. Subjects began and ended each display from a neutral face. Before performing each display, an experimenter described and modeled the desired display. Six of the displays were based on descriptions of prototypic basic emotions (i.e., joy, surprise, anger, fear, disgust)

Image sequences from neutral to target display were digitized into 640 by 480 or 490 pixel arrays with 8-bit precision for grayscale values. The final frame of each image sequence was coded using FACS (Facial Action Coding System) which describes subject's expression in terms of action units (AUs). There is a set of translation rules that may be used to re-label the AU coding into 'emotion prototypes' as defined in the Investigator's Guide to the FACS manual {0=neutral, 1=anger, 2=contempt, 3=disgust, 4=fear, 5=happy, 6=sadness, 7=surprise}

Here we have some examples from our database for different emotions

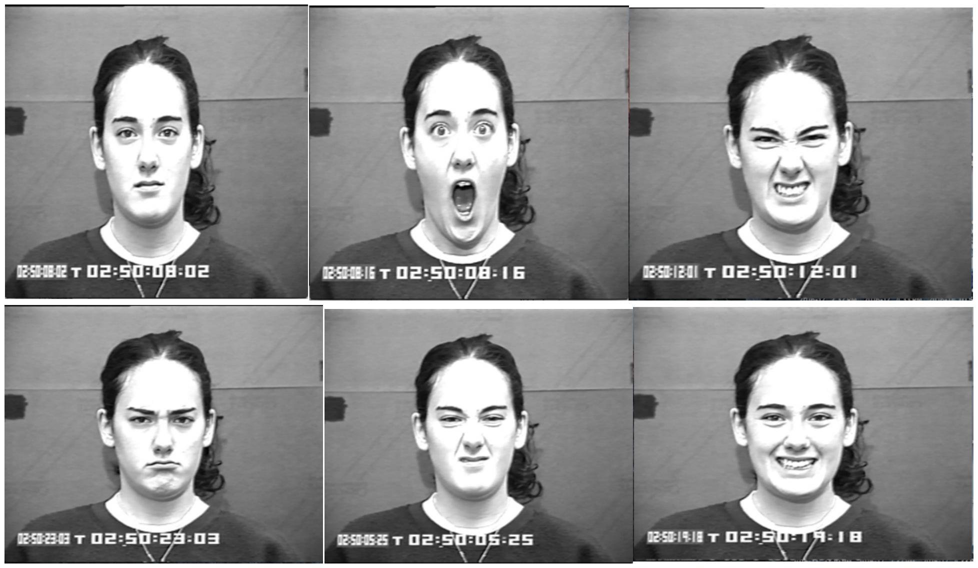
Fig 1. Cohn Kanade Dataset

However, there are some limitations to this approach. Firstly, all the images in the database were captured in a laboratory environment and the authors mention that only a subset of expressions contain archetypal emotions. Secondly, the dataset is very standardized. All faces are pointed at the camera. For real life applications, the lighting conditions, angles and head poses will have a lot of variation.

Background

Recognition of facial expressions from images entails finding solution to three distinct problems. First, the detection of the facial location from the image. Second, extracting relevant features from the facial area to effectively represent the face. The optimal features should minimize within-class variations of expressions while maximizing between class variations. The final step is to use the extracted features and employ any classification algorithm to identify correct facial expressions from images. There are several ways to do this which we have described in this report

Holistic Approach

In the holistic approach, we use the whole facial area to classify emotions. We used the Python OpenCV [2] (Open Source Computer Vision) module for face detection. OpenCV uses machine learning algorithms to search for faces in an image. To do this, OpenCV uses cascades. An OpenCV cascade breaks the problem of detecting faces into multiple stages. It first creates blocks of images. For each block of the image, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. In other words, rather than determining if the image does contain a face, we can more quickly determine if the image does not contain a face; because eliminations can be done quickly, while acceptance of faces will require more time.

