ST. XAVIER'S COLLEGE [AUTONOMOUS], KOLKATA

FACE EMOTION RECOGNITION USING CNN

by

Angana Banerjee (Roll 521),

Shreya Banerjee (Roll 520) and

Pratyusha Mitra (Roll 554)

Under Guidance of

Prof. Shalabh Agarwal

Submitted to the Department of Computer Science in partial fulfillment of the requirements for the degree of M.Sc.

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled **FACE EMOTION RECOGNITION USING CNN** submitted to the Department of Computer Science, ST. XAVIER'S COLLEGE (AUTONOMOUS), KOLKATA, in partial fulfillment of the requirement for the award of the degree of Master of Science (M.Sc) is an original work carried out jointly by **Angana Banerjee, Rgtn no :A01-2112-0786-16, Shreya Banerjee, Rgtn no: A01-2112-0754-16, Pratyusha Mitra, Rgtn no: A01-2112-0046-19** under my guidance . The matter embodied in this project is authentic and is genuine work done by the students and has not been submitted whether to this college or to any other Institute for the fulfillment of the requirement of any course of study.

Signature of the Professor

Date :

Name and Address of the Professor:

ACKNOWLEDGEMENT

We are thankful to our project mentor, Mr. Shalabh Agarwal, for guiding us through the whole procedure of documenting our project work. We thank the college authorities as well as the Head of the Department, Mr. Romit Beed for organising and conducting projects for all final year students even in the midst of the Covid-19 Pandemic.

We generously thank all the writers, publishers and personnels in the related fields for sharing their knowledge with us. As far as possible, all the related papers are cited in the Bibliography section.

Lastly, we are proud to convey that all of the team members have participated explicitly and aimously to make this project a success. The participation and contribution of each team member was absolutely vital. We thank everyone for their kindness and support.

ROLES AND RESPONSIBILITIES FORM

Name of the Project : Facial Emotion Recognition using CNN Date: 02-01-2021

Name of the Team Member	Role	Tasks and Responsibilities
Angana Banerjee	Project Leader	Coding, Formatting,Testing
Shreya Banerjee	Project Member	Dataset Management,Testing
Pratyusha Mitra	Project Member	Documentation, Testing

Name and Signature of the Project Team members:

Name **Signature**

1. Angana Banerjee ……..

2. Shreya Banerjee ……

3. Pratyusha Mitra ……

Signature of the Professor: …… Date: …….

CONTENTS

CHAPTER 1

INTRODUCTION

Abstract: Emotions have been the greatest mystery in the field of machines. Like intelligence, there are no clear borderline definitions of emotion. We, as human beings, have developed the intuition to recognise the emotion in a given situation based on many different parameters, example colors, lights, contrasts, objects, people, clothes, and even absence of some of these parameters. Also, emotions, like intelligence, vary from person to person. Based on past experiences, different people might associate the same contexts with different emotions. Understanding emotion, thus, in a given context is challenging.
Everything has a start. The way we learnt about human intelligence and implemented it in artificial intelligence, we are taking steps to unveil the mysterious cover of emotions as well. Emotion recognition can be considered as a very first step. In our work, we are creating a model and training it on a dataset containing faces with various emotions. The model is supposed to learn the way a face looks when it is happy, sad, fearful, disgusted, surprised or just neutral. This project, however, might be extended to recognise emotion in a frame of reference where no face is present. Such a model would be multi labelled and would decide the emotions based on surrounding objects and a few parameters discussed above.

1.1. Background

Our Project involves two main components -

1) Convolution neural network (For recognizing emotions)
2) Haarcascades classifier (For recognizing frontal faces)

Convolution Neural Network: -

CNN is a class of deep learning neural networks that are most suitable for processing images. They are modelled after the visual cortex in some mammals, like cats, where individual optical neurons respond to light only in a restricted portion of the receptive field. The receptive fields overlap to form the entire visual range. In CNN, individual neurons learn from some restricted region in the image. The restricted regions, in turn, overlap each other to create connectivity in the entire image.
CNN architecture, like neural networks, consists of input, hidden and output layers. The hidden layers comprise convolutional layers, pooling layers and fully connected layers. A brief structure of a basic CNN model is given as fig 1.1.

