# Table Of Contents

# Index Of Images

# Index Of Tables

# 1. INTRODUCTION :

**“2018 is the year when machines learn to grasp human emotions” --Andrew Moore, the dean of computer science at Carnegie Mellon.**

With the advent of modern technology our desires went high and it binds no bounds. In the present era a huge research work is going on in the field of digital image and image processing. The way of progression has been exponential and it is ever increasing. Image Processing is a vast area of research in present day world and its applications are very widespread.

Image processing is the field of signal processing where both the input and output signals are images. One of the most important application of Image processing is Facial expression recognition. Our emotion is revealed by the expressions in our face. Facial Expressions plays an important role in interpersonal communication. Facial expression is a non verbal scientific gesture which gets expressed in our face as per our emotions. Automatic recognition of facial expression plays an important role in artificial intelligence and robotics and thus it is a need of the generation. Some application related to this include Personal identification and Access control, Videophone and Teleconferencing, Forensic application, Human-Computer Interaction, Automated Surveillance, Cosmetology and so on.

The objective of this project is to develop Automatic Facial Expression Recognition System which can take human facial images containing some expression as input and recognize and classify it into seven different expression class such as :

I. **Neutral**
II. **Angry**
III. **Disgust**
IV. **Fear**
V. **Happy**
VI. **Sadness**
VII. **Surprise**


**1.**

Several Projects have already been done in this fields and our goal will not only be to develop an Automatic Facial Expression Recognition System but also improving the accuracy of this system compared to the other available systems.

# 2. MOTIVATION :

Significant debate has risen in past regarding the emotions portrayed in the world famous masterpiece of Mona Lisa. British Weekly 'New Scientist' has stated that she is in fact a blend of many different emotions, 83%happy, 9% disgusted, 6% fearful, 2% angry.

**2.1.**

We have also been motivated observing the benefits of physically handicapped people like deaf and dumb. But if any normal human being or an automated system can understand their needs by observing their facial expression then it becomes a lot easier for them to make the fellow human or automated system understand their needs.

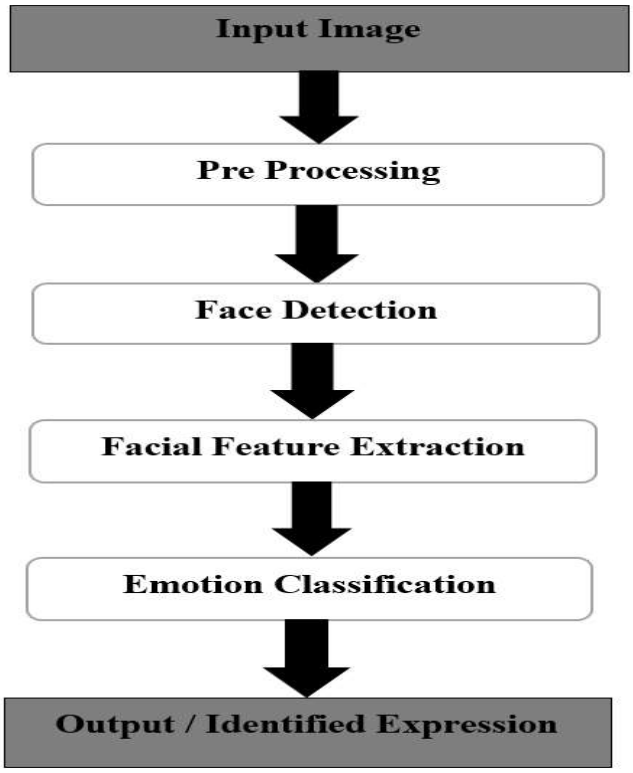
**2.2.**

# 3.PROBLEM DEFINITION :

Human facial expressions can be easily classified into 7 basic emotions: happy, sad, surprise, fear, anger, disgust, and neutral. Our facial emotions are expressed through activation of specific sets of facial muscles. These sometimes subtle, yet complex, signals in an expression often contain an abundant amount of information about our state of mind. Through facial emotion recognition, we are able to measure the effects that content and services have on the audience/users through an easy and low-cost procedure. For example, retailers may use these metrics to evaluate customer interest. Healthcare providers can provide better service by using additional information about patients' emotional state during treatment. Entertainment producers can monitor audience engagement in events to consistently create desired content.

Humans are well-trained in reading the emotions of others, in fact, at just 14 months old, babies can already tell the difference between happy and sad. **But can computers do a better job than us in accessing emotional states?** To answer the question, We designed a deep learning neural network that gives machines the ability to make inferences about our emotional states. In other words, we give them eyes to see what we can see.

Problem formulation of our project:




**3.**

Facial expression recognition is a process performed by humans or computers, which consists of:

1. Locating faces in the scene (e.g., in an image; this step is also referred to as facedetection),
2. Extracting facial features from the detected face region (e.g., detecting the shape of facialcomponents or describing the texture of the skin in a facial area; this step is referred to asfacial feature extraction),
3. Analyzing the motion of facial features and/or the changes in the appearance of facialfeatures and classifying this information into some facial-expression-interpretativecategories such as facial muscle activations like smile or frown, emotion (affect)categories like happiness or anger, attitude categories like (dis)liking or ambivalence, etc.(this step is also referred to as facial expression interpretation).

   Several Projects have already been done in this fields and our goal will not only be to develop a Automatic Facial Expression Recognition System but also improving the accuracy of this system compared to the other available systems.

# 4. LITERATURE STUDY :

As per various literature surveys it is found that for implementing this project four basic steps are required to be performed.

i. Preprocessing
ii. Face registration
iii. Facial feature extraction
iv. Emotion classification

Description about all these processes are given below-

## ❖ Preprocessing :

Preprocessing is a common name for operations with images at the lowest level of abstraction both input and output are intensity images. Most preprocessing steps that are implemented are –

a. Reduce the noise

b. Convert The Image To Binary/Grayscale.

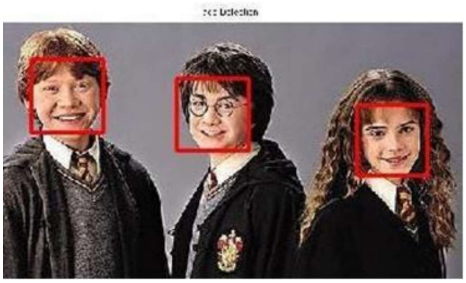
c. Pixel Brightness Transformation.

d. Geometric Transformation.


**4.1.**

## ❖ Face Registration :

Face Registration is a computer technology being used in a variety of applications that identifies human faces in digital images. In this face registration step, faces are first located in the image using some set of landmark points called “face localization” or “face detection”. These detected faces are then geometrically normalized to match some template image in a process called “faceregistration”.

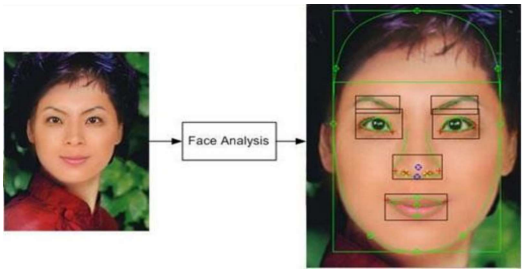


**4.2.**

## ❖ Facial Feature Extraction :

Facial Features extraction is an important step in face recognition and is defined as the process of locating specific regions, points, landmarks, or curves/contours in a given 2-D image or a 3D range image. In this feature extraction step, a numerical feature vector is generated from the resulting registered image. Common features that can be extracted are-

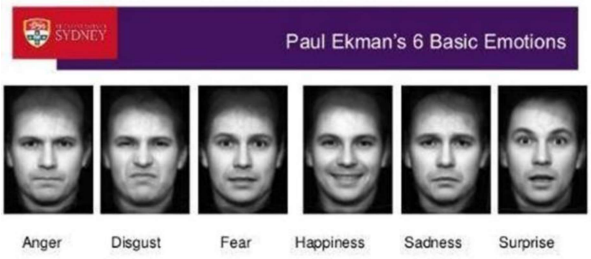
a. Lips
b. Eyes
c. Eyebrows
d. Nose tip




**4.3.**

## ❖ Emotion Classification :

In the third step, of classification, the algorithm attempts to classify the given faces portraying one of the seven basic emotions.




**4.4.**

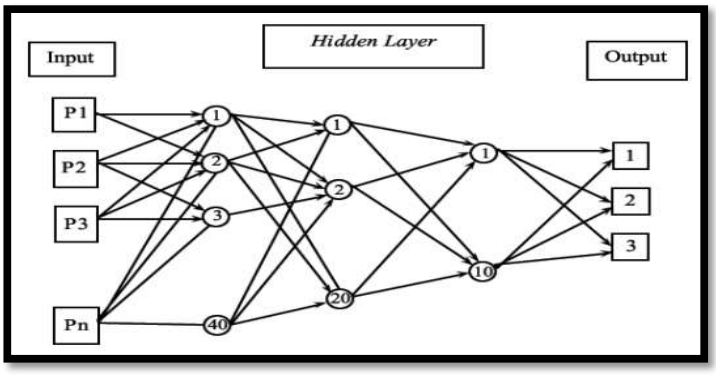
Paul Ekman (born February 15, 1934) is an American psychologist and professor emeritus at the University of California, San Francisco who is a pioneer in the study of emotions and their relation to facial expressions. He has created an "atlas of emotions" with more than ten thousand facial expression.

Different approaches which are followed for Facial Expression Recognition:

- ## Neural Network Approach :

The neural network contained a hidden layer with neurons. The approach is based on the assumption that a neutral face image corresponding to each image is available to the system. Each neural network is trained independently with the use of on-line back propagation.

Neural Network will be discussed later.



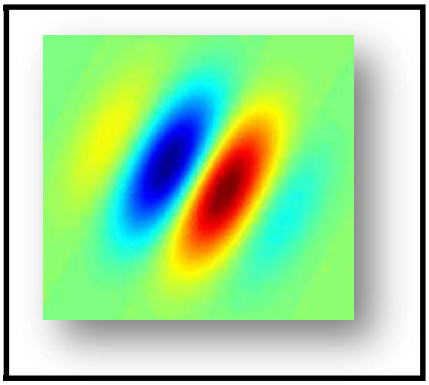

**4.5.**

- ## Principal of Component Analysis :

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variable called Principal Components.

- ## Gabor Filter :

In image processing, a **Gabor filter**, named after Dennis Gabor, is a linear filter used for texture analysis, which means that it basically analyses whether there are any specific frequency content in the image in specific directions in a localized region around the point or region of analysis. Frequency and orientation representations of Gabor filters are claimed by many contemporary vision scientists to be similar to those of the human visual systemm, though there is no empirical evidence and no functional rationale to support the idea. They have been found to be particularly appropriate for texture representation and discrimination. In the spatial domain, a 2D Gabor filter is a Gaussiann kernel functionn modulated by a sinusoidal plane wave.



**4.6.**

Gabor filters are directly related to Gabor wavelets, since they can be designed for a number of dilations and rotations. However, in general, expansion is not applied for Gabor wavelets, since this requires computation of bi-orthogonal wavelets, which may be very time-consuming. Therefore, usually, a filter bank consisting of Gabor filters with various scales and rotations is created. The filters are convolved with the signal, resulting in a so-called Gabor space. This process is closely related to processes in the primary visual cortex. Jones and Palmer showed that the real part of the complex Gabor function is a good fit to the receptive field weight functions found in simple cells in a cat's striate cortex

- **Support Vector Machine** :

  In machine learning, **support vector machines** (**SVMs**, also **support vector networks**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary model (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

  In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick implicitly mapping their inputs into high-dimensional feature spaces.

  When data are not labeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to theseformed groups. The **support vector clustering**algorithm created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

- **Training & Testing Database:**

  In machine learning, the study and construction of algorithms that can learn from and make predictions on data is a common task. Such algorithms work by making data-driven predictions or decisions, through building a mathematical model from input data.

  The data used to build the final model usually comes from multiple datasets. In particular, three data sets are commonly used in different stages of the creation of the model.

