# The Facial Emotion Detection using Neural Network

**A PROJECT REPORT**
**Submitted By**

**Anil Kumar Singh**
**2000290140018**

**Akash Kumar Singh**
**2000290140011**

**Submitted in partial fulfilment of the Requirements for the Degree of**

## MASTER OF COMPUTER APPLICATIONS

**Under the Supervision of**
**Dr. Akash Rajak**
**Professor**


**Submitted to**
**DEPARTMENT OF COMPUTER APPLICATIONS**
**KIET Group of Institutions, Delhi-NCR, Ghaziabad**
**MAY 2022**

# CERTIFICATE

Certified that **Anil Kumar Singh(2000290140018),Akash Kumar Singh(2000290140011)** have carried out the project work having "**The Facial Detection using Neural Network"** for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

**Anil Kumar Singh 2000290140018**
**Akash Kumar Singh 2000290140011**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Date:**

**Dr. Akash Rajak**

**Professor**
**Department of Computer Applications**
**KIET Group of Institutions, Ghaziabad**

**Signature of Internal Examiner** **Signature of External Examiner**

**Dr. Ajay K Shrivastava**

**Head, Department of Computer Applications**

**KIET Group of Institutions, Ghaziabad - 201206**

# ABSTRACT

Our Mood Recognition System identifies emotions to the best of its capabilities depending on the internet and the hardware implemented in the system. communication that varies in complexity, intensity, and meaning. Purposed system depends upon human face as we know face also reflects the human brain activities or emotions.

A Facial expression is the visible manifestation of the affective state, cognitive activity, intention, personality, and psychopathology of a person and plays a communicative role in interpersonal relations. Automatic recognition of facial expressions can be an important component of natural human-machine interfaces; it may also be used in behavioural science and in clinical practice. An automatic Facial Expression Recognition system needs to perform detection and location of faces in a cluttered scene, facial feature extraction, and facial expression classification.

Facial expression recognition system is implemented using Convolution Neural Network (CNN). Kaggle facial expression dataset with seven facial expression labels as happy, sad, surprise, fear, anger, disgust, and neutral is used in this project. The system achieved 56.77 % accuracy and 0.57 precision on testing dataset.

# ACKNOWLEDGEMENTS

Success in life is never attained single headedly. My deepest gratitude goes to my thesis supervisor, **Dr. Akash Rajak** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Ajay Kumar Shrivastava**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Anil Kumar Singh**

**Akash Kumar Singh**

# TABLE OF CONTENTS

# Chapter 1 Introduction

## 1.1 Overview

A Facial expression is the visible manifestation of the affective state, cognitive activity, intention, personality, and psychopathology of a person and plays a communicative role in interpersonal relations. 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.

Automatic recognition of facial expressions can be an important component of natural human machine interfaces; it may also be used in behavioural science and in clinical practice. It has been studied for a long period of time and obtaining the progress recent decades. Though much progress has been made, recognizing facial expression with a high accuracy remains to be difficult due to the complexity and varieties of facial expressions.

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feedforward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation. Convolutional networks were inspired by biological processes and are variations of multilayer perceptron designed to use minimal amounts of pre-processing.

They have wide applications in image and video recognition, recommender systems and natural language processing. The convolutional neural network is also known as shift invariant or space invariant artificial neural network (SIANN), which is named based on its shared weights architecture and translation invariance characteristics.

An emotion recognition system can detect the emotion condition of a person either from his image or speech information. In this scope, an audio-visual emotion recognition system requires to evaluate the emotion of a person from his speech and image information together.

The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. The convolution layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height) but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have size 3x5x5 (i.e., images have depth 3 i.e. the colour channels, 5 pixels width and height). During the forward pass, each filter is convolved across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As the filter convolve over the width and height of the input volume it produces a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, there will be an entire set of filters in each convolution layer (e.g., 20 filters), and each of them will produce a separate 2-dimensional activation map.

A filter convolves with the input image to produce a feature map. The convolution of another filter over the same image gives a different feature map. Convolution operation captures the local dependencies in the original image. A CNN learns the values of these filters on its own during the training process (although parameters such as number of filters, filter size, architecture of the network etc. still needed to specify before the training process). The greater number of filters, the more image features get extracted and the better network becomes at recognizing patterns in unseen images.

The size of the Feature Map (Convolved Feature) is controlled by three parameters

- Depth: Depth corresponds to the number of filters we use for the convolution operation.
- Stride: Stride is the size of the filter, if the size of the filter is 5x5 then stride is 5.
- Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that filter can be applied to bordering elements of input image matrix. Using zero padding size of the feature map can be controlled.

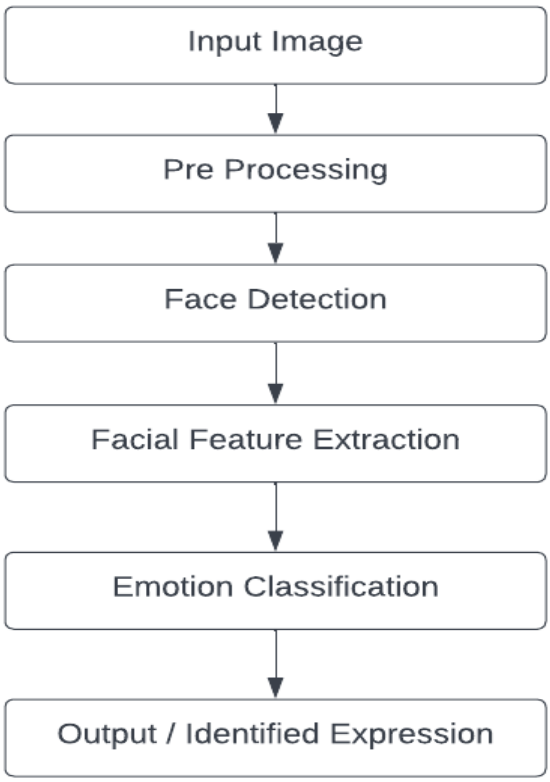
**PROBLEM DEFINITION –**

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 can 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**