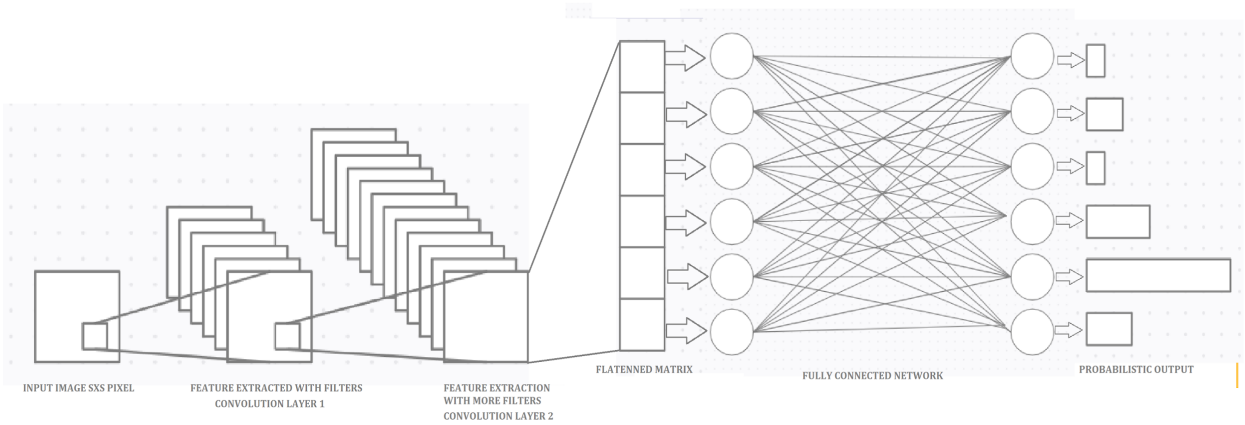


Fig 1.1 Typical architecture of CNN

CNN is exceptionally good at extracting features from an image and learning those features on its own. In our project, we are using a CNN model to recognise faces with emotions. We have used a dataset containing a large number of front profiles of faces with different emotions. The CNN model is supposed to learn to recognise faces and display the emotions.
The CNN, like any other deep learning model, consists of 2 stages, Forward Propagation and Backward Propagation.

1. *Forward Propagation*: the model inputs image, extract features, processes an output.
2. *Backward Propagation*: the loss is computed and fed back into the network to correct parameters in the network.

The stages are described in greater detail in the following sections.

Forward Propagation:

At first, the input is fed into the neural network. The input image can be considered as a matrix of pixels where pixel value represents the color and intensity of the image. We generally convert color images into grayscale so that the value of each cell in the image matrix contains the luminosity of the corresponding image.

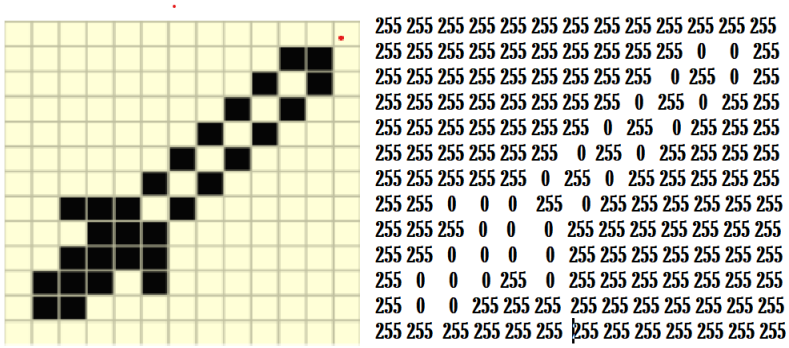


Fig 1.2 Pixel to matrix conversion

Each cell of the matrix is fed into a single neuron in the input layer. Hence, for an image of size m$pixels$×m$pixels$, the number of neurons in the input layer N_i is given by:

$$N_i = m \times m \quad \ldots(i)$$

After the input is fed into the network, the feature extraction is done in the convolution step. In this step, a filter is applied on the matrix to generate a convolution matrix for a particular feature. The Kernel size of the convolution layers defines the number of filters applied on the image matrix. The convolution process can be defined as:

$$M_{output} = M_{input} * f_i \quad \ldots(ii)$$

Where M_{input} is the image matrix, f_i is the ith filter, and M_{output} is the convolution matrix. The filter is essentially a smaller matrix (say, of size f×f) that is multiplied cell-wise to a submatrix (size f×f) of M_{input} to generate an intermediate matrix(size f×f). The cells of the intermediate matrix are added cell wise to obtain the value of the corresponding cell in M_{output}. The dimension of M_{output} is thus (m-f+1)×(m-f+1) for a single filter The process can be visualized as below.

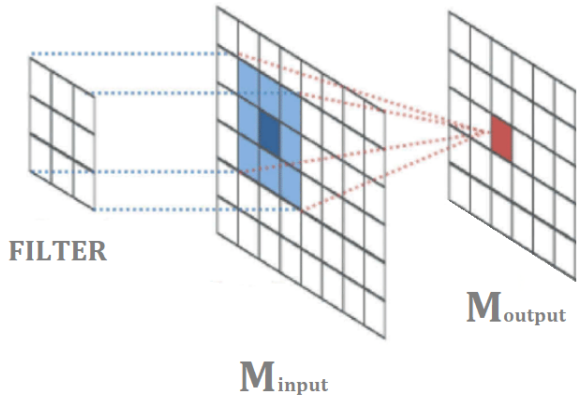


Fig 1.3 Convolution matrix generation with the help of 3 x 3 filter

In the next step, the convolution matrix passes through the fully connected layers, which essentially share similar architecture with traditional neural networks. Here, the convoluted matrix is first flattened to form a 1D matrix which is fed into the network. The fully connected layer adjusts weights at each node to generate an output. Initially, the weights and biases are set to random values, which are readjusted based on the backpropagation parameters. The fully connected layer performs two operations on the incoming data – a linear transformation and a non-linear transformation. The linear transformation function can be defined as:

$$Y = W \times X + B \quad \ldots(iii)$$

Where Y is the output matrix, X is the flattened convolution matrix, W is the weight matrix and B is the matrix containing bias values. This step is very similar to linear regression.

The non-linear transformation is brought about by the activation function. The purpose of the activation function is to determine which nodes will be activated during a particular operation, in other words, discard unimportant noise from image data. This nonlinearity helps the model to capture and learn complex relations and patterns in image data.

In our project, we are using ReLu (Rectified Linear Unit) activation functions for convolution layers. For the output layer, we are using the Softmax Activation function that is used for categorization of data.

The fully connected network can be visualized as follows.