  The model is initially fit on a **training dataset**, that is a set of examples used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model. The model (e.g. a neural net or a naive Bayes classifier) is trained on the

training dataset using a supervised learning method (e.g. gradient descent or stochastic gradient descent). In practice, the training dataset often consist of pairs of an input vector and the corresponding *answer* vector or scalar, which is commonly denoted as the *target*. The current model is run with the training dataset and produces a result, which is then compared with the *target*, for each input vector in the training dataset. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the **validation dataset**. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters (e.g. the number of hidden units in a neural network). Validation datasets can be used for regularization by early stopping: stop training when the error on the validation dataset increases, as this is a sign of overfitting to the training dataset. This simple procedure is complicated in practice by the fact that the validation dataset's error may fluctuate during training, producing multiple local minima. This complication has led to the creation of many ad-hoc rules for deciding when overfitting has truly begun.

Finally, the **test dataset** is a dataset used to provide an unbiased evaluation of a *final* model fit on the training dataset.

- **Various facial datasets available online are:**

1. Japanese Female Facial Expression (JAFFE)
2. FER
3. CMU MultiPIE
4. Lifespan
5. MMI
6. FEED
7. CK

- **Accuracy of various databases:**

| *Traing* | *Testing* | *Accu* |
|---|---|---|
| FER2013 | CK+ | 76.05 |
| FER2013 | CK+ | 73.38 |
| JAFFE | CK+ | 54.05 |
| MMI | CK+ | 66.20 |
| FEED | CK+ | 56.60 |
| FER2013 | JAFFE | 50.70 |
| FER2013 | JAFFE | 45.07 |
| CK+ | JAFFE | 55.87 |
| BU-3DFE | JAFFE | 41.96 |
| CK | JAFFE | 45.71 |
| CK | JAFEE | 41.30 |
| FEED | JAFFE | 46.48 |
| FEED | JAFFE | 60.09 |

**1.**

- **Accuracy of various approaches are stated as follows:**

| **Sr. No** | **Method/ Technique(s)/ (Database)** | **Result/Accuracy** | **Conclusion** | **Future work** |
|---|---|---|---|---|
| 1 | Neural Network + Rough Contour Estimation Routine (RCER) [15] (Own Database) | 92.1% recognition Rate | In this paper, they describe radial basis function network (RBFN) and a multilayer perception (MLP) network. | |
| 2 | Principal Component Analysis [18] (FACE94) | 35% less computation time and 100% Recognition | Useful where larger database and less computational time | They want to repeat their experiment on larger and different databases. |
| | | 83% Surprise in CK, | Compared with the facial | Future work is to develop |

| | | | | |
|---|---|---|---|---|
| 3 | PCA + Eigenfaces [19]<br>(CK, JAFFE) | 83% Happiness in<br>JAFFE,<br>Fear was the most<br>Confused<br>Expression | expression<br>recognition method based on the video<br>sequence, the one based on the static<br>image is more difficult due to the lack of<br>temporal information. | a facial expression<br>recognition system,<br>which combines body<br>gestures of the user with<br>user facial expressions. |
| 4 | 2D Gabor filter [22]<br>(Random Images) | 12 Gabor Filter<br>bank used to locate<br>Edge | Multichannel Gabor filtration scheme<br>used for the detection of salient points<br>and the extraction of texture features for<br>image retrieval applications. | They work on adding<br>global and local colour<br>histograms and<br>parameters connected<br>with the shapes of objects<br>within images. |
| 5 | Local Gabor Filter +<br>PCA + LDA [23]<br>(JAFFE) | Obtained 97.33%<br>recognition rate<br>with the help of<br>PCA+LDA<br>Features | They conclude that PCA+LDA features<br>partially eliminate sensitivity of-<br>illumination. | |
| 6 | PCA + AAM [24]<br>(Image sequences<br>from FG-NET<br>consortium) | The performance<br>ratios are 100 % for<br>Expression<br>recognition<br>from<br>extracted faces, | The computational time and complexity<br>was also very small. Improve the<br>Efficiency | Extend the work to<br>identify the face and it‟s<br>expressions from 3D<br>images. |

**2.**

# 5. SOFTWARE REQUIREMENT :

As the project is developed in python, we have used Anaconda for Python 3.6.5 and Spyder.

- **Anaconda**

  Itis a free and open source distribution of the Python and R programming languages for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda*. The Anaconda distribution is used by over 6 million users, and it includes more than 250 popular data science packages suitable for Windows, Linux, and MacOS.

- **Spyder**

  Spyder (formerly Pydee) is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates NumPy, SciPy, Matplotlib and IPython, as well as other open source software. It is released under the MIT license.

  Spyder is extensible with plugins, includes support for interactive tools for data inspection and embeds Python-specific code quality assurance and introspection instruments, such as Pyflakes, Pylint and Rope. It is available cross-platform through Anaconda, on Windows with WinPython and Python (x,y), on macOS through MacPorts, and on major Linux distributions such as Arch Linux, Debian, Fedora, Gentoo Linux, openSUSE and Ubuntu.

Features include:

- editor with syntax highlighting and introspection for code completion
- support for multiple Python consoles (including IPython)
- the ability to explore and edit variables from a GUI

Available plugins include:

- Static Code Analysis with Pylint
- Code Profiling
- Conda Package Manager with Conda

### ❖ Hardware Interfaces

1. **Processor :** Intel CORE i5 processor with minimum 2.9 GHz speed.
2. **RAM :** Minimum 4 GB.
3. **Hard Disk :** Minimum 500 GB

### ❖ Software Interfaces

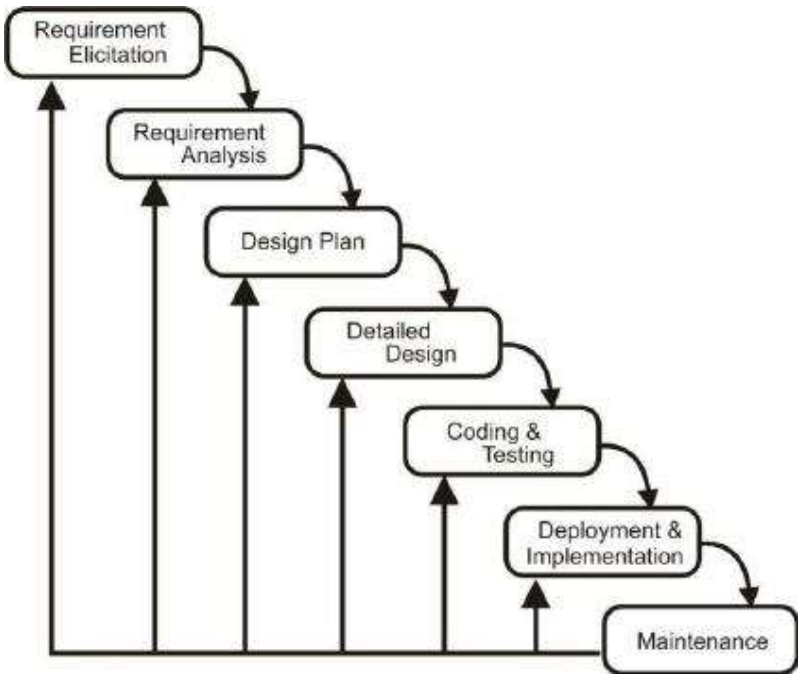
1. Microsoft Word 2003
2. **Database Storage :** Microsoft Excel
3. **Operating System :** Windows10

# 6. PLANNING :

The steps we followed while developing this project are-:

1. Analysis of the problem statement.
2. Gathering of  the requirement specification
3. Analysation of the feasibility of the project.
4. Development of a general layout.
5. Going by the journals regarding the previous related works on this field.
6. Choosing the method for developing the algorithm.
7. Analyzing the various pros and cons.
8. Starting the development of the project
9. Installation of software like ANACONDA.
10. Developing an algorithm.
11. Analysation of algorithm by guide.
12. Coding as per the developed algorithm in PYTHON.

We developed this project as per the iterative waterfall model:



# 7. DESIGN :

## DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

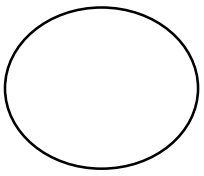
Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to Systems Design.

### Symbols and Notations Used in DFDs

Using any convention's DFD rules or guidelines, the symbols depict the four components of data flow diagrams -

**External entity**: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

**Process**: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.



**Data store**: files or repositories that hold information for later use, such as a database table or a membership form.

**Data flow**: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”



**DFD levels and layers**

A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish.

**DFD Level 0** is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

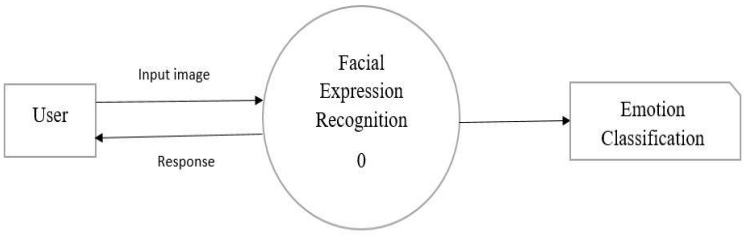
**DFD Level 1** provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

**DFD Level 2** then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system’s functioning.

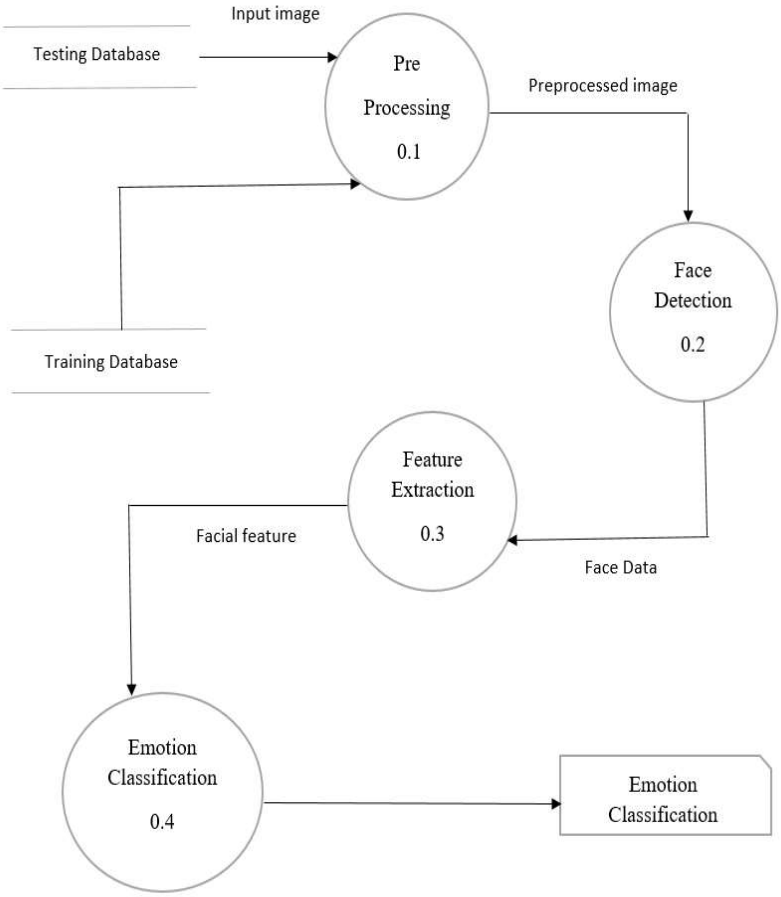
Progression to Levels 3, 4 and beyond is possible, but going beyond Level 3 is uncommon. Doing so can create complexity that makes it difficult to communicate, compare or model effectively.

Using DFD layers, the cascading levels can be nested directly in the diagram, providing a cleaner look with easy access to the deeper dive.