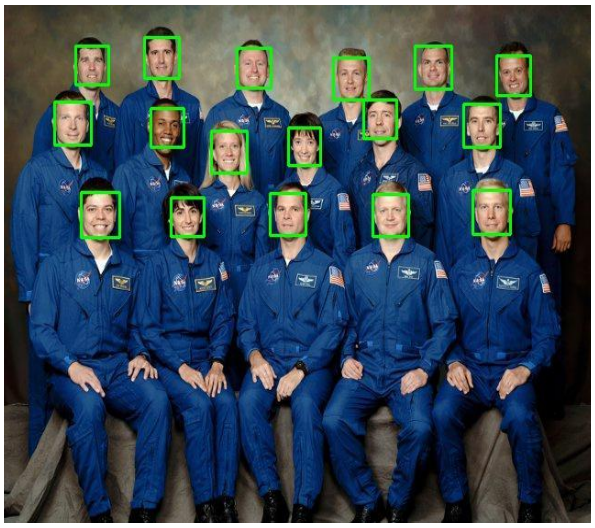
Fig 2. Face Detection with OpenCV

OpenCV comes with several built-in cascades for detecting everything from faces to eyes to hands and legs. We use the default cascade provided by OpenCV for face detection, haarcascade.

Preprocessing Methods

Cohn-Kanade's database comes with a pre-labeled folder for each source image. We initially have the images and their respective labels stored in separate folders (should create manually) as source_images and source_emotion. We then obtain the files and read the emotion and classify them accordingly into their 'emotion' folder in the dataset folder using Open CV2's HAAR filter to automate face finding, then resizing them from a 640*640 to a 350*350 image. We also converted all the images to grey scale.
On further investigating the dataset, we found that there is more than one neutral image for the same person. Hence, we have cleaned up the neutral dataset to include only one neutral image per person. From this cleaned dataset, we have divided the training and testing set by taking 80% and 20% respectively.

Dimensionality Reduction using Principle Component Analysis (PCA)

Principle Component Analysis has been widely used in facial recognition to find eigenfaces or a number of principle components less than or equal to the number of original variables that show the most variances. Thus, the first principle component will have the maximum possible variance. PCA thus has the effect of concentrating much of the signal into the first few principal components, while the later principal components may be dominated by noise, and so can be disposed off without great loss. These reduced set of principle components can then be fed into our classification algorithms.

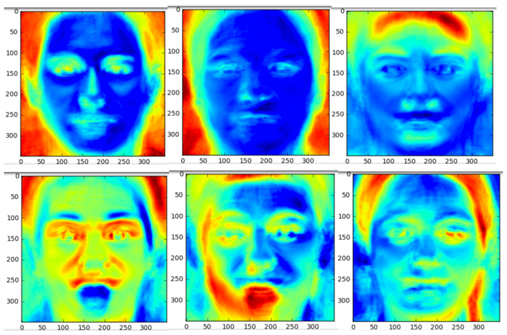
Fig 3. First 6 Principle Components

Dimensionality Reduction using Linear Discriminant Analysis (LDA)

A generalization of Fisher Discriminant Analysis, LDA allows us to find a linear combination of features that separates two or more classes. We want the data points from the same class to be as close as possible while maximizing the distance between the data points from different classes.

LDA works when the measurements made on independent variables for each observation are continuous quantities. The linear combination obtained is then used for dimensionality reduction before later classification by SVM or Neural Networks in our case.

Classification Methods

After performing PCA and LDA we go ahead and apply our classification techniques to train and predict the training and testing set respectively. In our case, we use SVM and Neural Networks.

NEURAL NETWORK

Neural Networks is an information processing paradigm that takes an inspiration from a human's biological system. It is a network largely connected by several nodes (neurons) arranged in layers that work together on complicated and imprecise data to solve problems such as pattern recognition (extract patterns) or data classification (detect trends) through training.

We use PCA or LDA for dimension reduction and then define a model using MLPClassifier by defining three hidden layers of 1000 nodes. We tested the accuracy of the model on the testing data and measured the precision, recall and f-1 scores.

SVM

SVMs are supervised learning models that are used to classification, where the classifier is formally defined by a separating hyperplane. An SVM Model is a representation of points mapped into their respective categories and divided by a gap that is as wide as possible.
We use the PCA or LDA for dimension reduction and then define a model using SVC and using a linear kernel as we observed that it gives the most accuracy. We tested the accuracy of the model on the testing data and measured the precision, recall and f-1 scores.