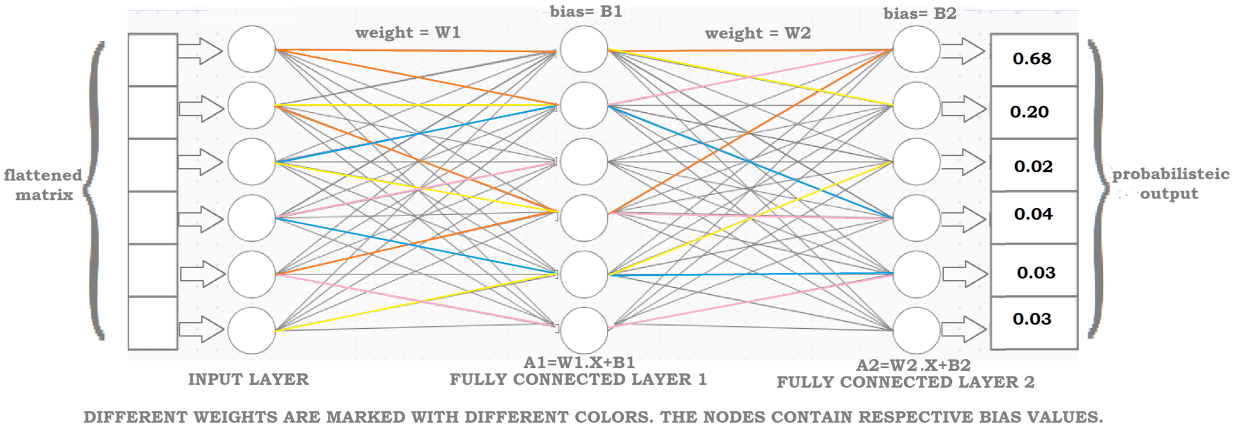


Fig 1.4 Fully Connected Layers in CNN. Here, both layers have equal numbers of neurons.

Rectified Linear Unit

ReLU, also called Rectified Linear Unit, is a function that is interested only in the positive values of parameters. A neuron is activated only if its weight with respect to the parameter is positive. The function can be visualised in fig 1.5.

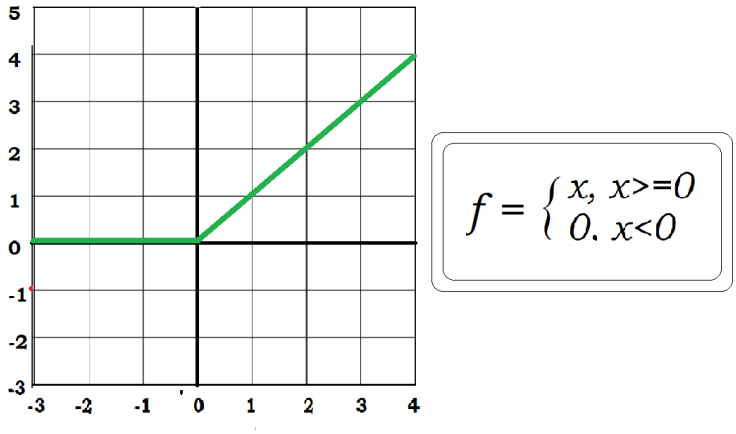


Fig 1.5 Rectified Linear Unit (ReLU)

Softmax Activation Function

The outputs produced by the output layer in the fully connected networks are often numbers, rather than probabilities. The function of softmax is to squash these outputs into a range of 0 to 1, which can be interpreted as class score or the probability of x to belong to a single class. Based on this, the highest class probability value will be taken to be the category of the given input, x. Softmax function is discussed further in the backpropagation section. It can be visualised in figure 1.6.

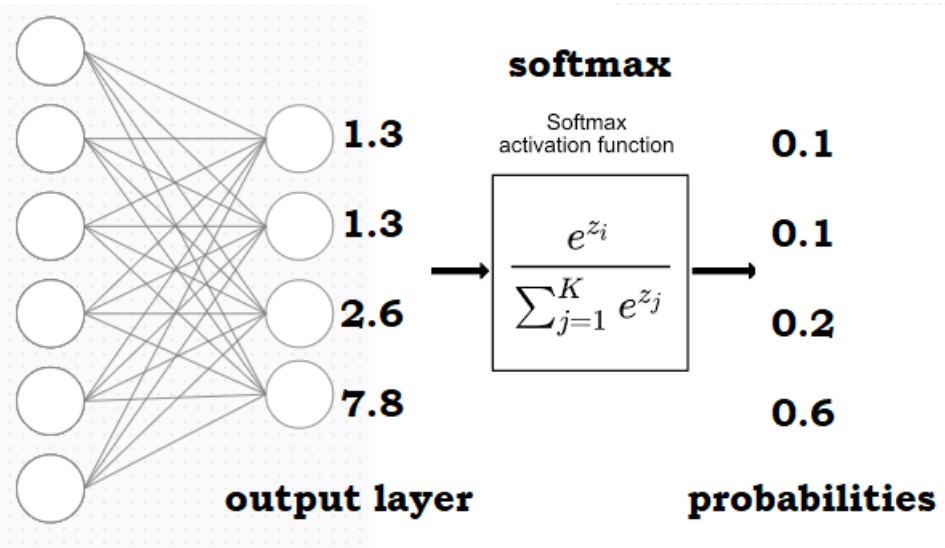


Fig 1.6 Softmax Activation Function

Backward Propagation:

The backward propagation in CNN is what decides the values in the filter matrices among other things. In this stage, the model rectifies itself by adjusting filters, weights and biases to generate more accurate overall prediction. The steps in backward propagation consists of calculating loss and optimizing loss function for updation of weights and biases in fully connected networks, and updating filters in convolutional layers.