## Level 0

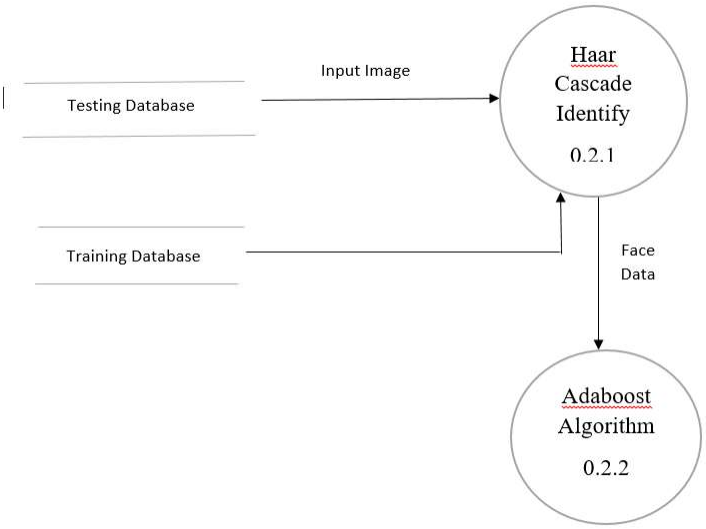


## Level 1

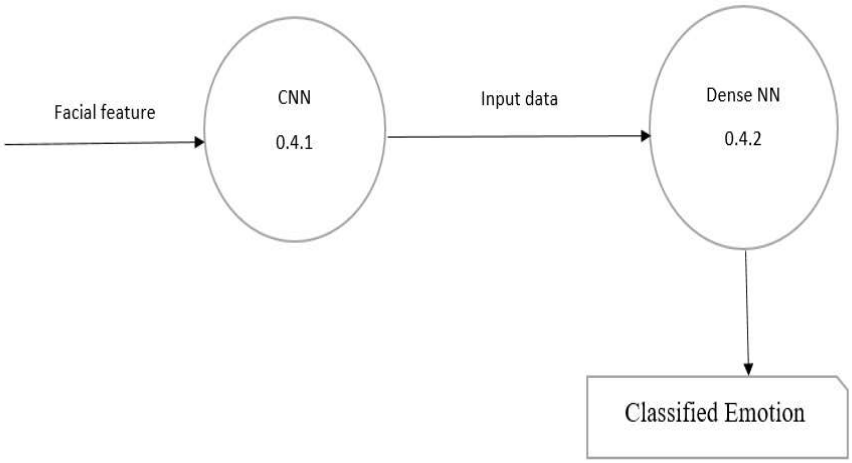



## Level 2

Face Detection-



Emotion Classification-



# 8.ALGORITHM :

**Step 1 :**Collection of a data set of images. (In this case we are using FER2013 database of **35887 pre-cropped, 48-by-48-pixel grayscale images** of faces each labeled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral.

**Step 2 :**Pre-processing of images.

**Step 3 :**Detection of a face from each image.

**Step 4 :**The cropped face is converted into grayscale images.

**Step 5 :** The pipeline ensures every image can be fed into the input layer as a (1, 48, 48) numpy array.

**Step 5 :**The numpy array gets passed into the Convolution2D layer.

**Step 6 :Convolution** generates feature maps.

**Step 7 :**Pooling method called MaxPooling2D that uses (2, 2) windows across the feature map only keeping the maximum pixel value.

**Step 8 :**During training, Neural network Forward propagation and Backward propagation performed on the pixel values.

**Step 9 :**The Softmax function presents itself as a probability for each emotion class.

The model is able to show the detail probability composition of the emotions in the face.

# 9. IMPLEMENTATION DETAILS:

- ## The Database :

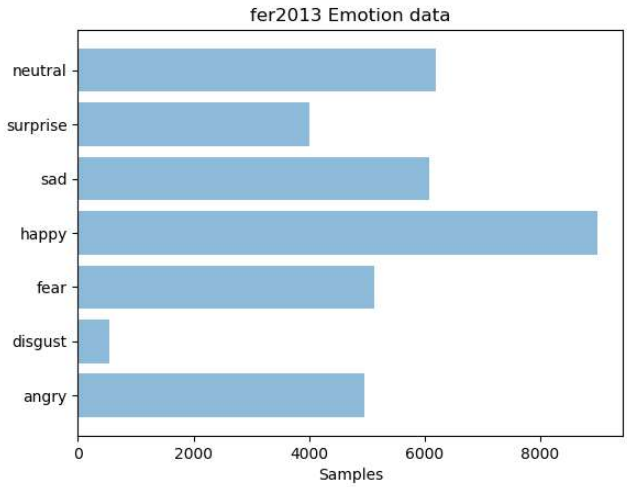
The dataset, used for training the model is from a Kaggle Facial Expression Recognition Challenge a few years back (FER2013). The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=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.

**Emotion labels in the dataset:**

**0:** -4593 images- ***Angry***
**1:** -547 images- ***Disgust***
**2:** -5121 images- ***Fear***
**3:** -8989 images- ***Happy***
**4:** -6077 images- ***Sad***
**5:** -4002 images- ***Surprise***
**6:** -6198 images- ***Neutral***

**5.1.**




**5.2.**

- **The Library & Packages :**
  - **OpenCV :**

    OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.
    It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

    **OpenCV's application areas include :**

    - 2D and 3D feature toolkits
    - Egomotion estimation
    - Facial recognition system
    - Gesture recognition
    - Human–computer interaction (HCI)
    - Mobile robotics
    - Motion understanding
    - Object identification
    - Segmentation and recognition
    - Stereopsis stereo vision: depth perception from 2 cameras
    - Structure from motion (SFM)
    - Motion tracking
    - Augmented reality

    To support some of the above areas, OpenCV includes a statistical machine learning library that contains :

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

- **Numpy :**

NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier Transform, and random number capabilities.

- **Numpy Array :**

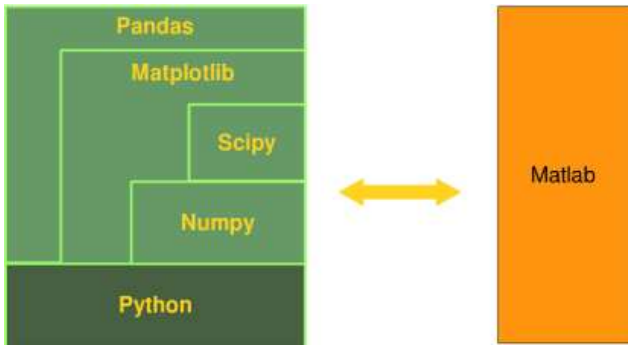
A **numpy array** is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the **array**; the shape of an **array** is a tuple of integers giving the size of the **array** along each dimension.

- **SciPy :**

SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others.
NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is build on the code of Numeric and the features of Numarray.

- **The Python Alternative To Matlab :**

  Python in combination with Numpy, Scipy and Matplotlib can be used as a replacement for MATLAB. The combination of NumPy, SciPy and Matplotlib is a free (meaning both "free" as in "free beer" and "free" as in "freedom") alternative to MATLAB. Even though MATLAB has a huge number of additional toolboxes available, NumPy has the advantage that Python is a more modern and complete programming language and - as we have said already before - is open source. SciPy adds even more MATLAB-like functionalities to Python. Python is rounded out in the direction of MATLAB with the module Matplotlib, which provides MATLAB-like plotting functionality.




**5.3.**

- **Keras :**

  Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

  Keras contains numerous implementations of commonly used neural network building blocks suchas layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

  Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.

  It also allows use of distributed training of deep learning models on clusters of Graphics Processing Units (GPU).

- **TensorFlow :**

  TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top ofTensorFlow.

- **Guiding Principles :**

  - **User Friendliness :** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it

minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

- **Modularity :** A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Easy Extensibility :** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python :** No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

## ○ SYS :

System-specific parameters and functions. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. The sys module provides information about constants, functions and methods of the Python interpreter. dir(system) gives a summary of the available constants, functions and methods. Another possibility is the help() function. Using help(sys) provides valuable detail information.

## ○ Sigmoid Function :

The sigmoid function in a neural network will generate the end point (activation) of inputs multiplied by their weights. For example, let's say we had two columns (features) of input data and one hidden node (neuron) in our neural network. Each feature would be multiplied by its corresponding weight value and then added together and passed through the sigmoid (just like a logistic regression). To take that simple example and turn it into a neural network we just add more hidden units. In addition to adding more hidden units, we add a path from every input feature to each of those hidden units where it is multiplied by its corresponding weight. Each hidden unit takes the sum of its inputs weights and passes that through the sigmoid resulting in that unit's activation.

### ❖ Properties of Sigmoid Function :

- The sigmoid function returns a real-valued output.
- sigmoid function take any range real number and returns the output value which falls in the range of **0 to 1.**

  The first derivative of the sigmoid function will be non-negative or non-positive.
- **Non-Negative:** If a number is greater than or equal to zero.
- **Non-Positive:** If a number is less than or equal to Zero.

## ○ Softmax Function :

- Softmax function calculates the probabilities distribution of the event over 'n' different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the

calculated probabilities will be helpful for determining the target class for the given inputs.

- The main advantage of using Softmax is the output probabilities range. The range will **0 to 1**, and the sum of all the probabilities will be **equal to one**. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.
- The formula computes the **exponential (e-power)** of the given input value and the **sum of exponential values** of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.

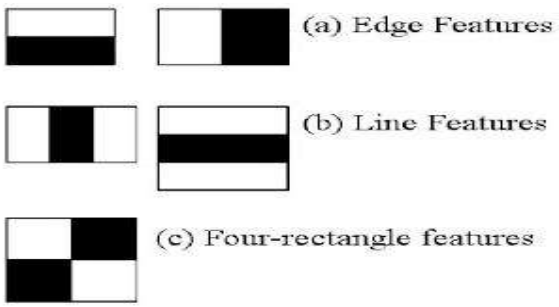
❖ **Properties Of Softmax Function :**

- The calculated probabilities will be in the range of 0 to 1.
- The sum of all probabilities is equal to 0.
-

# • Face Registration :

## ○ Haar Features :

Haar feature is similar to Karnals, which is generally used to detect edge. All human faces share some similar features, like eye region is darker than upper check region , nose region is brighter than eye region. By this match able features, their location and size will help us to detect a face.




**5.4.**

Here are some Haar feature , using those we can say there is a face or not. Haar feature signifies that black region is represented by +1 and white region is represented by -1 .

It uses a 24X24 window for an image. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.Now all possible sizes and locations of each kernel is used to calculate plenty of features.For each feature calculation, we need to find sum of pixels under white and black rectangles. For 24X24 window, there will be 160000+ Haar features, which is a huge number. To solve this, they introduced the integral images. It simplifies calculation of sumof pixels, how large may be the number of pixels, to an operation involving just four pixels.

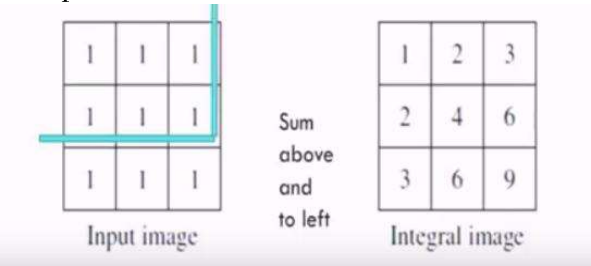
- **Integral Images :**

The basic idea of integral image is that to calculate the area. So, we do not need to sum up all the pixel values rather than we have to use the corner values and then a simple calculation is to be done.

The integral image at location x , y contains the sum of the pixels above and to the left of x , y, inclusive :

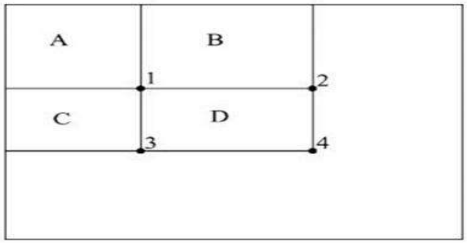
$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y'),$$

For this input image the integrated image will be calculated by summing up all the above and left pixels. Like –




The sum of the pixels within rectangle D can be computed with four array references:

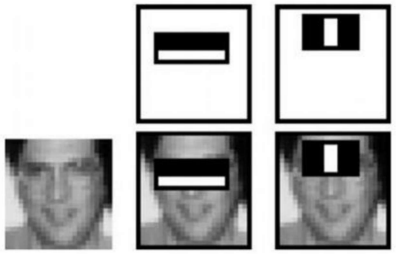
- The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A + B, at location 3 is A + C, and at location 4 is A + B + C + D.
- The sum within D can be computed as 4 + 1 − (2 + 3).