**Literature Study** –

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

(i) Pre-processing

(ii) Face registration

(iii) Facial feature extraction

(iv) Emotion classification

Description about all these processes are given below-

**(i) Pre-processing:** Pre-processing is a common name for operations with images at the lowest level of abstraction both input and output are intensity images. Most pre-processing steps that are implemented are –

a. Reduce the noise

b. Convert the Image to Binary/Grayscale.

c. Pixel Brightness Transformation.

d. Geometric Transformation.

**(ii) 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 “face registration”.

**(iii) 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

**(iv) Emotion Classification:** In the third step of classification, the algorithm attempts to classify the given faces portraying one of the seven basic emotions.

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 expressions.

Anger Disgust Fear Joy


**Figure Seven basic human emotions**

**Different approaches which are followed for Facial Expression Recognition:**

## 1.2 Project Scope

Modern day security systems rely heavily on bioinformatics, like speech, fingerprint, facial images and so on. Besides, determination of a user's emotional state with facial and voice analysis plays a fundamental part in human-machine interaction (HMI) systems, since it employs non-verbal cues to estimate the user's emotional state. This software system will be able to perform emotion recognition from audio, video and audio-visual video. With the easy-to-use user-interface of the system, the user can either record instant video/real time or upload an existing video to the system and perform emotion recognition. This system allows big corporate companies to measure customer satisfaction and perform the necessary analysis.

## 1.3 Hardware/Software Used in Project

### (i) Hardware Specification

- Central Processing Unit (CPU) - Intel Core i5 6th gen or AMD processor equivalent
- RAM - 8 GB minimum, 16 GB or higher is recommended.
- Graphics Processing Unit (GPU) - NVIDIA GeForce GTX 960 or higher
- Inbuilt Camera or Webcam Support
- Operating System (OS) - Ubuntu or Microsoft Windows 10
- Storage - 20 GB

- Any IDE, we used Visual Studio Code IDE

**(ii) Software Specification**

- Python 3
- Library - open cv

**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. Analysing 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.

# Chapter 2 Design

## 2.1 Data flow diagram-

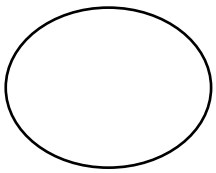
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into detail, which can later be elaborated. DFDs can also be used for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

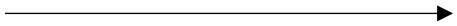
**External entity:** an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system, or a business system. They are also 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 labelled 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 analysed or modelled. 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.

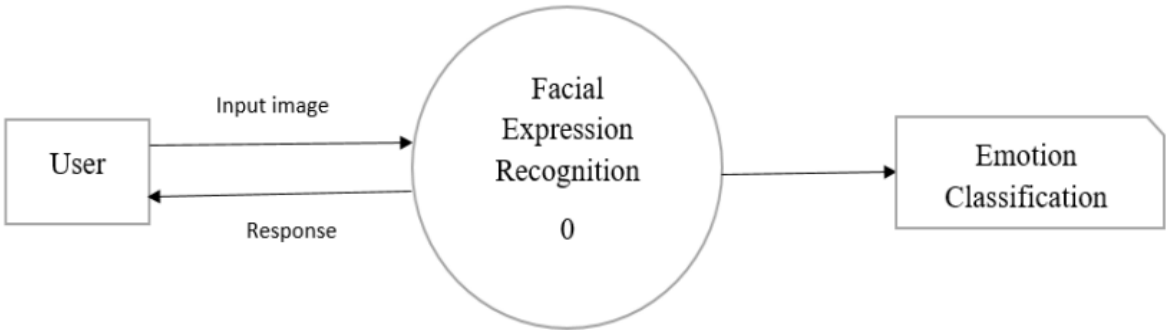
**DFD Level 1** provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