Calculation of loss (Categorical cross entropy)

Every supervised deep learning model, having weight matrix, W, penalizes itself based on computed loss. The loss computed at the end of each training instance with parameter matrix, X can be defined as some function, L, given by:

$$Loss = L(X, W, y) \quad \ldots(iv)$$

where y is the model-predicted outcome of instance X.

Our project, being an example of multi-class classification problem, is concerned with finding the categorical cross entropy loss for our data.

Categorical cross entropy can be thought of as successive implementation of two loss functions, softmax loss and cross entropy loss.

Softmax loss is nothing more than the softmax activation function used earlier. It is interpreted as the class probabilities for an element and defined as follows:

$$s(s)_i = \frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}} \quad \ldots(v)$$

Where C is the set of classes and s is the output score, s_i is output score for i^{th} class, s_j is the loss inferred for all other classes. The softmax loss for a particular class is dependent on loss of all classes.

The cross entropy loss is also called logistic loss. The loss function can be defined as

$$CE = -\sum_{i=1}^{C} f(x_i).log(s_i) \quad \ldots(vi)$$

Where f(i) is the original output and s_i is the CNN score for the i^{th} element of the training dataset(x_i) and s_i is the CNN score for the same.

The categorical cross entropy function is an extension of cross entropy function, which works on the class probabilities of the classes in C. Thus, the categorical cross entropy can be defined as:

$$CCE = -\sum_{i=1}^{C} f(s_i).log(s(s)_i) \quad \ldots(vii)$$

In multi-class classification, the labels are one-hot, meaning that the output of the correct class should be 1, the rest 0. So, $f(s_c)=1$ for the correct class, c, whereas for $j \neq c$, $f(s_j)=0$. Therefore, only the correct class keeps its loss; the other loss terms are reduced to 0 and discarded. Hence, the final loss function can be summarized as:

$$CCE_{multiclass} = -\log\left(\frac{e^{s_c}}{\sum_{j=1}^{C} e^{s_j}}\right) \quad \ldots(viii)$$

where s_c is the score of correct class.

Now, as the loss is defined, our target would be to optimize the loss and in turn, back propagate and tune the parameters in the CNN network. For optimization, we are using Adam optimizer (discussed in the following section)

Optimization

In CNN's fully connected layer, optimization generally requires the model to find values of weights and biases for the edges connecting the nodes in the successive layers so that the loss function is minimized. Hence, the objective of the optimization function is to minimize the loss function and tune the parameters accordingly.

For a loss function, L(x), when plotted on a graph, usually takes the following shape. fig(1.7)

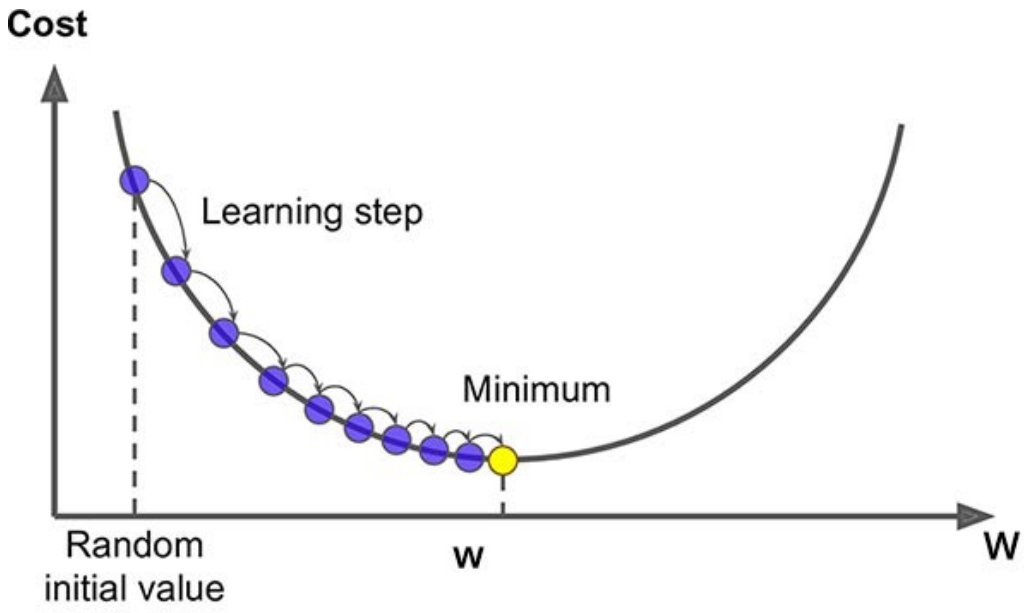


Fig1.7[8]. Gradient descent figure

To find the minima of the depicted curve, we usually take the derivative of the loss function and try to equate it with 0 to obtain gradient, ∇ . This process is called gradient descent. Based on this initial parameter, P_{old} and a constant, called learning rate, lr, the new parameters, P_{new} can be calculated using the following formula.

$$P_{new} = P_{old} - (lr \times \nabla) \quad \ldots(ix)$$

For a huge dataset, computing gradient descent can be time consuming. To overcome this problem, we use stochastic gradient descent, in which only a subset of the dataset is used to generate new parameters in a single step. Stochastic gradient descent reduces the time to compute gradient descent. The problem, however, increases in case of multivariate problems for a large dataset.