Analytical Approach

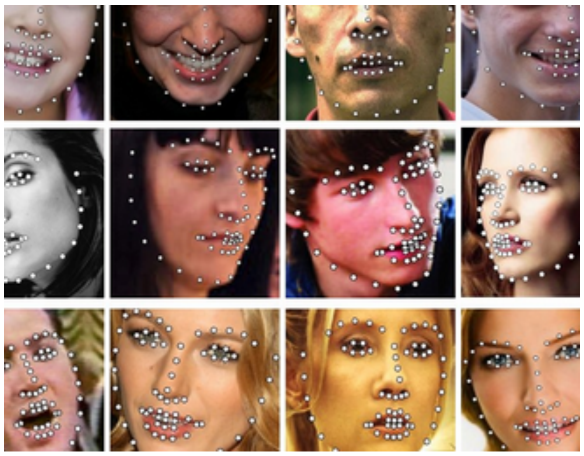

Fig 4. Detecting Facial Landmarks

Though not implemented here, it is worthwhile to mention the analytical approach as well. In this method, we use facial landmarks rather than using the entire face for classifying emotions. Facial landmarks are locations of different features like corners of the eyes, eyebrows, mouth, tip of the nose etc. You can use the dlib library from Python to extract facial landmarks. The next step after identifying facial landmarks is converting them to features to feed the classifier. To create meaningful features from the landmarks, extraction needs to be done correctly. One approach is to calculate the center of gravity of all facial landmarks which is nothing but the mean of both axes. One can then calculate the distance of all points relative to this central point. Since each line has both a magnitude (distance between both points) and a direction (angle relative to image where horizontal=0°), it is nothing but a vector. Another aspect that we need to consider is that faces maybe tilted. To correct for this, one can assume that the bridge of the nose is more or less straight for most people. Once can then offset all the angles calculated by the angle of the bridge of the nose. This rotates the entire vector array so that tilted faces look like non-tilted faces. Once the facial vectors are extracted in such a manner, they can be fed as inputs to any classification techniques like SVM or Random Forests.

Convolutional Neural Nets

We implemented convolutional neural networks, a category of neural networks that have proven effective in image recognition. There are several reasons that convolutional neural networks are becoming important. In traditional models for pattern recognition, feature extractors are hand designed. In CNNs, the weights of the convolutional layer being used for feature extraction as well as the fully connected layer being used for classification are determined during the training process. The improved network structures of CNNs lead to savings in memory requirements and computation complexity requirements.

Here, CNN was implemented in TensorFlow [4] the open source software library for machine intelligence.

Before we jump in into the workings of CNN, let us take a quick look at what and how TensorFlow works. TensorFlow is nothing but a programming system that represents computations as graphs and the nodes in the graph are called ops (operations). An op takes zero or more Tensors (a multi-dimension Array) and performs required manipulations and returns zero or more Tensors. For example, in our case we can represent a mini-batch of images as a 4D array of floating point numbers with dimensions [batch, height, width, channels].

To begin computation, a graph must be launched in a Session. A Session places the graph ops onto Devices, such as CPUs or GPUs, and provides methods to execute them. These methods return tensors as numpy ndarray objects in Python.

A brief description of the processing steps of CNN that we undertook follows.

There are four primary operations which are the building blocks of a CNN

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification

Convolution

CNNs derive their name from the “convolution” operator. The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

In CNN terminology, the 3×3 matrix is called a ‘filter’ or ‘kernel’ and the matrix formed by sliding the filter over the image and computing the dot product is called the ‘Feature Map’.