**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.

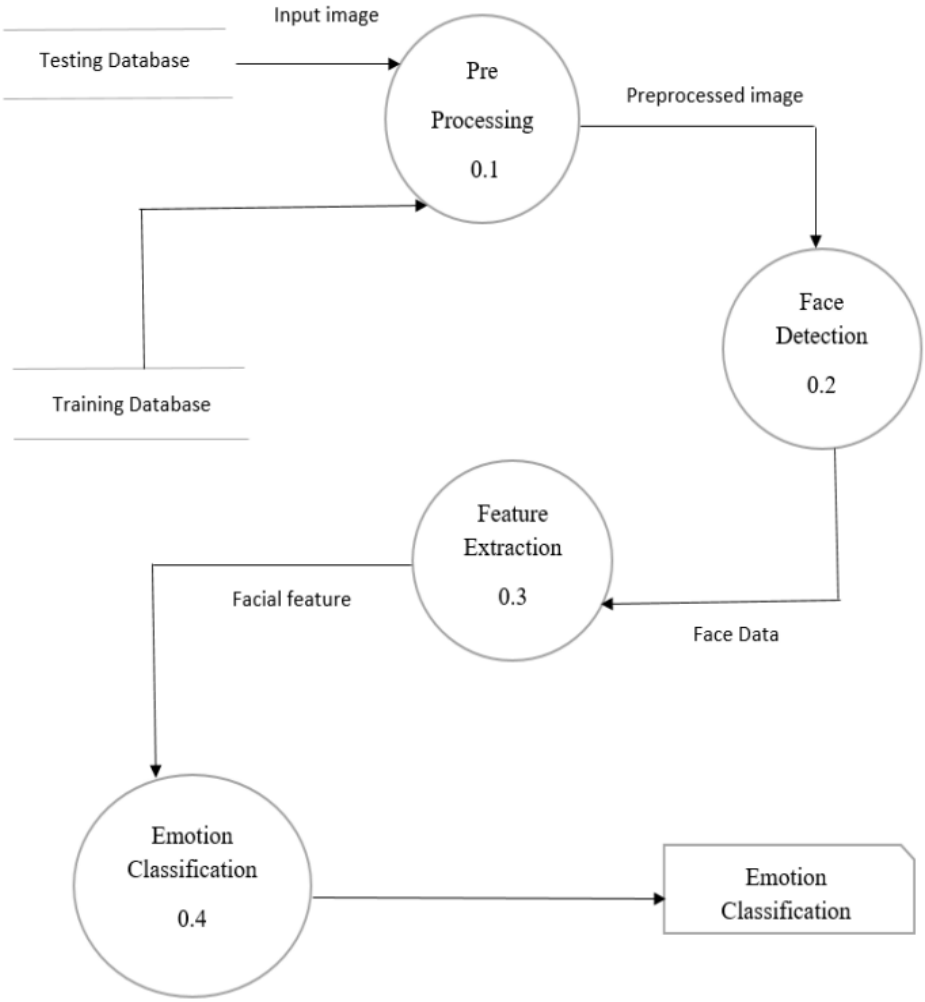
Progression to Levels 3, 4 and beyond is possible, but going beyond Level 3 is uncommon. Doing so can create complexity that makes it difficult to communicate, compare or model effectively.

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 DFD**

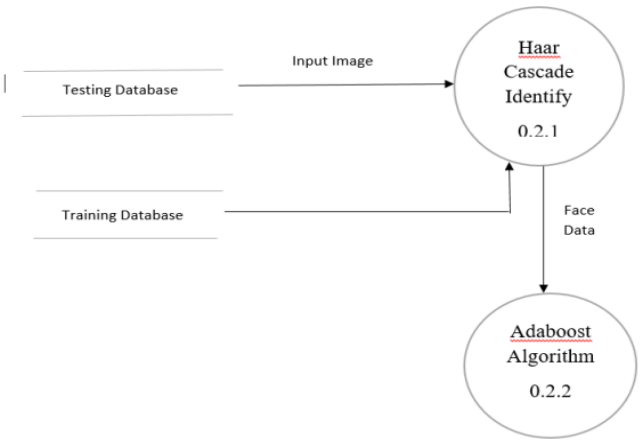


## Level 1 DFD

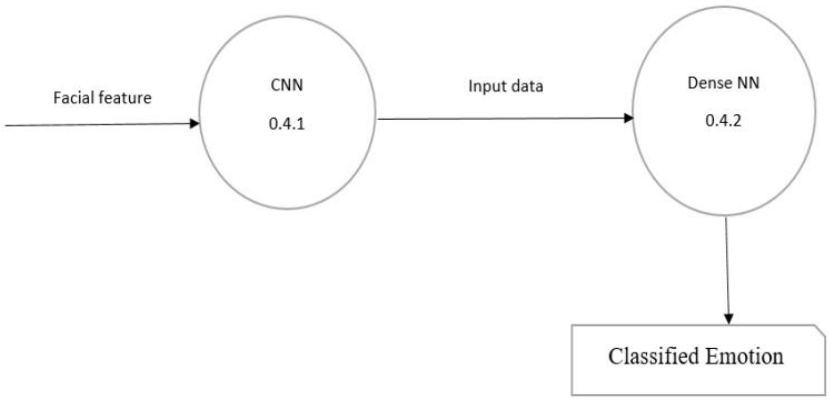



## Level 2

Face Detection-



Emotion Classification-




## 2.2 Feasibility Study

A feasibility study is a high-level capsule version of the entire System analysis and design Process. The study begins by classifying the problem definition. Feasibility is to determine if it's worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analysed carefully. There are 3 parts in feasibility study.

### Technical Study

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs, and procedures. This can be qualified in terms of volume of data, trends, frequency of updating inorder to introduce the technical system. The application is the fact that it has been developed on windows 10 platform and a high configuration of 8 GB RAM on Intel Pentium Dual core processor. This is technically feasible. The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

### Operational Study

Operational feasibility is the measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system

development. The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives about development schedule, delivery date, corporate culture and existing business processes. To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability, and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realized. A system design and development require appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters.