This is easier than the previous one. This is the advantage of converting an image into an integrated image.

- **Adaboost :**

Adaboost is used to eliminate the redundant feature of Haar. A very small number of these features can be combined to form an effective classifier. The main challenge is to find these features. Avariant of AdaBoost is used both to select the features and to train the classifier.



**5.5.**

The second feature is used for detecting the nose bridge, but it is irrelevant for upper lips as upper lips has more or less constant feature. So, we can easily eliminate it. Using adaboost we can determine which are relevant out of 160000+ feature. After finding all the features, a weighted value is added to it which is which is used to evaluate a given window is a face or not.

$F(x) = a1f1(x) + a2f2(x) + a3f3(x) + a4f4(x) + a5f5(x) + \ldots.$

F(x) is strong classifier and f(x) is weak classifier.

Weak classifier always provide binary value i.e. 0 and 1. If the feature is present the value will be 1, otherwise value will be 0. Generally 2500 classifiers are used to make a strong classifier. Here selected features are said to be okay if it perform better than the random guessing i.e. it has to detect more than half of cases.

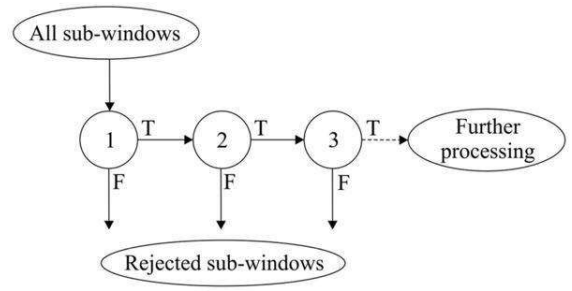
- **Cascading :**

Suppose, we have an input image of 640X480 resolution. Then we need to move 24X24 window through out the image and for each window 2500 features are to be evaluated. Taking all 2500 features in a linear way it checks weather there is any threshold or not and then decide it is a face or not.

But instead of using all 2500 features for 24X24 times we will use cascade. Out of 2500 features , 1st 10 features are classified in one classifier, next 20-30 featutes are in next classifier, then next 100 in another classifier. So, like this we will increase the complexity.

The advantage is we can eliminate non face from 1st step instead of going through all 2500 features for 24X24 window.

Suppose we have an image. If the image pass through 1st stage where 10 classifiers are stored, it may be a face. Then the image will go for 2nd stage checking. It the image does not pass the 1st stage, we can easily eliminate that.




**5.6.**

Cascading is smaller, most efficient classifier . It is very easy to non face areas using cascading.

- **Haar Cascade Classifier in OpenCv :**

The algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our

convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is the plan !!!

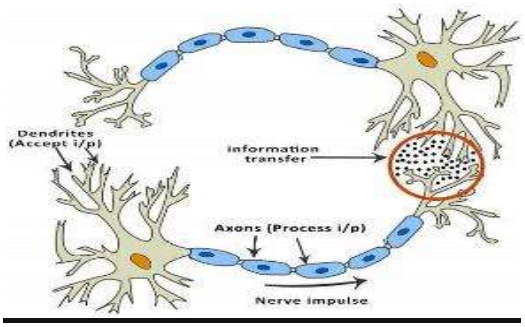
Authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read paper for more details or check out the references in Additional Resources section.

- ## Artificial Neural Networks :

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.




**5.7.**

ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called

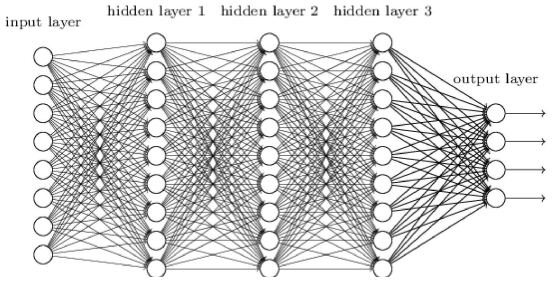
its **activation** or **node value.** Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values.

- ## Deep Convolutional Neural Networks (DCNN) :

  Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.

  - ### Overview of DCNN architecture :

    DCNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells (Hubel & Wiesel, 1959, 1962), motivates their architecture. CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model. It illustrates typical CNN architecture for a toy image classification task. An image is input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations from these operations feed one or more fully connected layers. Finally, the last fully connected layer outputs the class label. Despite this being the most popular base architecture found in the literature, several architecture changes have been proposed in recent years with the objective of improving image classification accuracy or reducing computation costs. Although for the remainder of this section, we merely fleetingly introduce standard CNN architecture.




**5.8.**

  - ### Convolutional Layers :

    The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent through a nonlinear activation function. All neurons within a feature map have weights that are constrained

to be equal; however, different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location.

- **Pooling Layers :**

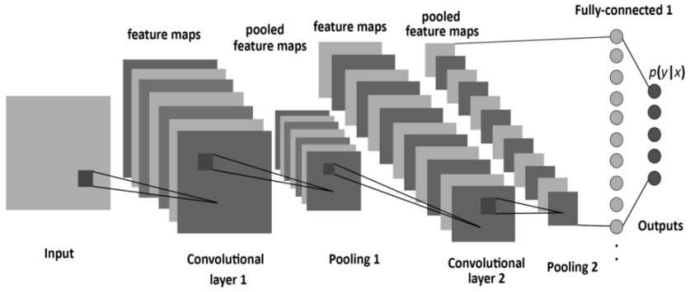
  The purpose of the pooling layers is to reduce the spatial resolution of the feature maps and thus achieve spatial invariance to input distortions and translations. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values, of a small neighborhood of an image to the next layer. However, in more recent models, , max pooling aggregation layers propagate the maximum value within a receptive field to the next layer.

- **Fully Connected Layers :**

  Several convolutional and pooling layers are usually stacked on top of each other to extract more abstract feature representations in moving through the network. The fully connected layers that follow these layers interpret these feature representations and perform the function of high-level reasoning. . For classification problems, it is standard to use the softmax operator on top of a DCNN. While early success was enjoyed by using radial basis functions (RBFs), as the classifier on top of the convolutional towers found that replacing the softmax operator with a support vector machine (SVM) leads to improved classification accuracy.

- **Training :**

  CNNs and ANN in general use learning algorithms to adjust their free parameters in order to attain the desired network output. The most common algorithm used for this purpose is backpropagation. Backpropagation computes the gradient of an objective function to determine how to adjust a network's parameters in order to minimize errors that affect performance. A commonly experienced problem with training CNNs, and in particular DCNNs, is overfitting, which is poor performance on a held-out test set after the network is trained on a small or even large training set. This affects the model's ability to generalize on unseen data and is a major challenge for DCNNs that can be assuaged by regularization.

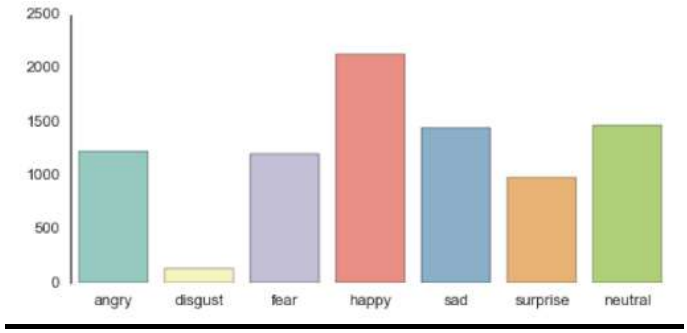



**5.9.**

# 10. IMPLEMENTATION OF PROBLEM :

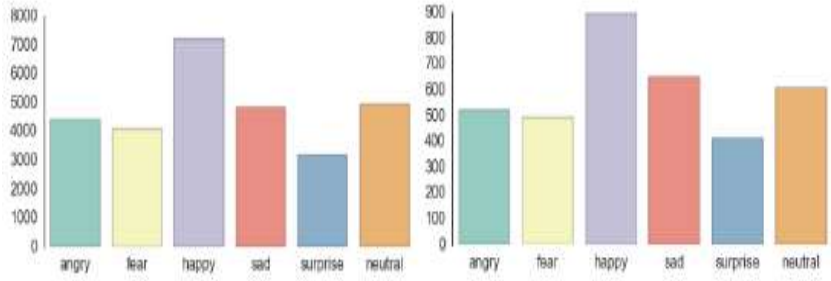
- ## The Database :

The dataset we used for training the model is from a Kaggle Facial Expression Recognition Challenge a few years back (FER2013). It comprises a total of **35887 pre-cropped, 48-by-48-pixel grayscale images** of faces each labeled with one of the 7 emotion classes: **anger, disgust, fear, happiness, sadness, surprise, and neutral**.




**6.1.**

As we were exploring the dataset, we discovered an imbalance of the "disgust" class compared to many samples of other classes. We decided to merge disgust into anger given that they both represent similar sentiment. To prevent data leakage, We built a data generator **fer2013datagen.**py that can easily separate training and hold-out set to different files. We used 28709 labeled faces as the training set and held out the remaining two test sets (3589/set) for after-training validation. The resulting is a **6-class, balanced dataset**, that contains angry, fear, happy, sad, surprise, and neutral. Now we're ready to train.



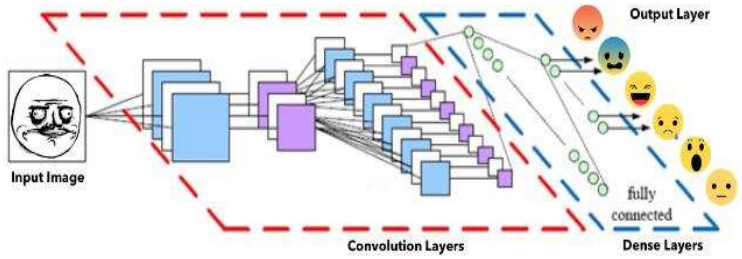

**6.2.**

- ## The Model :

Deep learning is a popular technique used in computer vision. We chose Convolutional Neural Network (CNN) layers as building blocks to create our model architecture. CNNs are known to imitate how the human brain works when analyzing visuals. We have used a picture of Mr. Bean as an example to explain how images are fed into the model, **because who doesn't love Mr. Bean?**

A typical architecture of a convolutional neural network contain an input layer, some convolutional layers, some dense layers (aka. fully-connected layers), and an output

layer . These are linearly stacked layers ordered in sequence. In Keras, the model is created as `Sequential()` and more layers are added to build architecture.




**6.3.**

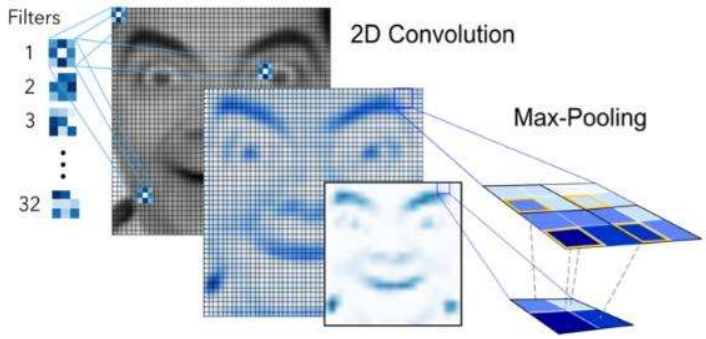
- **Input Layer :**

  The input layer has pre-determined, fixed dimensions, so the image must be **pre-processed** before it can be fed into the layer. We used OpenCV, a computer vision library, for face detection in the image. The haar-cascade_frontalface_default.xml in OpenCV contains pre-trained filters and uses Adaboost to quickly find and crop the face.