To visualize this problem, we might consider the following diagram, fig 1.8, plotting 2 parameters along the x- y plane, the outcome is plotted along the z axis. The plane can be seen to have certain bumps and valleys, usually called ravines. Ravines usually have steeper slope and stochastic gradient descent cannot navigate it properly. If a ball is rolled starting from a random point, which follows the classical stochastic gradient descent approach, then it is quite likely that the ball might be trapped in some local minima. To reach the global minima, the ball might need to display a momentum, or a direction away from the local minima towards the global one.

This can be achieved by what is called a stochastic gradient descent with momentum.

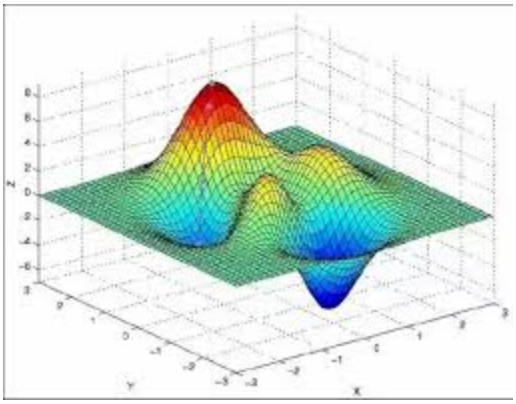


Fig. 1.8[9]- stochastic plane

Stochastic gradient descent with momentum can be described in the following manner. We use a hyper parameter, β, within range [0,1] such that the function will be evaluated over an average of last 1/1-β values in data sequences. The weight W can be updated using the momentum, V_t and learning rate, lr. The momentum, V_t can be thought of to be the moving average of gradients, ∇_w obtained for the loss function Loss(W, X, y) of a net with weight matrix W on data X and outcome y . The process can be formulated as follows.

$$V_t = \beta V_{t-1} + lr \nabla_w Loss(W, X, y)$$

$$W_{adjusted} = W_{unadjusted} - V_t \quad \ldots(x)$$

The introduction of momentum in stochastic gradient descent helps in accelerating the convergence of the global minimization function.

The learning rate, lr, of stochastic decent changes dynamically with each time step. This can decay with a polynomial, exponential or even a constant rate for successive time steps.
An advancement to this algorithm is Adam[2], which we have used in our project. Adam implements different decay rates for lr for different parameters in the data. In other words, the decay rate of lr for parameter x_1 can be polynomial whereas that of x_2 can be exponential. The algorithm for the Adam optimization as provided in [2] is given below.

Require: α: Step size
Require: $\beta 1, \beta 2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: f(θ): Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize time step)
while θ_t not converged do
$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at time step t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_3^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t

The idea to be noted here is the updation of parametersθ in the stochastic function. g_t is the moving gradient calculated on parameters θ in timestep t. The first moment estimate is the mean of the g_t while the second momentum estimate is the uncentered variance. Although both of them are initialized to a vector of zeros, the correction steps counteract the error in the beginning of the dataset. The effective step size is bounded above by learning rate, α. Thus, the effective step size tends to zero as we tend to reach the optimum. This helps in estimating the magnitude of α for the next step size and it can compute the same with only a few iterations.
The main advantages of Adam include:

1. Requirement of less memory.
2. Requires only first order gradient.
3. Magnitude of parameter updates are independent of gradient rescaling
4. It naturally forms a step size annealing.

We have used Adam to update weights and biases in the fully connected layers in CNN.

Filter adjustment in convolution layer

As stated earlier, the convolution process can be formulated as follows.

$$M_{output} = M_{input} \cdot f_i$$

To adjust the filter parameters, given by f_{new}, we would need to calculate loss, or error with respect to initial filter parameters, f_{old}. The formula for adjusting filter parameters remains similar to that of weight and bias adjustment, with a little difference.

$$f = f - lr \times (\delta L / \delta f)$$

Here, $\delta L / \delta f$ defines the change in loss L w.r.t filter parameter f. However, the loss is usually not expressed as a function of the filter parameters. We would try to find out equations defining loss in terms of filter parameter, f.
The loss L can be defined as

$$L = CCE_{multiclass} = -\ log\left(\frac{e^{s_c}}{\sum_{j=1}^{C} e^{s_j}}\right) = -\ s_c + \ log\left(\sum_{j=1}^{C} e^{s_j}\right)$$ (from equation iv)

Thus, derivative of loss w.r.t each class score can be derived as

$$\delta L/\delta s_i = \frac{e^{s_c}}{\sum_{j=1}^{C} e^{s_j}} - \delta s_i(s_c) = \frac{e^{s_c}}{\sum_{j=1}^{C} e^{s_j}} - 1(y = i)\ where\ 1(y = i) = 1\ if\ y = i,\ 0\ otherwise$$ …(v)

Let us define the last or n^{tg} layer of neurons, containing k nodes, that have the activations as a_0^n, a_1^n, …, a_k^n. For simplicity, we take A_n to be the integrated activations in the layer n. Similarly, activations of the previous layer are A_{n-1}, and so on. In each layer, the activations produce outcomes based on weights and biases. Again, for simplicity, let us take integrated weights and biases of the n^{th} layer $W_{n.}$ and $B_{n.}$, respectively. Therefore, the class score, s_i and its derivative w.r.t A_n can be defined as:

$$s_i = W_n \times A_{n+} B_n$$

$$\delta s_i/\delta A_n = W_n$$ …(vi)

Next, we have the activation A_n as the ReLU function of the previous layer's output, s'i.
$A_n = ReLU(S_{n-1}) = s'i$ when $s'_i = 1$; 0 otherwise.