A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

### 2.3 Behavioural Study

Establishing the cost-effectiveness of the proposed system i.e., if the benefits do not outweigh the costs, then it is not worth going ahead. In the fast-paced world today there is a great need of online social networking facilities. Thus, the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

# Chapter 3

## 3.1 Flow Chart

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Basic Symbols used in Flowchart Designs-

- **Terminal:** The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.
- **Input/Output**: A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.
- **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.
- **Decision:** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.
- **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.
- **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

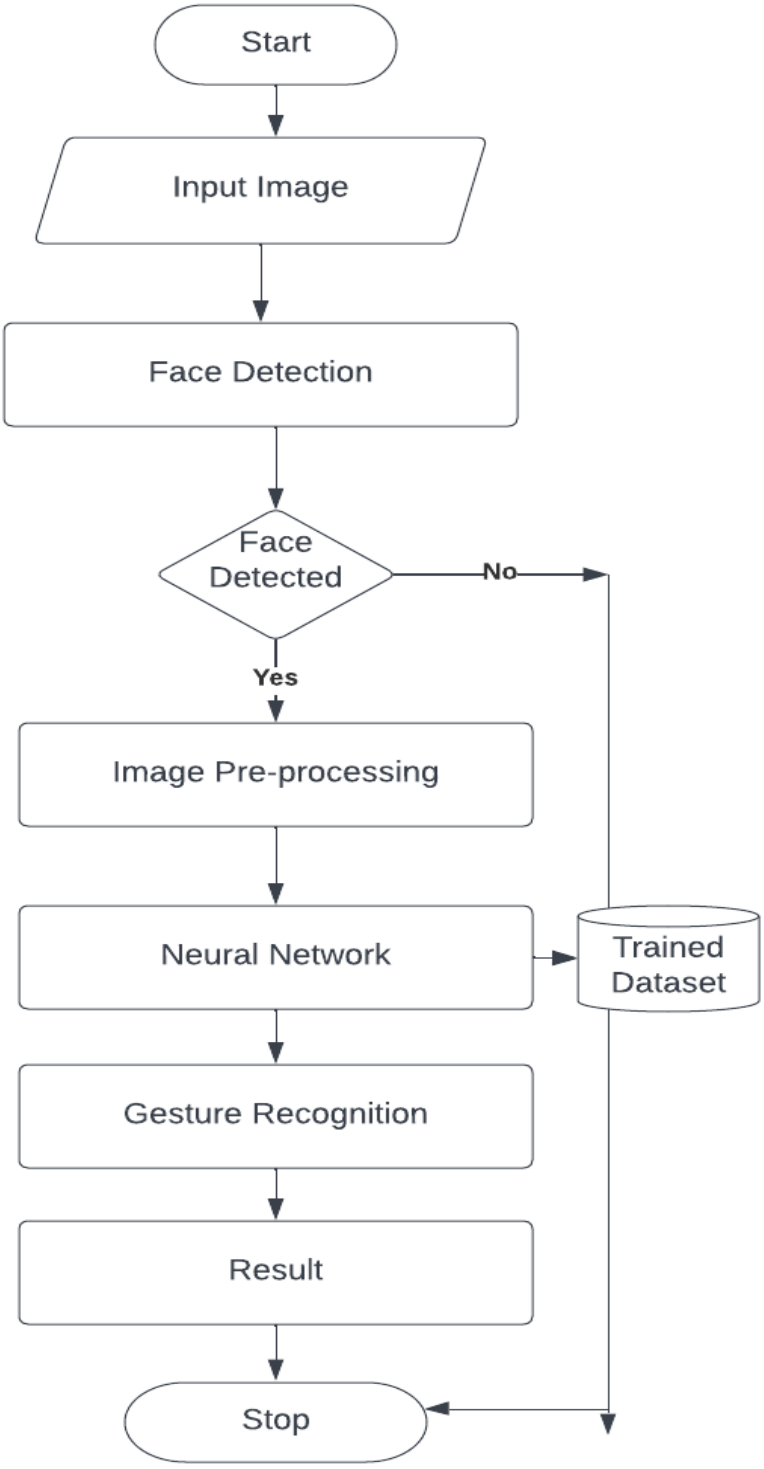



**Figure 3.1 Flowchart**

### 3.2 Use Case Diagram

In the Unified Modelling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve

**Use case Diagram Components**

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.

- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.

- **Goals:** The result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

**Use case diagram symbols and notation**

The notation for a use case diagram is straightforward and doesn't involve as many types of symbols as other UML diagrams.

**Use cases**

Horizontally shaped ovals that represent the different uses that a user might have.

- **Actors:** Stick figures that represent the people employing the use cases.

- **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

- **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.

- **Packages:** A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

# Chapter 4

# Design and Implementation

## 4.1 Methodology

The facial expression recognition system is implemented using convolutional neural network. The block diagram of the system is shown in following figures:


**Figure 4.1- Training Phase**

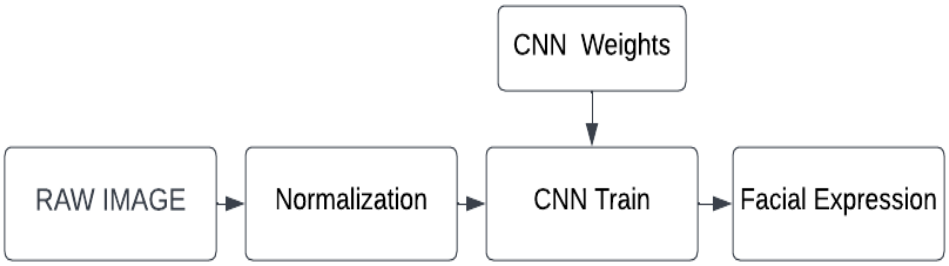



**Figure 4.2 - Testing Phase**

During training, the system received a training data comprising grayscale images of faces with their respective expression label and learns a set of weights for the network. The training step took as input an image with a face. Thereafter, an intensity normalization is applied to the image. The normalized images are used to train the Convolutional Network. To ensure that the training performance is not affected by the order of presentation of the examples, validation dataset is used to choose the final best set of weights out of a set of trainings performed with samples presented in different orders. The output of the training step is a set of weights that achieve the best result with the training data. During test, the system received a grayscale image of a face from test dataset and output the predicted expression by using the final network weights learned during training. Its output is a single number that represents one of the seven basic expressions.