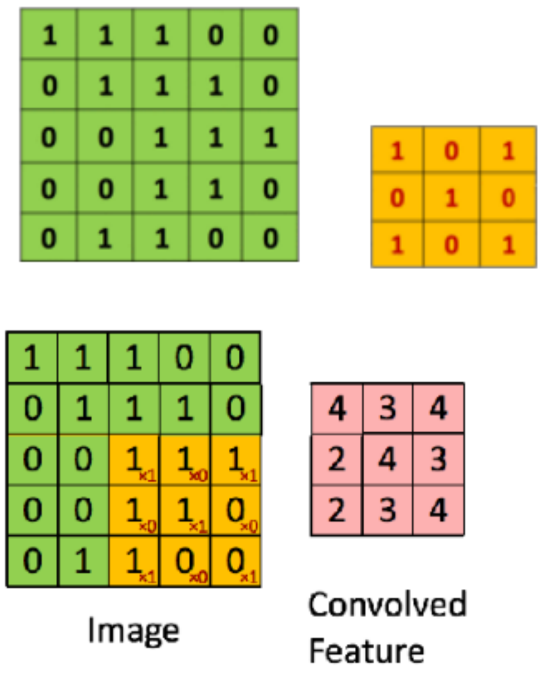

Fig 5. Convolution

Filters acts as feature detectors from the original input image. Different values of the filter matrix produce different Feature Maps for the same input image.

Non Linearity (ReLU)

A CNN learns the values of these filters during the training process. The more number of filters we have, the more image features get extracted and the better the network becomes at recognizing patterns in unseen images. We then introduce some nonlinearity in our CNN by performing an additional operation called Rectified Linear Unit. It is an element wise operation and replaces all negative pixel values in the feature map by 0. Since Convolution is a linear operation, we account for nonlinearity in real world data by introducing this function.

Pooling or Sub Sampling

We then perform downsampling to reduce the dimensionality of the feature map while retaining most of the important information. There can be different types of spatial pooling methods (sum, max, min, etc.) but we use max pooling since it has been shown to work better. In case of max pooling, we define a spatial neighborhood, a 2×2 window for our data and take the largest element from the rectified feature map within that window. We then slide this window and take the maximum value in the window region each time. This reduces the dimensionality of the feature map. Pooling also helps us to arrive at a scale invariant representation of an image. This is important because we can detect objects in an image no matter where they are located.