$$\delta A_n/\delta s'_i = 1 \text{ when } s'_i = 1$$
$$= 0 \text{ otherwise.}$$ ...(vii)

Therefore, only the positive scores would be propagated as errors.
Lastly, the score, s'_i is basically, the convolution matrix, M_{output} as we obtained in the convolution step during forward propagation.

$$\text{Or, } s'_i = M_{input} \cdot f$$

Differentiating s'_i w.r.t f gives us M_{input}. Hence,

$$\delta s'_i/\delta f = M_{input}$$ …(viii)

Finally, we can compute the $\delta L/\delta f$ from equations v, vi, vii, and viii as:

$$\delta L/\delta f = \delta L/\delta s_i . \delta s_i/\delta A_n . \delta A_n/\delta s'_i \cdot \delta s'_i/\delta f$$ …(ix)

The final derivative is convoluted instead of dot product since the corresponding step in forward propagation was convoluted. So, for a single convolution layer, the filter parameters can be adjusted in the following way:

$$f_{new} = f_{old} - lr \times (\delta L / \delta f)$$

Haarcascades:

The haarcascade was first introduced by Viola and Jones in 2001.
It uses the concept of haar like features which identifies dark and light regions in an input image for example in fig 1.9 - The first extracted feature extracts the information that the region of eyes is darker than the surrounding regions.
And the second feature extracts that the bridge of the nose is lighter compared to the eyes.
Haar like features consists of edges, lines and diagonals as given in fig 1.10

Haar Cascading initially takes up the input image, identifies the haar like features and produces a single value by taking the sum of the intensities of the light regions and subtracts that value by the sum of the intensities of dark regions. For an image of size 24*24 pixels if all features are applied on it then the computation is of order of a million. Therefore, for each feature it finds the best threshold to classify the faces from non-face images using Adaboost. Here each image is given equal weight initially and then weights of misclassified images are increased, this process is repeated and error rates are calculated until the required accuracy is achieved. This helps to reduce the computation drastically, leaving it easier and faster.

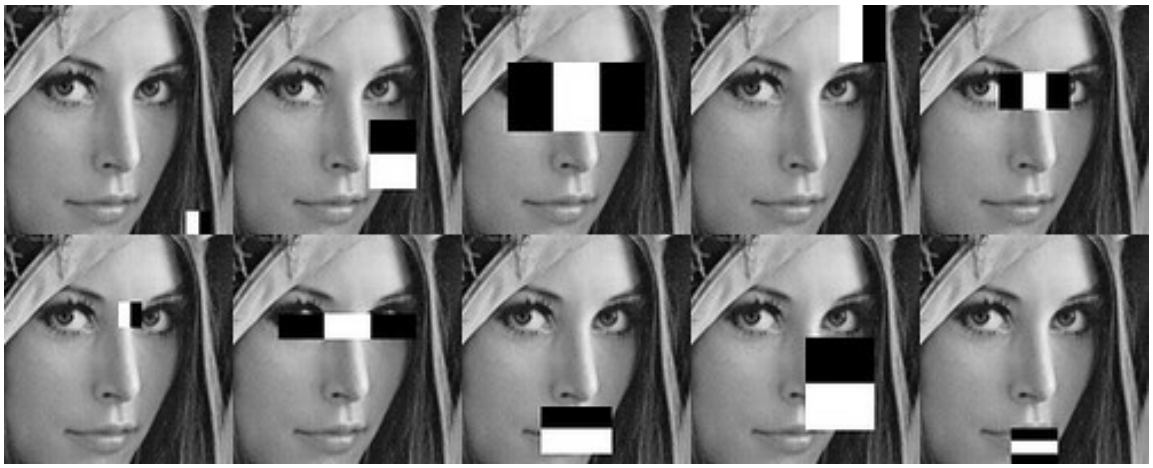


Fig. 1.9[12]- frontal feature extraction

Haar cascading is implemented with a classifier which is trained to identify faces from a dataset containing images of both face and non-face images. So, traditionally the haar classifier is required to extract features from all the images faces, non-faces and find the lines with minimum errors, which prominently identifies the minimal requirements of a face.

This method is very fast and effective. However, if the kernel size (no of features extracted) is 6000 for a single image of dimension 24*24 pixels then the haar classifier would have to extract all 6000 features even in those images which are not faces. To avoid this, the concept of cascade of classifiers is introduced. The idea is to divide feature extraction into stages which are applied

on an image one after another (cascades). If an image does not test positive for all filters in stage 1, the image is discarded since it is assumed to contain no face and hence the name Haar Cascade classifier.

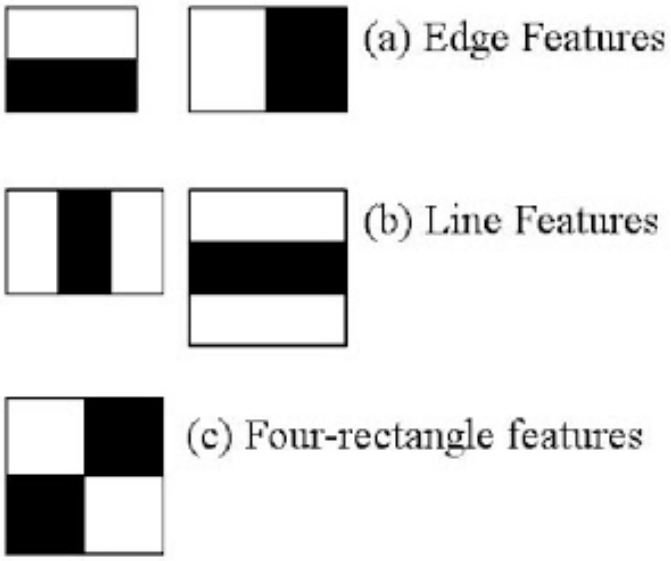


Fig 1.10[12]- edges line diagonals

1.2. Objectives

1) On executing the program file the attached web camera is supposed to open and capture and display a real time video stream.
2) A boundary box will be created around the faces in the frame labelled with the respective emotions displayed on the face.