  The cropped face is then converted into grayscale using cv2.cvtColor and resized to 48-by-48 pixels with cv2.resize. This step greatly reduces the dimensions compared to the original RGB format with three color dimensions (3, 48, 48). The pipeline ensures every image can be fed into the input layer as a (1, 48, 48) numpy array.

- **Convolutional Layers :**

  The numpy array gets passed into the Convolution2D layer where we specify the number of filters as one of the hyperparameters. The set of filters(aka. kernel) are unique with randomly generated weights. Each filter, (3, 3) receptive field, slides across the original image with shared weights to create a feature map.Convolution generates feature maps that represent how pixel values are enhanced, for example, edge and pattern detection. A feature map is created by applying filter 1 across the entire image. Other filters are applied one after another creating a set of feature maps.

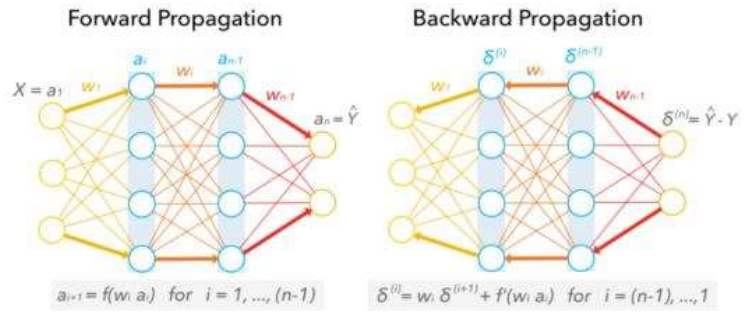



**6.4.**

**Pooling** is a dimension reduction technique usually applied after one or several convolutional layers. It is an important step when building CNNs as adding more convolutional layers can greatly affect computational time. We used a popular pooling method called MaxPooling2D that uses (2, 2) windows across the feature map only keeping the maximum pixel value. The pooled pixels form an image with dimentions reduced by 4.

- ## Dense Layers :

The dense layer (aka fully connected layers), is inspired by the way neurons transmit signals through the brain. It takes a large number of input features and transform features through layers connected with trainable weights.




**6.5.**

These weights are trained by forward propagation of training data then backward propagation of its errors. **Back propagation** starts from evaluating the difference between prediction and true value, and back calculates the weight adjustment needed to every layer before. We can control the training speed and the complexity of the architecture by tuning the hyper-parameters, such as **learning rate** and **network density**. As we feed in more data, the network is able to gradually make adjustments until errors are minimized.Essentially, the more layers/nodes we add to the network the better it can pick up signals. As good as it may sound, the model also becomes increasingly prone to overfitting the training data. One method to prevent overfitting and generalize on unseen data is to apply **dropout**. Dropout randomly selects a portion (usually less than 50%) of nodes to set their weights to zero during training. This method can effectively control the model's sensitivity to noise during training while maintaining the necessary complexity of the architecture.

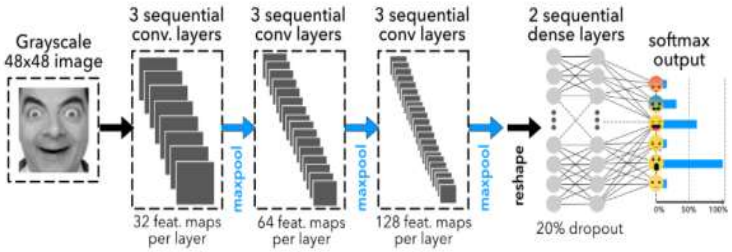
- ## Output Layer :

Instead of using sigmoid activation function, we used **softmax** at the output layer. This output presents itself as a probability for each emotion class.Therefore, the model is able to show the detail probability composition of the emotions in the face. As later on, you will see that it is not efficient to classify human facial expression as only a single emotion. Our expressions are usually much complex and contain a mix of emotions that could be used to accurately describe a particular expression.

Deep Learning we built a simple CNN with an input, three convolution layers, one dense layer, and an output layer to start with. As it turned out, the simple model performed poorly. The low accuracy of 0.1500 showed that it was merely random guessing one of the six emotions. The simple net architecture failed to pick up the subtle details in facial expressions. This could only mean one thing...

This is where deep learning comes in. Given the pattern complexity of facial expressions, it is necessary to build with a deeper architecture in order to identify subtle signals. So we fiddled combinations of three components to increase model complexity:

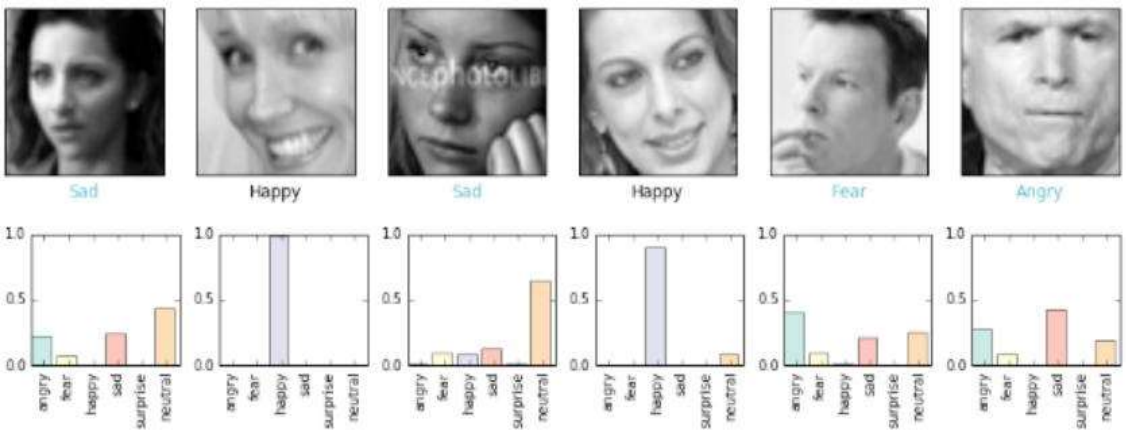
Models with various combinations were trained and evaluated using GPU computing g2.2xlarge on Amazon Web Services (AWS). This greatly reduced training time and increased efficiency in tuning the model. In the end, our final net architecture was 9 layers deep in convolution with one max-pooling after every three convolution layers as seen below.

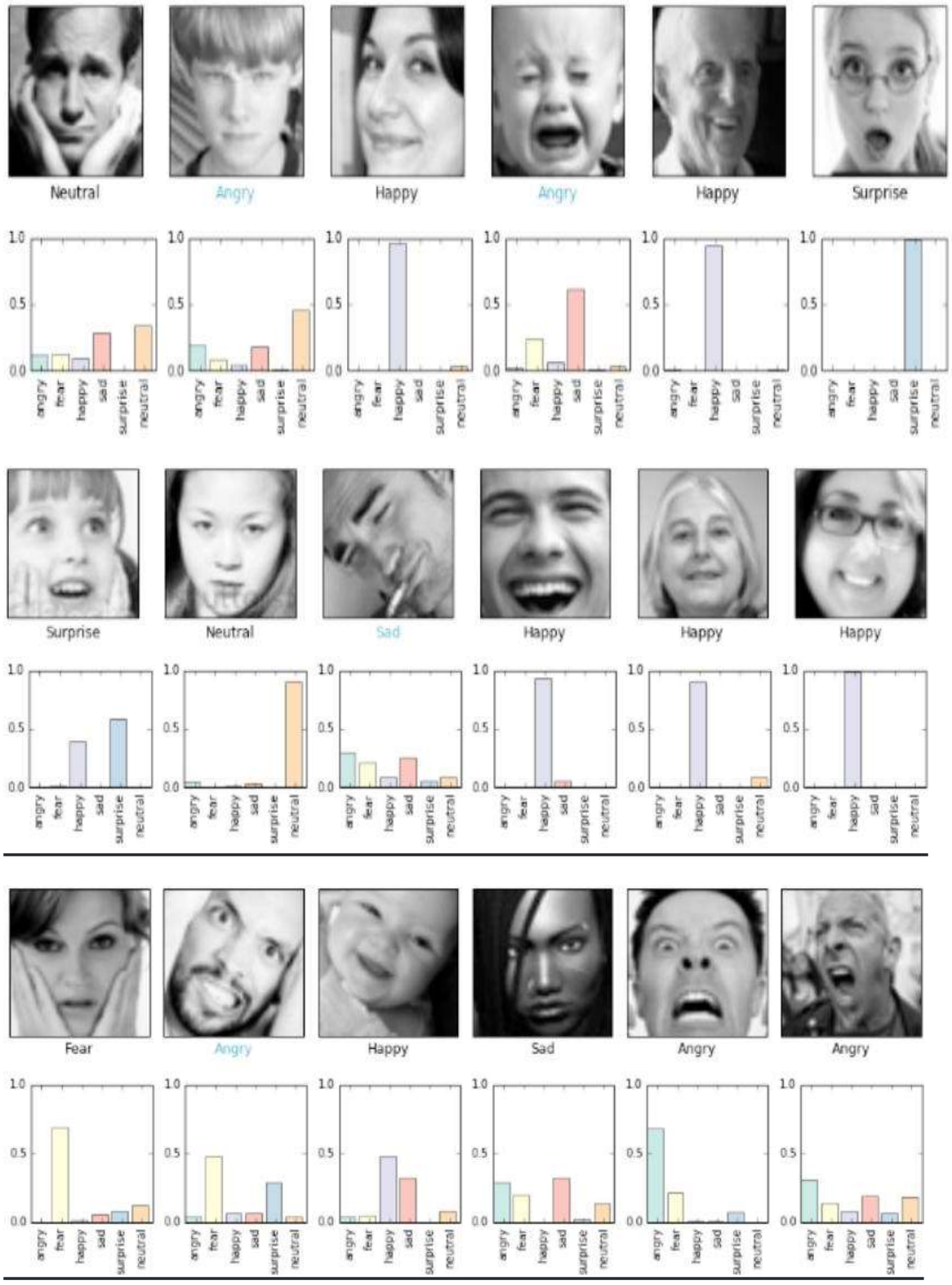



**6.6.**

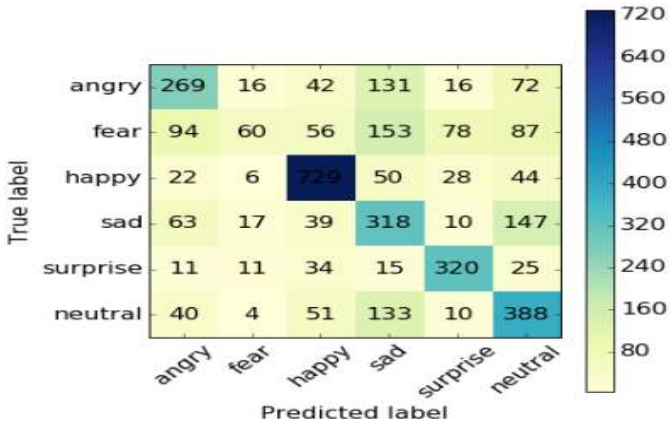
- ## Model Validation :

Performance As it turns out, the final CNN had a **validation accuracy of 58%**. This actually makes a lot of sense. Because our expressions usually consist a combination of emotions, and *only* using one label to represent an expression can be hard. **In this case, when the model predicts incorrectly, the correct label is often the second most likely emotion as seen in figure below.**




Neutral
Angry
Happy
Angry
Happy
Surprise
Surprise
Neutral
Sad
Happy
Happy
Happy
Fear
Angry
Happy
Sad
Angry
Angry
angry
fear
happy
sad
surprise
neutral
0.0
0.5
1.0