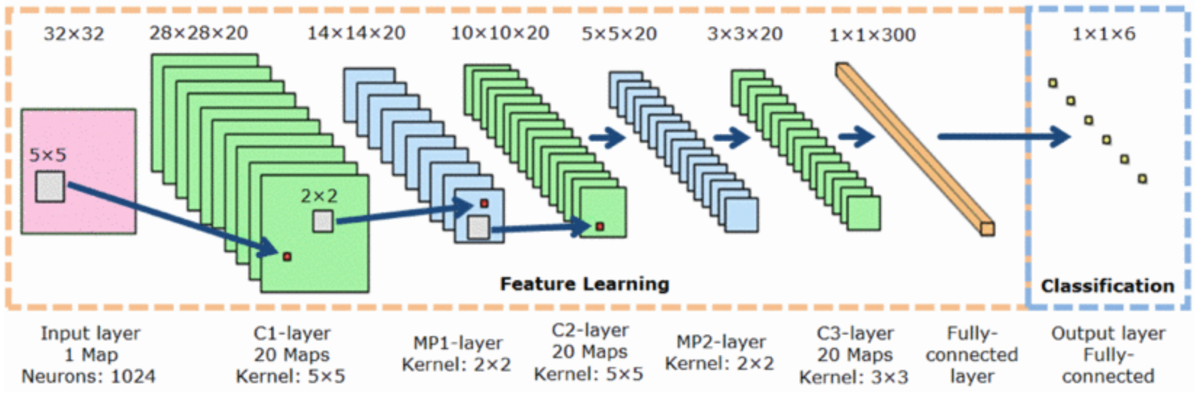

Fig 6. A Convolutional Neural Network

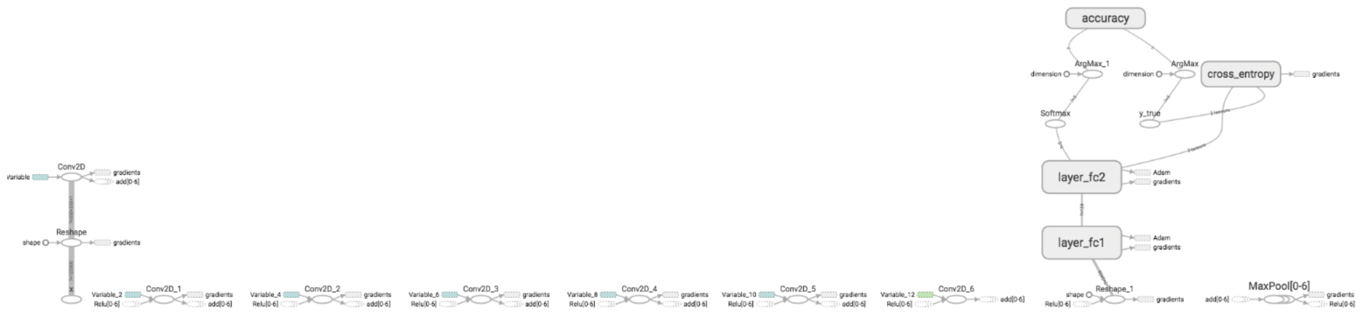

Fig 7. The TensorFlow Convolutional Neural Network

Fully Connected Layer

Fully connected layers are often used as the final layers of a CNN. These layers mathematically sum a weighting of the previous layer of features, indicating the precise mix of “ingredients” to determine a specific target output result. All the elements of all the features of the previous layer get used in the calculation of each element of each output feature

CNN TensorFlow Methodology

We use six convolution and pooling layers which act as feature extractors from the image with 16, 36, 48, 64, 80, 96 number of filters at each layer respectively. The filter size is 5x5 in each layer. The output of the last layer is then fed to a fully connected layer of 128 neurons.
Output of this FC layer is fed to the output layer of the NN with 8 neurons, one for each class.
The convolutional filters are initially chosen at random, so the classification is done randomly. The error between the predicted and true class of the input image is back propagated through the Convolutional Network using the chain-rule of differentiation and the filter-weights are updated to improve the classification error. This is done iteratively hundreds or thousands of times until the classification error is sufficiently low.

Additional Steps for Improving Accuracy

The dataset is imbalanced as there are very few examples for content, fear and sadness. Since the classifier does not have enough data points, it will not generalize well for these classes and hence these classes have been excluded and here onwards we only classify five emotions and consequently use 5 neurons in our output layer. Note that our dataset now includes training data (295×350×350) and testing data (72×350×350).

Evaluation

We ran the CNN algorithm for 600 iterations. The graphs here indicate that as the number of iterations increase, the prediction accuracy improves while the cross entropy reduces. The accuracy and cross entropy both converge after about 300 iterations.