1.3. Purpose, Scope and Applicability

1.3.1. Purpose

Emotion Recognition can be used for various purposes by a machine or AI. One of the purposes aimed at in this project is human machine interaction with AIs or trying to botch the results of the Turing Test. If applications like AI chatting apps, like Replika, were able to detect emotions on the human face via the use of webcams or front cameras while interacting with their users, the user would get more favourable outcomes from their interactions. It would make humankind many steps closer to making the perfect individual device which can function on its own.

1.3.2. Scope

The scope of this project covers the creation of a customized CNN model using keras, the model is supposed to be trained on a dataset containing images of faces labelled as happy, sad, angry, neutral, fear, surprise and disgust. We are using open cv to capture a video object using the attached web camera and disintegrate into frames. The program would use a haar cascade classifier to identify faces from the image frame and run the cnn classifier to recognize the emotion.

The primary assumption here is the fact all faces recognized are frontal profiles. Occulisions such as side faces or quarter faces are not yet implemented.

Additionally, the emotion in a reference frame does not only depend on the facial expressions but also on the context of the surrounding objects. For example, in a reference containing a person with a happy face if on a beach can be considered to be happy; at a birthday party can be considered to be surprised; an interviewer can be considered to be neutral and while holding a knife can be considered to be angry. So the emotions recognized in a frame with respect to both facial expressions as well as the context in the background can be considered a multi label classifier problem instead of multiclass classifier problem where the emotions resulting from all aspects in the frame is displayed probability wise.

1.3.3. Applications of face emotions

- **In the field software engineering**

Some aspects of software engineering can be improved with the help of these emotion recognitions by multimodal inputs and some of the limitations can be improved like questionnaires of usability.

Here we will focus on 2 aspects of software engineering where this emotion recognition has its applicability.

SCENARIO-1: USABILITY TESTING OF SOFTWARE

Human emotions influence a lot whenever a customer is trying to buy something. Emotional interactions play a huge role in this case of software products too. So these investigations of emotions are an object of interest for designers, investors, marketing persons, producers and customers as well.

There are multiple factors on which software usability depends like functionality, reliability, interface design, performance, maintenance in future can be possible or not. Now all these quality indicating factors are measured with the help of questionnaires which is misleading most of the time, to improve these factors if we don't solely depend on customers' satisfaction point of view that's a much better procedure. Here comes the implementation of emotion recognition to extend this usability testing further by the use of multimodal inputs.

For instance, when a web page is designed the first impression it provides to its users makes a user further go on to visit or make that quit. This first impression evokes a state of emotion like is it providing the user disgust or makes the user excited through a visual input. This first impression provides a distinctive effect for user's interest.

Another instance, when an application is developed it is checked whether a user can freely interact or not. From this there is an evaluation of overall user experiences. This is mostly done for content access and entertainment mediums and for other software applications also. Bottom-line is here emotion recognition is in the form of engagement even after there are bugs or other obstructions.

Comparative studies are also done on 2 different versions of software. This leads to main competitive products of software by combining different methodologies.

Another case, in a task based approach here there is a requirement to identify the typical tasks and what can be the optimal ways to perform them are taken into account. Tasks are in restricted environments like through keystroke analysis, biometric sensors etc. From these tasks emotional state fluctuations are analysed as well further usability also analysed. This is mostly carried by the user groups for a particular task not for entertainment systems etc. In this case emotion recognition is in the form of frustration generated from empowerment.

SCENARIO-2: PROCESS IMPROVEMENT VIA DEVELOPMENT

Software developers' performance metrics require high efficiency and quality code at the same time. Now the conflict happens here, because the computer program that is to be developed is under high pressure time constraints which usually leads to low code quality [MacCormack et al.2003]. Emotional state here also plays a significant role

in productivity in the developers in the development of software. Some sort of correlation can be derived between these factors like emotional state, code developed and the developers' efficiency. Now there are multiple factors which affect these emotional states of human beings like work environments, personal productivity, development of the quality code etc.

Taking some of the cases where emotion recognition has its applications are,
Relation of productivity and emotions, is it really that true that the productivity of developers are correlated to their present emotional state, even if we consider its true then how it is affecting. Here the programmer is affected by the environment of their working areas, this evokes a sort of negative emotion which decreases their productivity like no lines of code production rate is slow. Taken into account stress factors and boredom are induced. The analysis focuses mostly on optimal emotional space for a developer.

Another instance, how emotional state affects code quality are analysed. This requires continuous evaluation of the emotional state of the developers and as well the developments made in each stage of the source code i.e., collection of those incremental versions of the code. The correlation between these 2 factors (quality and emotions) can only be drawn only when cross-examination of emotional states and source code are done. Here emotional states to take under consideration are stress, frustration etc.