## 4.2 The Library & Packages:

**Python:** Python is a powerful scripting language and is very useful for solving statistical problems involving machine learning algorithms. It has various utility functions which help in pre-processing. Processing is fast and it is supported on almost all platforms. Integration

with C++ and other image libraries is very easy, and it has in-built functions and libraries to store and manipulate data of all types. It provides the pandas and numpy framework which helps in manipulation of data as per our need. A good feature set can be created using the numpy arrays which can have n-dimensional data.

**Dataset -** The dataset from a Kaggle Facial Expression Recognition Challenge (FER2013) is used for the training and testing. It comprises pre-cropped, 48-by-48-pixel grayscale images of faces each labelled with one of the 7 emotion classes: **anger, disgust, fear, happiness, sadness, surprise, and neutral**. Dataset has training set of 35887 facial images with facial expression labels. The dataset has class imbalance issue, since some classes have large number of examples while some has few. The dataset is balanced using oversampling, by increasing numbers in minority classes. The balanced dataset contains 40263 images, from which 29263 images are used for training, 6000 images are used for testing, and 5000 images are used for validation.

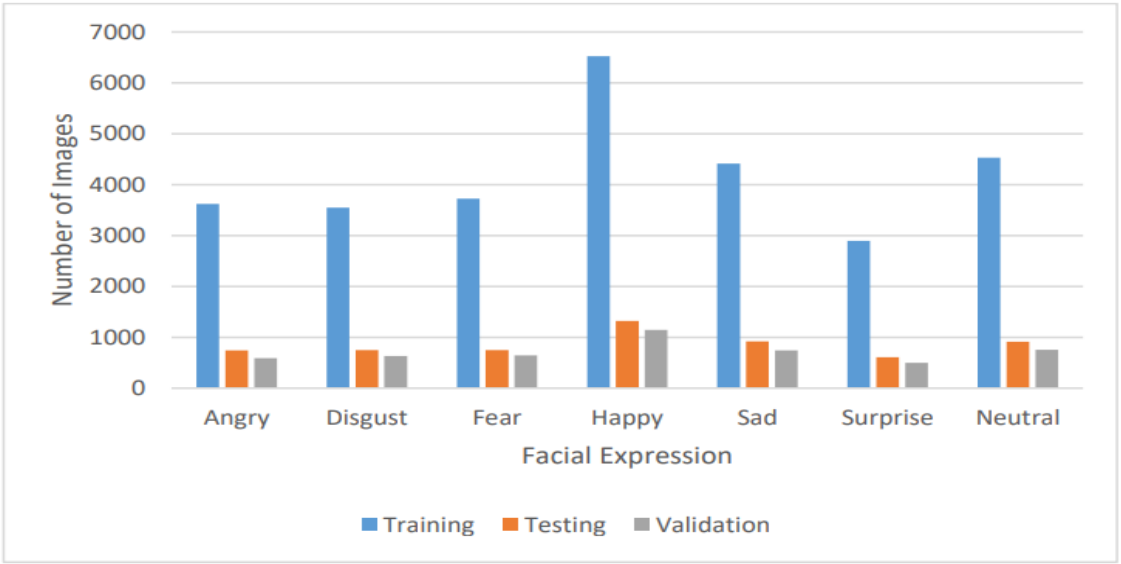



**Figure 4.3 - Training, Testing and Validation Data distribution**

**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 neighbour 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.

**Karas -**

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 such as 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 of TensorFlow.

**Guiding Principles-**

- User Friendliness: Keras is an API designed for human beings, not machines. It puts user experience front and centre. 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.

## 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 a smaller 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!!!

## 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.

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.

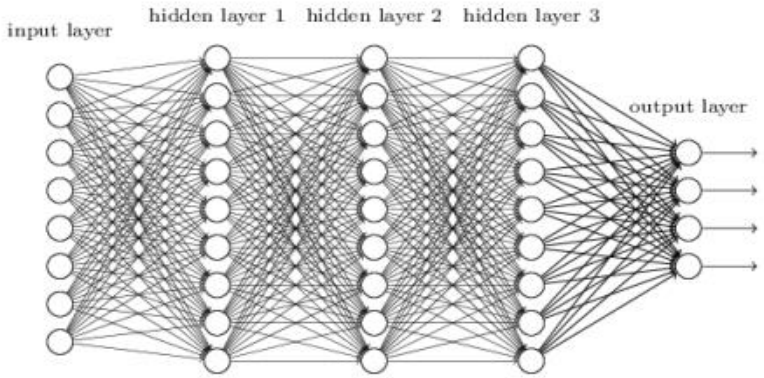
## Deep Convolutional Neural Networks (DCNN) –

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.




- **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 neighbourhood 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 neighbourhood 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.