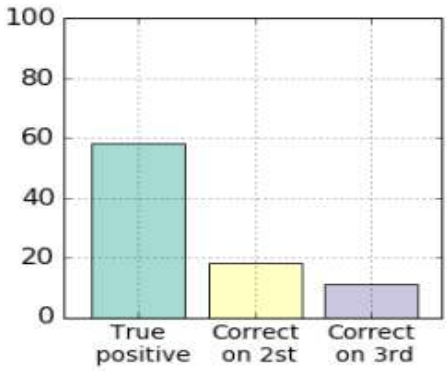
**6.7.**




**6.8.**

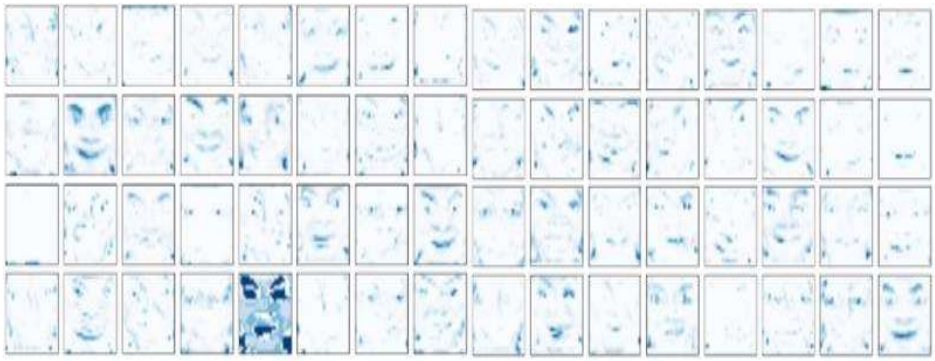
The confusion matrix gives the counts of emotion predictions and some insights to the performance of the multi-class classification model :

- The model performs really well on classifying **positive emotions** resulting in relatively high precision scores for happy and surprised. **Happy** has a precision of 76.7% which could be explained by having the most examples (~7000) in the training set. Interestingly, **surprise** has a precision of 69.3% having the least examples in the training set. There must be very strong signals in the surprise expressions.
- Model performance seems weaker across **negative emotions** on average. In particularly, the emotion **sad** has a low precision of only 39.7%. The model frequently misclassified angry, fear and neutral as sad. In addition, it is most confused when predicting sad and neutral faces because these two emotions are probably the least expressive (excluding crying faces).
- Frequency of prediction that misclassified by less than 3 ranks.

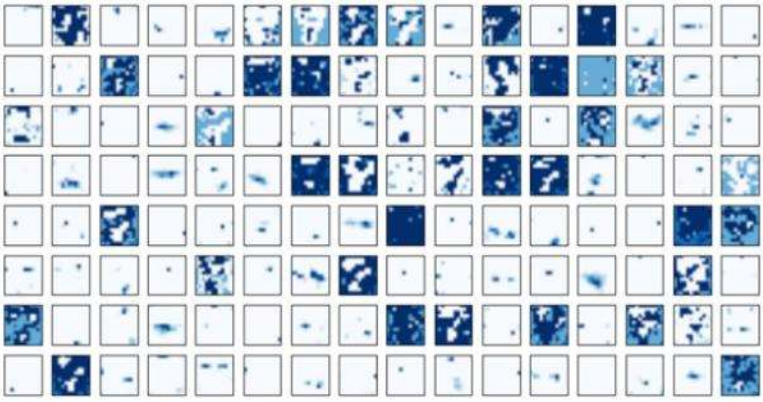



**6.9.**

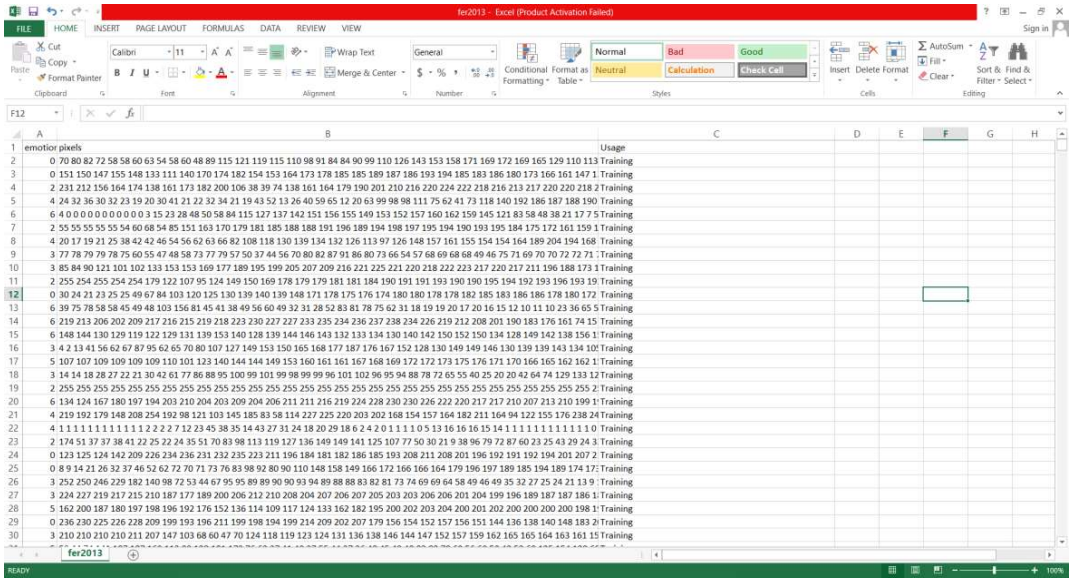
Computer Vision As a result, the feature maps become increasingly abstract down the pipeline when more pooling layers are added. This gives an idea of what the machine sees in feature maps after 2nd and 3rd max-pooling.



6.10.



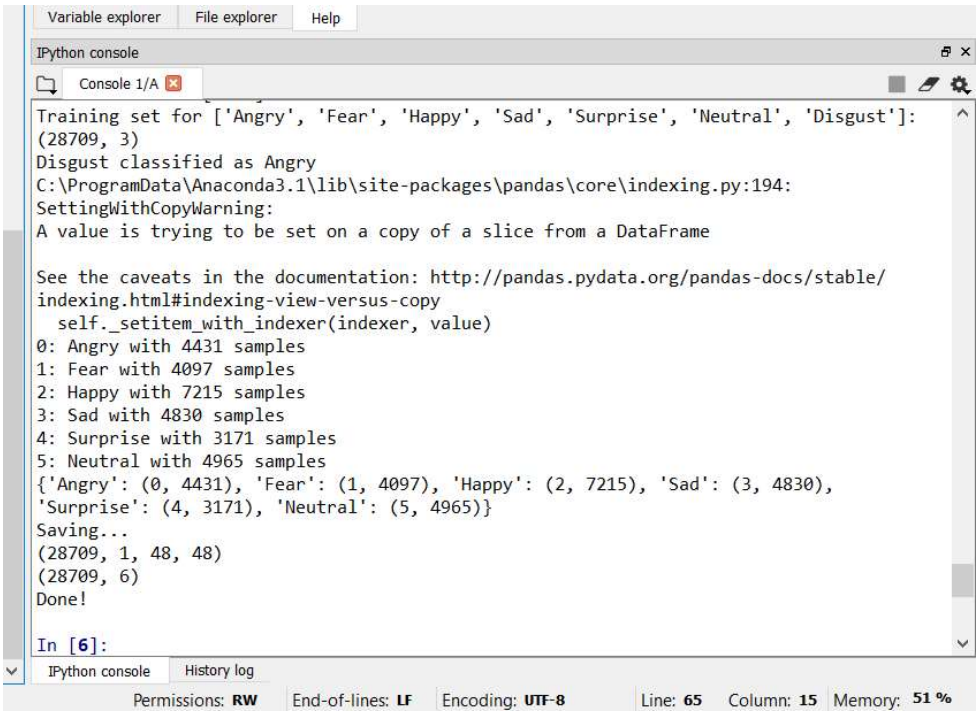
6.11.


fer2013 - Excel (Product Activation Failed)
FILE
HOME
INSERT
PAGE LAYOUT
FORMULAS
DATA
REVIEW
VIEW
Sign in
Cut
Copy
Format Painter
Paste
Clipboard
Calibri
11
Font
Wrap Text
Merge & Center
Alignment
General
Number
Conditional Formatting
Format as Table
Normal
Bad
Good
Neutral
Calculation
Check Cell
Styles
Insert
Delete
Format
Cells
AutoSum
Fill
Clear
Sort & Filter
Find & Select
Editing
F12
emotion
pixels
Usage
Training
fer2013
READY
100%

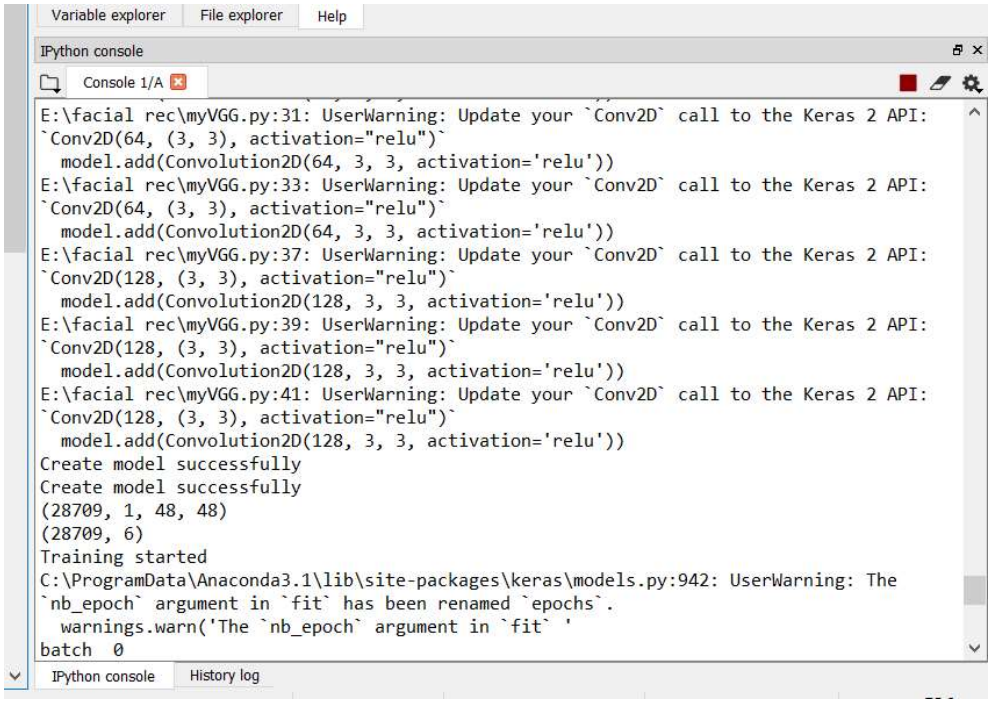
6.12.

# 11. RESULT :

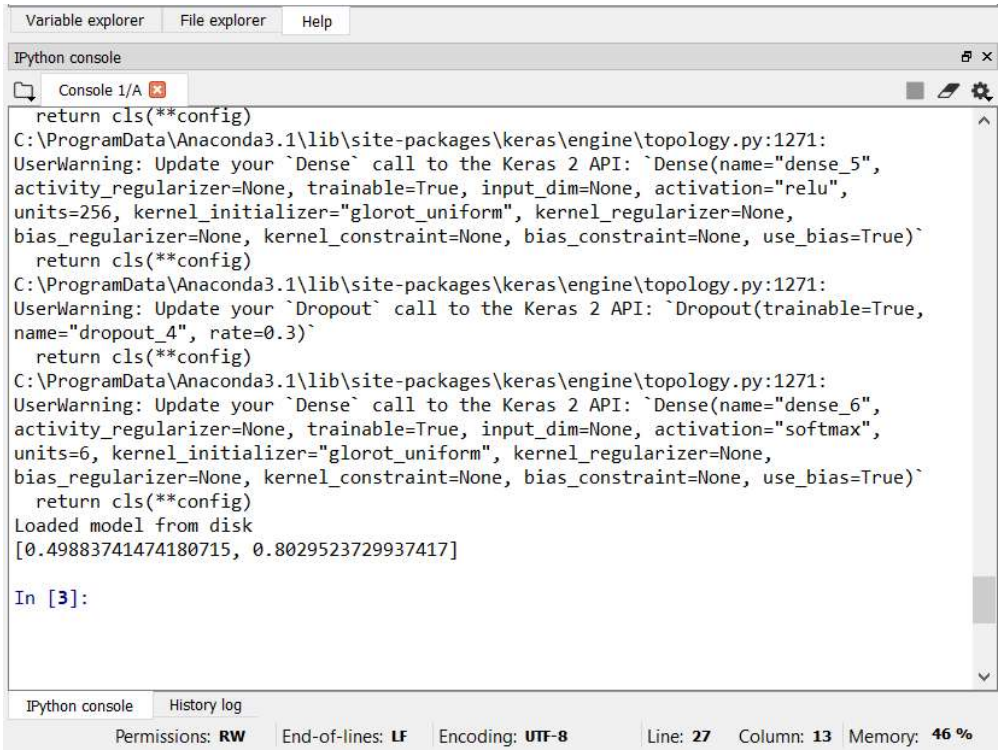
## Dataset Trained Succesfully




## Json Model Created Succesfully



# Accuracy Check Of The Model



## Output Sample 1

7.1.