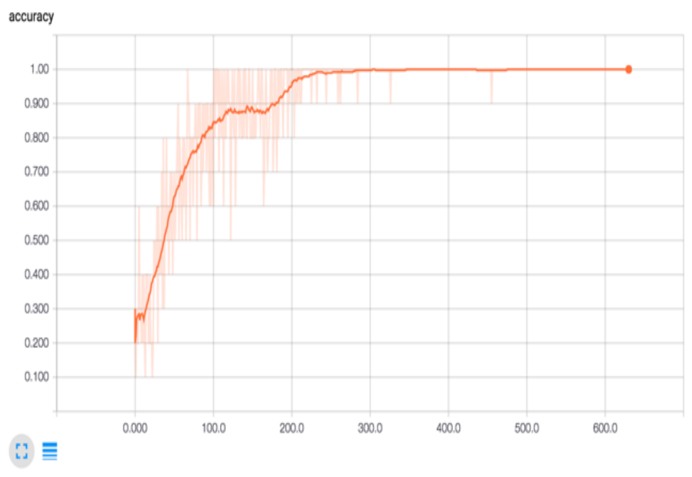

Fig 8. Prediction Accuracy

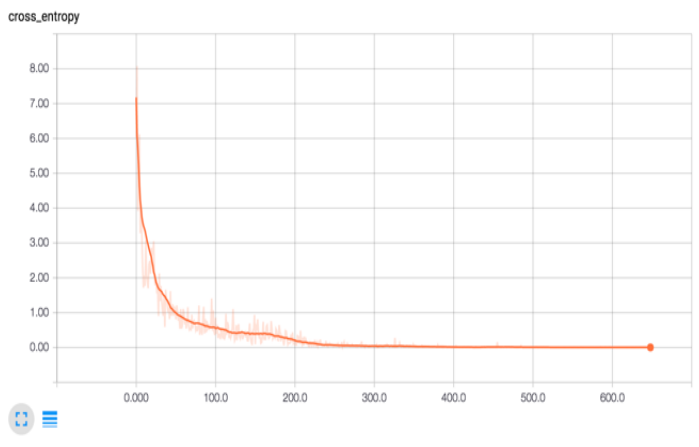

Fig 9. Cross Entropy

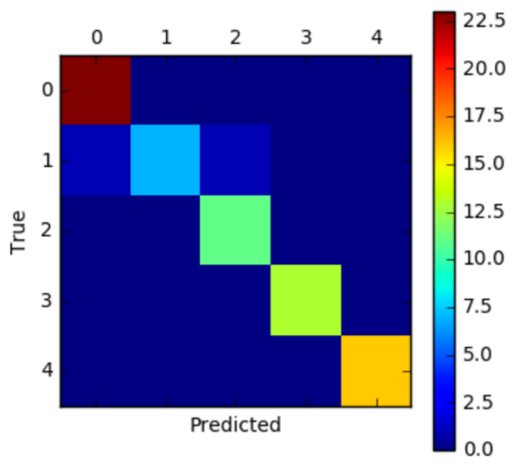

Fig 10. Confusion Matrix for 5 classes

PCA with Neural Networks MLP

precision	recall	f1-score	support
0.53	0.91	0.67	23
0.60	0.33	0.43	9
0.00	0.00	0.00	3
0.88	0.64	0.74	11
0.00	0.00	0.00	5
0.87	1.00	0.93	13
0.00	0.00	0.00	5
0.88	0.94	0.91	16
0.62	**0.69**	**0.63**	**85**

(avg/total)

```
Confusion Matrix
[[21  0  0  1  0  0  0  1]
 [ 6  3  0  0  0  0  0  0]
 [ 3  0  0  0  0  0  0  0]
 [ 3  1  0  7  0  0  0  0]
 [ 2  0  0  0  0  2  0  1]
 [ 0  0  0  0  0 13  0  0]
 [ 4  1  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0 15]]

Prediction Score
0.694117647059
```

Following are examples of images that have been misclassified.

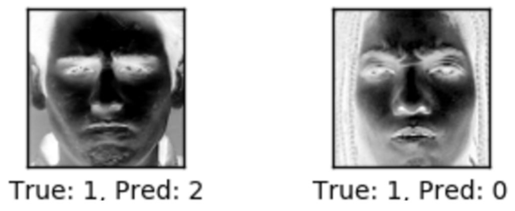

Fig 11. Misclassified Images

As we can see, these emotions are quite hard to interpret accurately even for a human eye.

SVM with PCA

precision	recall	f1-score	support
0.70	0.91	0.79	23
0.71	0.56	0.63	9
0.43	1.00	0.60	3
1.00	0.82	0.90	11
0.75	0.60	0.67	5
0.92	0.85	0.88	13
1.00	0.20	0.33	5
1.00	0.94	0.97	16
0.84	**0.80**	**0.79**	**85**

(avg / total)

```
Confusion Matrix
[[21  0  2  0  0  0  0  0]
 [ 4  5  0  0  0  0  0  0]
 [ 0  0  3  0  0  0  0  0]
 [ 1  0  1  9  0  0  0  0]
 [ 1  0  0  0  3  1  0  0]
 [ 1  0  1  0  0 11  0  0]
 [ 1  2  0  0  1  0  1  0]
 [ 1  0  0  0  0  0  0 15]]

Prediction Score
0.8
```

Neural_Network MLP with LDA

```
precision recall f1-score   support

  0.63      0.83      0.72       23
  0.57      0.44      0.50        9
  0.00      0.00      0.00        3
  1.00      0.82      0.90       11
  0.67      0.40      0.50        5
  0.87      1.00      0.93       13
  0.50      0.40      0.44        5
  0.94      0.94      0.94       16

 0.74      0.75      0.74       85
(avg / total)
```

Confusion Matrix

```
[[19  1  1  0  1  0  1  0]
 [ 4  4  0  0  0  0  1  0]
 [ 3  0  0  0  0  0  0  0]
 [ 1  0  0  9  0  1  0  0]
 [ 1  0  0  0  2  1  0  1]
 [ 0  0  0  0  0 13  0  0]
 [ 1  2  0  0  0  0  2  0]
 [ 1  0  0  0  0  0  0 15]]
```