### 4.3 Architecture of CNN

A typical architecture of a convolutional neural network contains an input layer, some convolutional layers, some fully connected layers, and an output layer. CNN is designed with some modification on LeNet Architecture [10]. It has 6 layers without considering input and output. The architecture of the Convolution Neural Network used in the project is shown in the following figure.

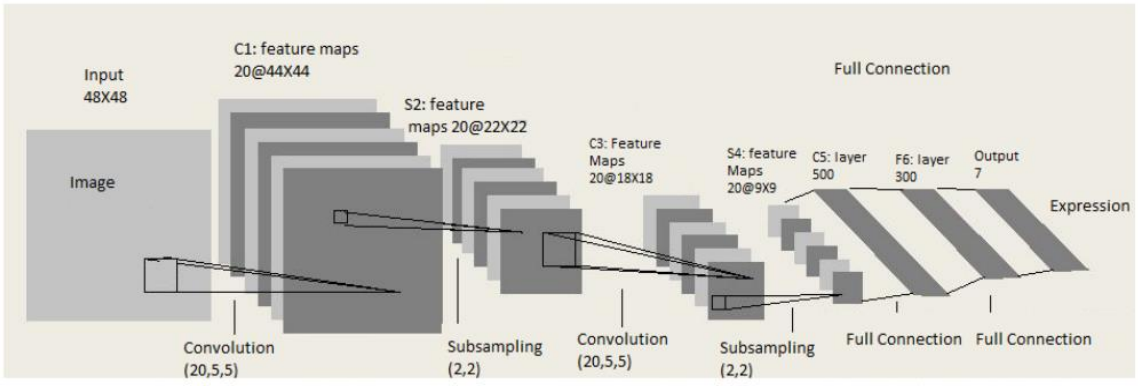



**Figure 3.2: Architecture of CNN (Modified from LeNet Architecture)**

**(i). 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. Normalized grey scale images of size 48 X 48 pixels from Kaggle dataset are used for training, validation and testing. For testing propose laptop webcam images are also used, in which face is detected and cropped using OpenCV Haar-Cascade Classifier and normalized.

**(ii). Convolution and Pooling (ConvPool) Layers –**

Convolution and pooling are done based on batch processing. Each batch has N images and CNN filter weights are updated on those batches. Each convolution layer takes image batch input of four-dimension N x Color-Channel x width x height. Feature map or filter for convolution are also four dimensional (Number of feature maps in, number of feature maps out, filter width, filter height). In each convolution layer, four-dimensional convolution is calculated between image batch and feature maps. After convolution only parameter that change is image width and height.

New image width = old image width – filter width + 1

New image height = old image height – filter height + 1

After each convolution layer down sampling / subsampling is done for dimensionality reduction. This process is called Pooling. Max pooling and Average Pooling are two famous pooling method. In this project max pooling is done after convolution. Pool size of (2x2) is

taken, which splits the image into grid of blocks each of size 2x2 and takes maximum of 4 pixels. After pooling only height and width are affected.

Two convolution layer and pooling layer are used in the architecture. At first convolution layer size of input image batch is Nx1x48x48. Here, size of image batch is N, number of color channel is 1 and both image height and width are 48 pixel. Convolution with feature map of 1x20x5x5 results image batch is of size Nx20x44x44. After convolution pooling is done with pool size of 2x2, which results image batch of size Nx20x22x22. This is followed by second convolution layer with feature map of 20x20x5x5, which results image batch of size Nx20x18x18. This is followed by pooling layer with pool size 2x2, which results image batch of size Nx20x9x9.

**(iii). Fully Connected Layer**

This layer is inspired by the way neurons transmit signals through the brain. It takes many input features and transform features through layers connected with trainable weights. Two hidden layers of size 500 and 300 unit are used in fully connected layer. The weights of these layers 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. Hyper-parameters for this layer include learning rate, momentum, regularization parameter, and decay.