7.2.

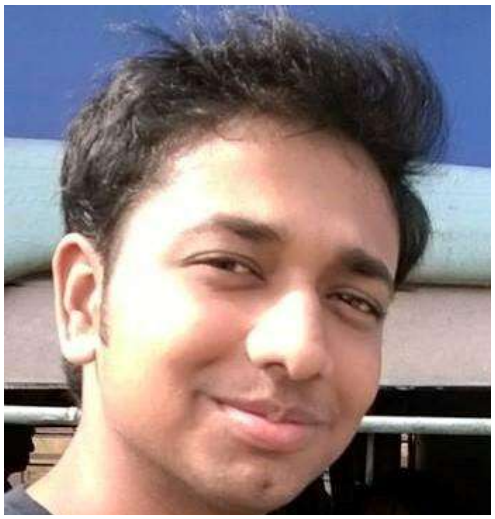
7.3.

```
from PIL import Image
from resizeimage import resizeimage

img11 = Image.open('ang2.jpg').convert('L')

img11.save('gimage10.jpg')


with open('gimage10.jpg', 'r+b') as f:
    with Image.open(f) as image:
        cover = resizeimage.resize_cover(image, [48, 48])
        cover.save('test-image-cover10.jpeg', image.format)

img11.save('resized_image12.jpg')

image = misc.imread('test-image-cover10.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
```

```
IPython console
Console 1/A
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  image = misc.imread('test-image-cover10.jpeg')
[[0.0779406  0.18848301 0.04663404 0.48889568 0.04269835 0.15534832]]

In [25]:
IPython console  History log
Permissions: RW  End-of-lines: LF  Encoding: UTF-8  Line: 63  Column: 1  Memory:
```

[[0.0779406 0.18848301 0.04663404 0.48889568 0.04269835 0.15534832]]

## Output Sample 2

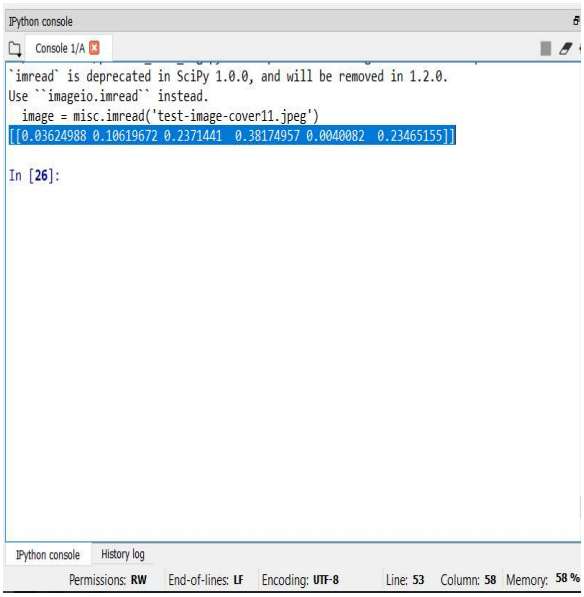


7.4.

7.5.

7.6.

```
from PIL import Image
from resizeimage import resizeimage

img12 = Image.open('sou.jpg').convert('L')

img12.save('gimage11.jpg')


with open('gimage11.jpg', 'r+b') as f:
    with Image.open(f) as image:
        cover = resizeimage.resize_cover(image, [48, 48])
        cover.save('test-image-cover11.jpeg', image.format)

img12.save('resized_image13.jpg')

image = misc.imread('test-image-cover11.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
```




```
[[0.03624988  0.10619672  0.2371441  0.38174957  0.0040082
0.23465155]]
```

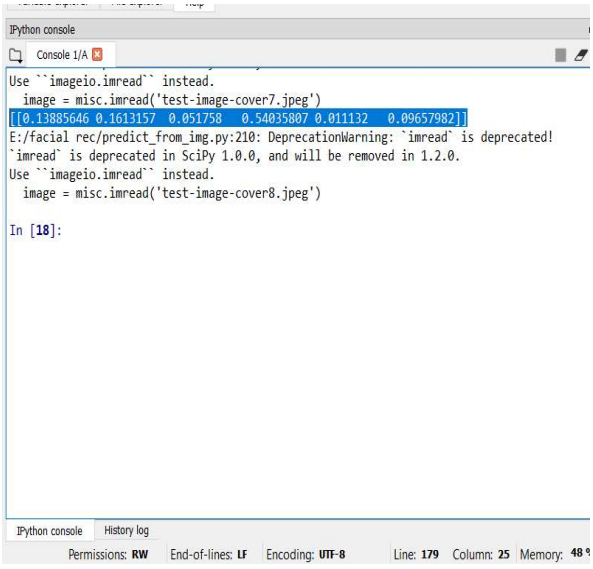
## Output Sample 3

7.7.

7.8.

7.9.

```
from PIL import Image
from resizeimage import resizeimage

img8 = Image.open('sus1.jpg').convert('L')

img8.save('gimage7.jpg')


with open('gimage7.jpg', 'r+b') as f:
    with Image.open(f) as image:
        cover = resizeimage.resize_cover(image, [48, 48])
        cover.save('test-image-cover7.jpeg', image.format)

img8.save('resized_image9.jpg')

image = misc.imread('test-image-cover7.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
```




```
[[0.13885646  0.1613157  0.051758  0.54035807  0.011132
0.09657982]]
```

# 12. CONCLUSION :

**In this case, when the model predicts incorrectly, the correct label is often the second most likely emotion.**

The facial expression recognition system presented in this research work contributes a resilient face recognition model based on the mapping of behavioral characteristics with the physiological biometric characteristics. The physiological characteristics of the human face with relevance to various expressions such as happiness, sadness, fear, anger, surprise and disgust are associated with geometrical structures which restored as base matching template for the recognition system.

The behavioral aspect of this system relates the attitude behind different expressions as property base. The property bases are alienated as exposed and hidden category in genetic algorithmic genes. The gene training set evaluates the expressional uniqueness of individual faces and provide a resilient expressional recognition model in the field of biometric security.

The design of a novel asymmetric cryptosystem based on biometrics having features like hierarchical group security eliminates the use of passwords and smart cards as opposed to earlier cryptosystems. It requires a special hardware support like all other biometrics system. This research work promises a new direction of research in the field of asymmetric biometric cryptosystems which is highly desirable in order to get rid of passwords and smart cards completely. Experimental analysis and study show that the hierarchical security structures are effective in geometric shape identification for physiological traits.

# 13. FUTURE SCOPE :

It is important to note that there is no specific formula to build a neural network that would guarantee to work well. Different problems would require different network architecture and a lot of trail and errors to produce desirable validation accuracy. **This is the reason why neural nets are often perceived as "black box algorithms."**.

In this project we got an accuracy of almost 70% which is not bad at all comparing all the previous models. But we need to improve in specific areas like-

- **number and configuration of convolutional layers**
- **number and configuration of dense layers**
- **dropout percentage in dense layers**

But due to lack of highly configured system we could not go deeper into dense neural network as the system gets very slow and we will try to improve in these areas in future.

We would also like to train more databases into the system to make the model more and more accurate but again resources becomes a hindrance in the path and we also need to improve in several areas in future to resolve the errors and improve the accuracy.

Having examined techniques to cope with expression variation, in future it may be investigated in more depth about the face classification problem and optimal fusion of color and depth information. Further study can be laid down in the direction of allele of gene matching to the geometric factors of the facial expressions. The genetic property evolution framework for facial expressional system can be studied to suit the requirement of different security models such as criminal detection, governmental confidential security breaches etc.

# 14. REFERENCES :


- **A literature survey on Facial Expression Recognition using Global Features**by Vaibhavkumar J. Mistry and Mahesh M. Goyani,International Journal of Engineering and Advanced Technology (IJEAT),April,2013 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.645.5162&rep=rep1&type=pdf]

- **Convolutional Neural Networks (CNN) With TensorFlow** by Sourav from Edureka[https://www.youtube.com/watch?v=umGJ30-15_A]

- **A Survey on Facial Expression Recognition Techniques** by Swati Mishra(Chhattisgarh Swami Vivekanand Technical University, Department of Computer Science & Engineering, Raipur Institute of Technology, Mandir Hasaud, Raipur, Chhattisgarh, India) andAvinash Dhole( Assistant Professor, Department of Computer Science & Engineering, RaipurInstitute of Technology, Chhattisgarh Swami Vivekanand andTechnical University, Mandir Hasaud, Raipur, Chhattisgarh, India) , International Journal of Science and Research (IJSR),2013 [https://pdfs.semanticscholar.org/e241/25e4e9471a33ba2c7f0979541199caa02f8b.pdf]

- **Recognizing Facial Expressions Using Deep Learning** by Alexandru Savoiu Stanford University and James Wong Stanford University [http://cs231n.stanford.edu/reports/2017/pdfs/224.pdf]

- **Deep Learning Simplified** by Sourav from Edureka [https://www.youtube.com/watch?v=dafuAz_CV7Q&list=PL9ooVrP1hQOEX8BKDplfG86ky8s7Oxbzg]

- **Predicting facial expressions with machine learning algorithms** by Alex Young , Andreas Eliasson , Ara Hayrabedian , Lukas Weiss , Utku Ozbulak[https://github.com/utkuozbulak/facial-expression-recognition]

- **"Robust Real-Time Face Detection"**, International Journal of Computer Vision 57(2), 137–154, 2004
- "**Facial expressions of emotions: an old controversy and new finding discussion**", by P. Ekman, E. T. Rolls, D. I. Perrett, H. D. Ellis, Pill Trans. Royal Soc. London Ser. B, Biol. Sci., 1992, vol. 335, no. 1273, pp. 63-69.

- **Going Deeper in Facial Expression Recognition using Deep Neural Networks**, by Ali Mollahosseini1, David Chan2, and Mohammad H. Mahoor1 Department of Electrical and Computer Engineering, Department of Computer Science, University of Denver, Denver, CO.