Prediction Score
0.752941176471

SVM with LDA

```
precision   recall f1-score   support

    0.61      0.83      0.70       23
    0.24      0.67      0.35        9
    0.00      0.00      0.00        3
    0.86      0.55      0.67       11
    0.50      0.40      0.44        5
    0.91      0.77      0.83       13
    0.50      0.20      0.29        5
    0.67      0.12      0.21       16

    0.63      0.54      0.52       85
(avg / total)
```

Confusion Matrix

```
[[19  2  1  0  0  0  1  0]
 [ 3  6  0  0  0  0  0  0]
 [ 3  0  0  0  0  0  0  0]
 [ 3  2  0  6  0  0  0  0]
 [ 1  0  0  0  2  1  0  1]
 [ 1  0  0  0  2 10  0  0]
 [ 1  2  0  1  0  0  1  0]
 [ 0 13  1  0  0  0  0  2]]
```

Prediction Score
0.541176470588

SVM with PCA

```
precision recall f1-score   support

  0.79      0.96      0.86       23
  0.83      0.56      0.67        9
  1.00      0.82      0.90       11
  1.00      1.00      1.00       13
  0.94      0.94      0.94       16

 0.90      0.89      0.89       72
(avg/total)
```

Confusion Matrix

```
[[22  1  0  0  0]
 [ 3  5  0  0  1]
 [ 2  0  9  0  0]
 [ 0  0  0 13  0]
 [ 1  0  0  0 15]]
```

Prediction Score
0.888888888889

Neural Networks with PCA

```
precision recall f1-score   support
 0.70     1.00     0.82       23
 0.80     0.44     0.57        9
 1.00     0.55     0.71       11
 0.92     0.92     0.92       13
 1.00     0.94     0.97       16
  0.86     0.83     0.82       72
(avg / total)

Confusion Matrix
[[23  0  0  0  0]
 [ 5  4  0  0  0]
 [ 3  1  6  1  0]
 [ 1  0  0 12  0]
 [ 1  0  0  0 15]]

Prediction Score (depicts F1-score)
0.833333333333
```

SVM with LDA

```
precision    recall f1-score  support
   0.76     0.96     0.85       23
   0.83     0.56     0.67        9
   1.00     0.82     0.90       11
   0.93     1.00     0.96       13
   1.00     0.88     0.93       16
 0.89     0.88     0.87       72
(avg / total)

Confusion Matrix
[[22  1  0  0  0]
 [ 4  5  0  0  0]
 [ 2  0  9  0  0]
 [ 0  0  0 13  0]
 [ 1  0  0  1 14]]

Prediction Score
0.875
```

Conclusion

In our paper, we begin with training the dataset with all the 7 emotions available to us, starting with Neural Networks and then SVM (with LDA and PCA) and we observed that mostly SVM performs better. Then, CNN is brought into picture so that so that the feature extraction is even more concentrated and sharp, after experimenting on the no of layers, learning rates and other parameters, this ends up giving us the most accurate prediction with the accuracy rate of 97.2% which is among the best seen. Hence, we can conclude that the CNN proves to be the most optimal solution when it comes to facial emotion recognition.

References

[1]Kanade, Cohn, Tian *Comprehensive Database for Facial Expression Analysis, 2000* http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/

[2]http://opencv.org/https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

[3] *Paul Vangent, Emotion Recognition using Facial Landmarks, Python, DLib and OpenCV* http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/

[4] https://www.tensorflow.org/

[5] https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

[6]http://www.scholarpedia.org/article/Fisherfaces

[7] Duncan, Shine, English *Facial Emotion Recognition in Real Time*

[8] Sun *Neural Networks for Emotion Classification*

[9] http://www.pitt.edu/~jeffcohn/CVPR2010_CK+2.pdf

[10] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[11] http://www.cs.utah.edu/~widanaga/papers/Widanagamaachchi.2009.thesis.pdf

[12] https://github.com/sbondada/emotion-detection

[13] https://arxiv.org/pdf/1105.6014.pdf