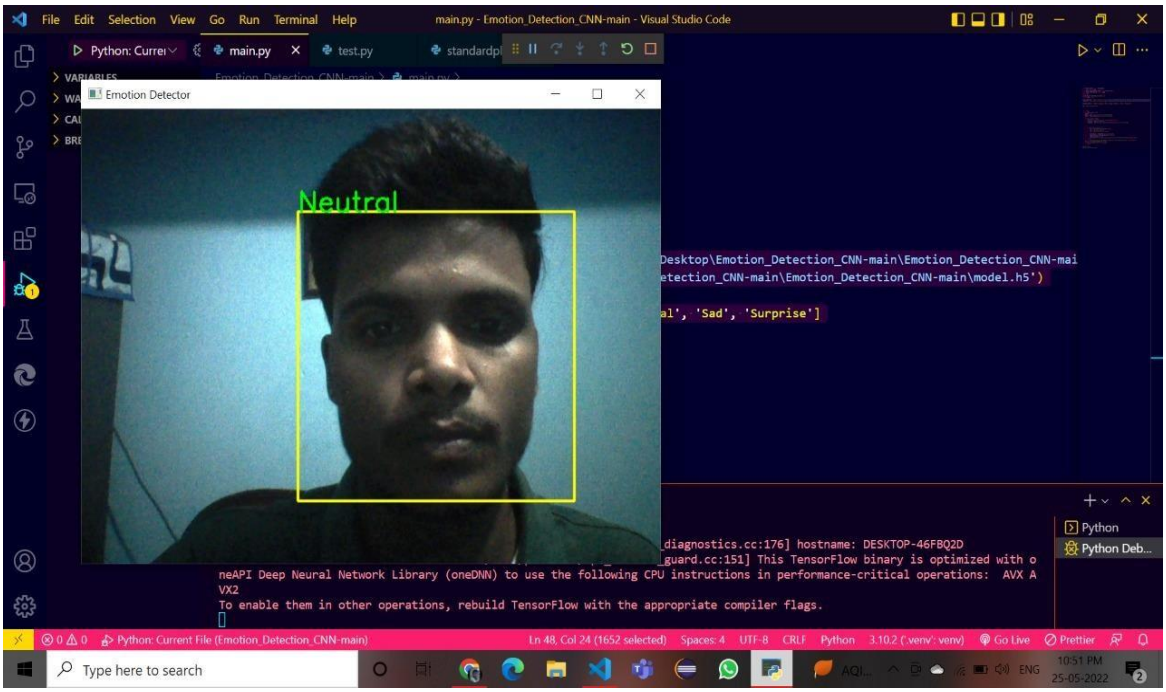
**(iv). Output Layer –**

Output from the second hidden layer is connected to output layer having seven distinct classes. Using SoftMax activation function, output is obtained using the probabilities for each of the seven class. The class with the highest probability is the predicted class.

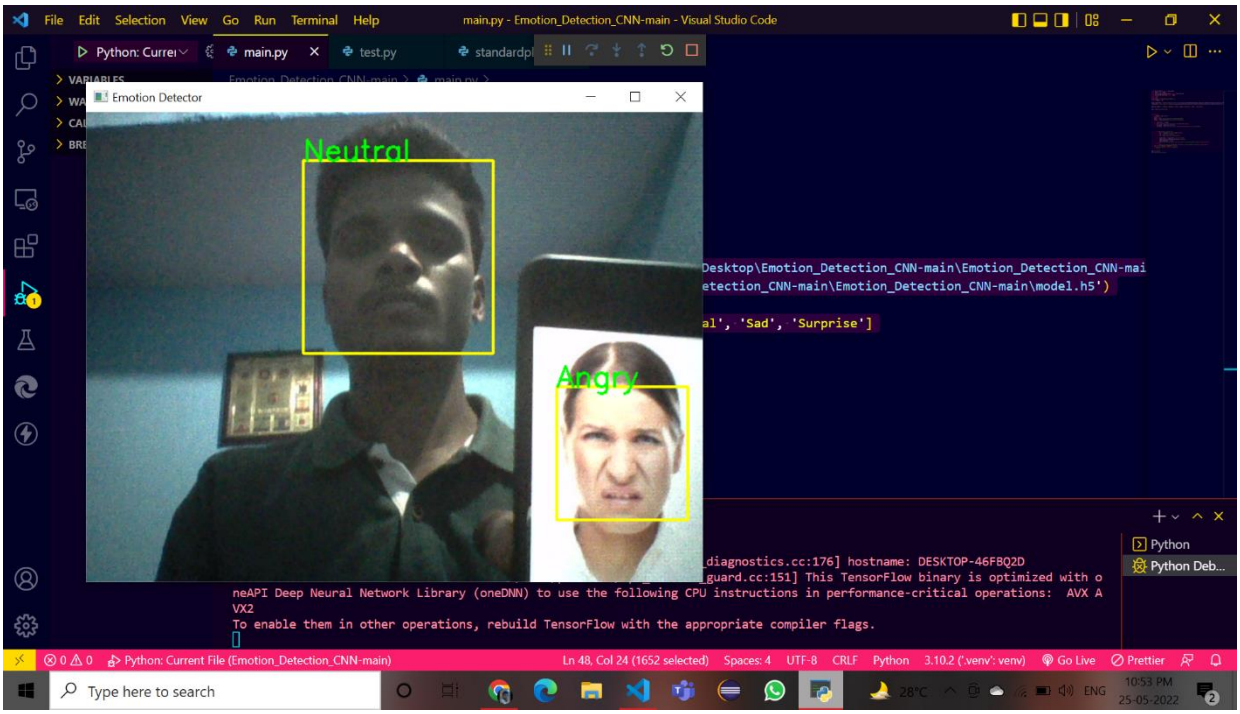
**Model Validation –**

Performance As it turns out, the final CNN had a validation accuracy of 58%. This makes a lot of sense. Because our expressions usually consist of 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.