- **Journal on Convoluted Neural Network** by IIIT,Hyderabad.
- **Journal on Artificial Intellegence** by Prof. M.K. Anand, 2014.
- Journal by Ghuangjhow University on Image Processing.
- Project_Report_On_FACIAL_EXPRESSION_RECOGNITION_USING_IMAGE_PROCESSING
- **VIOLA JONES FACE DETECTION EXPLAINATION** by Rahul Patil.
- Wikipedia- **Artificial Neural Netwok & Convoluted Neural Netwok**
- Neeta Sarode et. Al ./(IJCSE) International Journal on Computer Science.
- **Facial Expression Detection Techniques: Based on Viola and Jones algorithm and Principal Component Analysis** by Samiksha Agrawal and Pallavi Khatri, ITM University Gwalior(M.P.), India, 2014.
- **Project_Report** by Udacity, INDIA on **Image Processing,** 2013

# 15. APPENDIX :

## code1.py

```
Created on Sat Apr 21 12:39:40 2018
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: angana
"""
#data generation
fromkeras.utils.np_utils import to_categorical
import pandas as pd
importnumpy as np
import random
import sys
# fer2013 dataset:
# Training       28709
# PrivateTest     3589
# PublicTest      3589
# emotion labels from FER2013:
emotion = {'Angry': 0, 'Disgust': 1, 'Fear': 2, 'Happy': 3,
          'Sad': 4, 'Surprise': 5, 'Neutral': 6}
emo     = ['Angry', 'Fear', 'Happy',
          'Sad', 'Surprise', 'Neutral']
def reconstruct(pix_str, size=(48,48)):
pix_arr = []
for pix in pix_str.split():
pix_arr.append(int(pix))
pix_arr = np.asarray(pix_arr)
returnpix_arr.reshape(size)
defemotion_count(y_train, classes, verbose=True):
```

```
emo_classcount = {}
print ('Disgust classified as Angry')
y_train.loc[y_train == 1] = 0
classes.remove('Disgust')
fornew_num, _class in enumerate(classes):
y_train.loc[(y_train == emotion[_class])] = new_num
class_count = sum(y_train == (new_num))
if verbose:
print ('{}: {} with {} samples'.format(new_num, _class, class_count))
emo_classcount[_class] = (new_num, class_count)
returny_train.values, emo_classcount

defload_data(sample_split=0.3, usage='Training', to_cat=True, verbose=True,
classes=['Angry','Happy'], filepath='fer2013.csv'):
df = pd.read_csv(filepath)
    # printdf.tail()
    # printdf.Usage.value_counts()
df = df[df.Usage == usage]
frames = []
classes.append('Disgust')
for _class in classes:
class_df = df[df['emotion'] == emotion[_class]]
frames.append(class_df)
data = pd.concat(frames, axis=0)
rows = random.sample(list(data.index), int(len(data)*sample_split))
data = data.ix[rows]
print ('{} set for {}: {}'.format(usage, classes, data.shape))
data['pixels'] = data.pixels.apply(lambda x: reconstruct(x))
    x = np.array([mat for mat in data.pixels]) # (n_samples, img_width, img_height)
X_train = x.reshape(-1, 1, x.shape[1], x.shape[2])
y_train, new_dict = emotion_count(data.emotion, classes, verbose)
print (new_dict)
```

```
ifto_cat:
y_train = to_categorical(y_train)
returnX_train, y_train, new_dict
defsave_data(X_train, y_train, fname='', folder=''):
np.save(folder + 'X_train' + fname, X_train)
np.save(folder + 'y_train' + fname, y_train)

if __name__ == '__main__':
    # makes the numpy arrays ready to use:
print ('Making moves...')
emo = ['Angry', 'Fear', 'Happy',
          'Sad', 'Surprise', 'Neutral']
    '''
X_train, y_train, emo_dict = load_data(sample_split=1.0,
classes=emo,
usage='PrivateTest',
verbose=True)
    '''
X_train, y_train, emo_dict = load_data(sample_split=1.0,
classes=emo,
usage='Training',
verbose=True)
print ('Saving...')
save_data(X_train, y_train, fname='_train')
print (X_train.shape)
print (y_train.shape)
print ('Done!')
```

## code2.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Sat Apr 21 12:53:59 2018
@author: angana
"""
import sys, os
import cv2
importnumpy as np
def preprocessing(img, size=(48, 48)):
img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
img = cv2.resize(img, size).astype(np.float32)
   #img = img.transpose((2, 0, 1))
   # img = np.expand_dims(img, axis=0)
returnimg
defextract_features(path):
   X, y = [], []
label = 0
fordirnames in os.listdir(path):
       # print(dirnames)
sub_path = os.path.join(path, dirnames)
       # print(sub_path)
for filename in os.listdir(sub_path):
           # print (filename)
file_path = os.path.join(sub_path, filename)
img = cv2.imread(file_path)
img = preprocessing(img)
X.append(img)
class_label = [0, 0, 0, 0, 0, 0, 0]
class_label[label] = 1
y.append(class_label)
label += 1
 X = np.asarray(X)
   y = np.asarray(y)
return X, y
```

```
if __name__ == "__main__" :
    X, y = extract_features(sys.argv[1])
print(X, y)
print(type(X), type(X[0]))
print(X[212])
print(len(X))
```

## code3.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 21 13:04:13 2018
@author: angana
"""
importargparse
import cv2
importnumpy as np
import code2 as fu
importmyVGG
fromkeras.callbacks import LambdaCallback, EarlyStopping
parser = argparse.ArgumentParser(description=("Model training process."))
# parser.add_argument('data_path', help=("The path of training data set"))
parser.add_argument('--test', help=("Input a single image to check if the model works well."))
args = parser.parse_args()
def main():
model = myVGG.VGG_16()
ifargs.test is not None:
print ("Test mode")
img = cv2.imread(args.test)
img = fu.preprocessing(img)
img = np.expand_dims(img, axis=0)
        y = np.expand_dims(np.asarray([0]), axis=0)
```

```
batch_size = 1024
model.fit(img, y, nb_epoch=400, \
batch_size=batch_size, \
validation_split=0.2, \
shuffle=True, verbose=0)
return
#input_path = args.data_path
    #print("training data path : " + input_path)
    #X_train, y_train = fu.extract_features(input_path)
X_fname = 'X_train_train.npy'
y_fname = 'y_train_train.npy'
X_train = np.load(X_fname)
y_train = np.load(y_fname)
print(X_train.shape)
print(y_train.shape)
print("Training started")
callbacks = []
earlystop_callback = EarlyStopping(monitor='val_loss', patience=5, verbose=0)
batch_print_callback = LambdaCallback(on_batch_begin=lambda batch, logs: print('batch
',batch))
epoch_print_callback = LambdaCallback(on_epoch_end=lambda epoch, logs: print("epoch:",
epoch))
callbacks.append(earlystop_callback)
callbacks.append(batch_print_callback)
callbacks.append(epoch_print_callback)
batch_size = 1024
model.fit(X_train, y_train, nb_epoch=400, \
batch_size=batch_size, \
validation_split=0.2, \
shuffle=True, verbose=0, \
callbacks=callbacks)
model.save_weights('my_model_weights.h5')
scores = model.evaluate(X_train, y_train, verbose=0)
```

```
print ("Train loss : %.3f" % scores[0])
print ("Train accuracy : %.3f" % scores[1])
print ("Training finished")
if __name__ == "__main__":
main()
```

## myVGG.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 21 13:42:16 2018
@author: angana
""" importos, sys
module_path = os.path.abspath(os.path.join('.'))
sys.path.append(module_path)
fromkeras.models import Sequential
fromkeras.layers.core import Flatten, Dense, Dropout
fromkeras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
fromkeras.optimizers import SGD
import cv2, numpy as np
fromkeras import backend as K
K.set_image_dim_ordering('th')
def VGG_16(weights_path=None, shape=(48, 48)):
model = Sequential()
model.add(ZeroPadding2D((1,1), input_shape=(1, 48, 48)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
```

```
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
" model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
'"model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
print ("Create model successfully")
ifweights_path:
model.load_weights(weights_path)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', \
metrics=['accuracy'])
model_json = model.to_json()
with open("model.json", "w") as json_file:
json_file.write(model_json)
return model
VGG_16()
```

## evaluate_model.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 21 13:42:16 2018

@author: angana
"""
importos, sys
module_path = os.path.abspath(os.path.join('.'))
sys.path.append(module_path)
fromkeras.models import Sequential
fromkeras.layers.core import Flatten, Dense, Dropout
fromkeras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
fromkeras.optimizers import SGD
import cv2, numpy as np
fromkeras import backend as K
K.set_image_dim_ordering('th')
def VGG_16(weights_path=None, shape=(48, 48)):
model = Sequential()
model.add(ZeroPadding2D((1,1), input_shape=(1, 48, 48)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
```

```
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
''' model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
   '' model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dense(6, activation='softmax'))
print ("Create model successfully")
ifweights_path:
model.load_weights(weights_path)
model.compile(optimizer='adam', loss='categorical_crossentropy', \
metrics=['accuracy'])
model_json = model.to_json()
with open("model.json", "w") as json_file:
json_file.write(model_json)
return model
VGG_16()
```

## predict_from_img.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 22 21:20:10 2018
@author: angana
"""
fromkeras.models import model_from_json
from flask import Blueprint, request, jsonify
importjson
# load json and create model arch
json_file = open('model_1.json','r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
# load weights into new model
model.load_weights('model_1.h5')
import cv2
import base64
def base64_encode_image(image_rgb):
```

```
image_bgr = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
ret, image_buf = cv2.imencode('.jpg', image_bgr, (cv2.IMWRITE_JPEG_QUALITY, 40))
image_str = base64.b64encode(image_buf)
return '' + image_str
defpredict_emotion(face_image_gray): # a single cropped face
resized_img = cv2.resize(face_image_gray, (48,48), interpolation = cv2.INTER_AREA)
    # cv2.imwrite(str(index)+'.png', resized_img)
image = resized_img.reshape(1, 1, 48, 48)
list_of_list = model.predict(image, batch_size=1, verbose=1)
angry, fear, happy, sad, surprise, neutral = [prob for lst in list_of_list for prob in lst]
return  angry, fear, happy, sad, surprise, neutral
fromscipy import misc
importnumpy as np
from PIL import Image
image = misc.imread('resized_image.png')
Image.fromarray(image)
image=image[:,:,0]
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
from PIL import Image
img = Image.open('me.jpg').convert('L')
baseheight = 48
hpercent = (baseheight / float(img.size[1]))
wsize = int((float(img.size[0]) * float(hpercent)))
img = img.resize((wsize, baseheight), Image.ANTIALIAS)
img.save('resized_image1.jpg')
image = misc.imread('resized_image1.jpg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
from PIL import Image
```

```
fromresizeimage import resizeimage
img1 = Image.open('happy.jpg').convert('L')
img1.save('gimage.jpg')
with open('gimage.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
cover.save('test-image-cover.jpeg', image.format)
img1.save('resized_image2.jpg')
image = misc.imread('test-image-cover.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
from PIL import Image
fromresizeimage import resizeimage
img2 = Image.open('Bean.jpg').convert('L')
img2.save('gimage1.jpg')
with open('gimage1.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
cover.save('test-image-cover1.jpeg', image.format)
img2.save('resized_image3.jpg')
image = misc.imread('test-image-cover1.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
from PIL import Image
fromresizeimage import resizeimage
img3 = Image.open('hap.jpg').convert('L')
img3.save('gimage2.jpg')
with open('gimage2.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
```

```
cover.save('test-image-cover2.jpeg', image.format)
img3.save('resized_image4.jpg')
image = misc.imread('test-image-cover2.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
from PIL import Image
fromresizeimage import resizeimage
img4 = Image.open('sad.jpg').convert('L')
img4.save('gimage3.jpg')
with open('gimage3.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
cover.save('test-image-cover3.jpeg', image.format)
img4.save('resized_image5.jpg')
image = misc.imread('test-image-cover3.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
jpeg', image.format)
img5.save('resized_image6.jpg')from PIL import Image
fromresizeimage import resizeimage
img5 = Image.open('angry.jpg').convert('L')
img5.save('gimage4.jpg')
with open('gimage4.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
cover.save('test-image-cover4.
image = misc.imread('test-image-cover4.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
```

```
from PIL import Image
fromresizeimage import resizeimage
img6 = Image.open('angry2.jpg').convert('L')
img6.save('gimage5.jpg')
with open('gimage5.jpg', 'r+b') as f:
withImage.open(f) as image:
cover = resizeimage.resize_cover(image, [48, 48])
cover.save('test-image-cover5.jpeg', image.format)
img6.save('resized_image7.jpg')
image = misc.imread('test-image-cover5.jpeg')
image=image.reshape(1,1,48,48)
score=model.predict(image)
print(score)
```