## 4.4 Output screenshots-




**4.1 Output screen 1 with Neutral emotion**




**4.2 Output screen 2 with multiple faces (Neutral and Angry emotion)**

# Chapter 5

## Coding

### 5.1: Main Source Code

**main.py file –**

```
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import numpy as np

face_classifier = cv2.CascadeClassifier(r'C:\Users\singh\Desktop\Emotion_Detection_CNN-main\Emotion_Detection_CNN-main\haarcascade_frontalface_default.xml')
classifier =load_model(r'C:\Users\singh\Desktop\Emotion_Detection_CNN-main\Emotion_Detection_CNN-main\model.h5')

emotion_labels = ['Angry','Disgust','Fear','Happy','Neutral', 'Sad', 'Surprise']

cap = cv2.VideoCapture(0)



while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)



        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            prediction = classifier.predict(roi)[0]
```

```
            label=emotion_labels[prediction.argmax()]
            label_position = (x,y)
            cv2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
        else:
            cv2.putText(frame,'No Faces',(30,80),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv2.imshow('Emotion Detector',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

**5.2 Dataset file –**

We have used dataset which downloaded from Kaggle- [Kaggle Dataset Download link]

# Chapter 6

# Testing

## 6.1 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

### 6.1.2 Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

**1. Find problems early:** Unit testing finds problems early in the development cycle. In test- driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

**2.Facilitates Change:** Unit testing allows the programmer to refactor code or upgrade system libraries later, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

**3. Simplifies Integration:** Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

**4. Documentation:** Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API).Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviours that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

## 6.2 Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

**(i) Purpose**

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested, and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross dependencies for software integration testing are schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatic complexity of the software and software architecture, reusability of modules and life cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns [2] are collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high frequency integration.

**(a)Big Bang**

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing. A type of big-bang integration testing is called "usage model testing" which can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use. Usage Model testing takes an optimistic approach to testing because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment. For integration testing, Usage Model testing can be more efficient and provides better test coverage than traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

**(b)Top-down And Bottom-up**

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level components. The process is repeated until the component at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. Top-down testing is an approach to integrated testing where the top integrated modules are tested, and the branch of the module is tested step by step until the end of the related module. Sandwich testing is an approach to combine top-down testing with bottom-up testing.

## 6.3: Software Verification and Validation

### (i) Introduction

In software project management, software testing, and software engineering, verification, and validation (V&V) is the process of checking that a software system meets specifications and that it fulfils its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

- Validation: Are we building the right product?
- Verification: Are we building the product right?

According to the Capability Maturity Model (CMMI-SW v1.1)

**Software Verification:** The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

**Software Validation:** The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right". Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfil its intended use.

From Testing Perspective

- Fault – wrong or missing function in the code.
- Failure – the manifestation of a fault during execution.
- Malfunction – according to its specification the system does not meet its specified functionality

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required. Within the modelling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

- M&S Verification is the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.
- M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).

**(ii) Classification of Methods**

In mission-critical software systems, where flawless performance is necessary, formal methods may be used to ensure the correct operation of a system. However, often for non-mission-critical software systems, formal methods prove to be very costly, and an alternative method of software V&V must be sought out. In such cases, syntactic methods are often used.

**(iii) Test Cases**

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Other methods, such as reviews, may be used early in the life cycle to provide for software validation.

**(iv) Black-Box Testing**

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system, and acceptance. It typically comprises most if not all higher-level testing but can also dominate unit testing as well.

**6.4 Test Procedures**

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

**Test Cases**

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

## 6.5: White-Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e., black-box testing). In white box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT). White-box testing can be applied at the unit, integration, and system levels of the software testing process. Although traditional testers tended to think of white box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

### 6.5.1 Levels

**1) Unit testing:** White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White box testing during unit testing catches any defects early on and aids in any defects that happen later after the code is integrated with the rest of the application and therefore prevents any type of errors later.

**2) Integration testing:** White box testing at this level is written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white box testing for any interactions of interfaces that are known to the programmer.

**3) Regression testing:** White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

### 6.5.2 Procedures

White-box testing's basic procedures involves the tester having a deep level of understanding of the source code being tested. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created.

These are the three basic steps that white-box testing takes in order to create test cases:

- Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white box testing to layout all the basic information.
- Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
- Output involves preparing final report that encompasses all the above preparations and results.

### 6.5.3 Advantages

White-box testing is one of the two biggest testing methodologies used today. It has several major advantages:

- Side effects of having the knowledge of the source code is beneficial to thorough testing**.**
- Optimization of code by revealing hidden errors and being able to remove these possible defects.
- Gives the programmer introspection because developers carefully describe any new implementation**.**
- Provides traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.
- White box testing give clear, engineering-based, rules for when to stop testing.

### 6.5.5 Disadvantages

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:

- White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high level of knowledge due to the complexity of the level of testing that needs to be done.
- On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.
- The tests focus on the software as it exists, and missing functionality may not be discovered.

### 6.6 System Testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

### 6.7: Test Cases

| S.NO. | INPUT | ANGRY | HAPPY | NEUTRAL | SAD | SURPRISE | OUTPUT |
|---|---|---|---|---|---|---|---|
| 1 | SITTING IN FRONT OF CAMERA | 0.14683995 | 0.16525857 | 0.21751054 | 0.44684792 | 0.02354303 | SAD |
| 2 | SITTING IN FRONT | 0.085875 | 0.5673613 | 0.16948242 | 0.161315919 | 0.0141221 | HAPPY |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | OF CAMERA | | | | | | |
| **3** | SITTING IN FRONT OF CAMERA | 0.64683995 | 0.16525857 | 0.11751054 | 0.14684792 | 0.02354303 | ANGRY |
| **4** | SITTING IN FRONT OF CAMERA | 0.085875 | 0.5673613 | 0.16948242 | 0.161315919 | 0.0141221 | HAPPY |
| **5** | SITTING IN FRONT OF CAMERA | 0.14683995 | 0.16525857 | 0.21751054 | 0.44684792 | 0.02354303 | SAD |
| **6** | SITTING IN FRONT OF CAMERA | 0.14683995 | 0.16525857 | 0.41751054 | 0.14684792 | 0.02354303 | NEUTRAL |

## 7.1 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 behavioural 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 behavioural 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.

## 7.2 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 become 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.

## Bibliography


- https://pypi.org/project/keras/
- https://numpy.org/
- https://www.wisegeek.com/what-isemotionalintelligence.htm