Another instance applicable to this new normal era, working from home and from office comparison. Differences in productivity and emotional states while working from home and office are analysed. This is to be conducted on real work environments situations. It can be done only when the methods of users' emotional states recognition methods are reliable, non-intrusive etc. Here main aim of emotion recognition is to capture a whole range of emotions.

However, it is possible nowadays to capture emotions of programmers with the help of biometric sensors. More accurate forms of recognition with respect to non-intrusive methods.

This is a natural choice for the implementation purpose of emotion recognition mechanisms in human-computer interface, because the primary working environment of a programmer is solely the computer.

- **In the field of education and e-education.**

Evidence says some emotional states support learning processes and others may suppress them. [Hud licka 2003, Picard 2003, Sheng et al.2010] However it can be stated that positive mood may not involve good vibes for learning process whereas negative emotional states includes critical state of thinking and provides good analytical tasks skills. Therefore, instead of typical questionnaire based investigations, it's better to have automated emotional recognition algorithms to explore learner emotional states.

Through various cases we can explain emotional states and its applications, here possible emotional states generated while execution of educational tasks is examined. This aims at identification of emotional templates at the time of tasks, this is further defined through various sets of effective and counterproductive emotional states while

solving specific task types. Learners' performance in the execution of tasks and his/her emotional state are measured. This correlation is achieved through various sets of effective and counter-productive emotions that took place in task types. To make this effective and reliable significant number of respondents are also in requirement. These kinds of data after generated can be further used for educational problems diagnosis, tool design, or in further exploration of educational purposes.

Another instance, educational tools are tested and their usability is evaluated over here. These tests are based on eye tracking techniques and we propose to extend it with user emotion recognition which is a valuable information while evaluating user experience [5]. Tasks involved with these tools are: resources search, viewing results, feedback information, communication with mentors, classmates etc., resource launch, performing interactive tasks. With the help of a representative group of students these are performed in a controlled way to recognize the emotional states. This will further help in improvement of software products that are designed for learning methods, as this information holds various effects on emotional states of a learner and his/her levels of fluctuations in frustrations.

Another consideration is evaluations of resources related to educational purposes, for particularly those meant for self-learning. When a learner opts for distance learning and electronic education, the learner lacks self-discipline which leads him/her in discontinuation of the courses and learning methods and the learner comes up with vague reasons like “boring resources”. Here the learners’ interaction with resources along with their emotional states are analysed to calculate the parts of resources that caused the boredom. That data leads improvement of resources and removal of weak areas from those resources. A fresh set of recorded lectures in a new interactive way is put up along with interactive study materials, quizzes, short hands on experience projects or assignments are also provided so the engagement and no hours spent is more. This leads to a quality in the educational resources after the emotional states are determined.

During a particular educational setting typical sets of emotional states are observed. Likewise, novice learners show symptoms of frustrations while experienced ones show boredom. To make this hypothesis more evident under the same set of tasks of growing difficulty were provided to both types of learners and measured their variations of emotional states. Learner stereotypes are further used for e-educational environments also for a difference in learning paths and differences in interaction models. So the fact is every human personality is born with different emotions within them, but due to the same culture or incidents the states may be the same.

➢ In the field of video games

A large portion of today’s youth community are always in this video games world. Sometimes they are so engrossed with being able to play in virtual form as players that they feel absorbed by those emotions that happen to occur during playing mode. Emotional attachment is with those virtual avatars of themselves as a player forms a greater part of their day to day life. As the primary goal of a video game is to

entertain the player [2], each video game lets the player fulfil their dream. Video games try to make it attractive with the help of graphics, genre, good gameplay, immersing storytelling, novelty etc. Emotions over here play a greater role while video games are being played and few try to incorporate their player's affective states into the gameplay. These games are called effect aware games or affective ones. This effect awareness state brings out the representative player based on the model assumed for these representations. Two problems generated from this each player differs from another in some way. Another player's emotional states are changed from time to time and from one session to another session so this radical change makes it impossible to predict a certain emotional state of current users at the developmental stage.

The factors which affect a players behaviour during these games are becoming an accustomed player, increasing in monotony, and some game independent factors are also connected with current physical and mental states. Some of the reasons are already predicted by game designers. For this reason, real time recognition of player's emotions is important for these video games industries. These video games are dynamically reacting to the player's emotions and as well truly affect awareness of these games.

Various instances are like where emotion recognitions finding applicability,

Adaption of stimulus, here the player's attention is bounded as long as possible with it. For these levels of engagement developers use interesting stories, high quality graphics etc.to keep the attention of the player. Basic aim is to verify, whether long, frequent, repeated stimulation causes negative impact on the player due to this stimuli adaptation. After a certain period of time the players stop to react based on stimulus, and probably the players want to have more powerful experiences. After the stimuli disappears the moment of weariness is tested. So basically here the time or span of time where the player stops reacting on the stimuli is determined due to the loss of it.

Wide spectrum of emotions is generated while these video games are played by the players but not all of them are observed. This expression of emotions depends on the game's genre, player's genre, experience and other predictable and unpredictable factors [2]. Reactions that are based on an individual's emotions are due to individual personality, current mood and other environmental factors too externally affected. So narrowing down here which emotions are common while playing are taken into further consideration. These common emotional states are due to specific game genres, specific group of player's involvement. To get the different emotional states different video games with different levels of players along with prepared scenes need to be taken into consideration. Certain questionnaires will lead to predictions of players age, gender and gaming experiences.

Reaction of players on external signals on different levels of indulgence, sometimes there is so much indulgence that they forget the real world around them. Here monitoring is important as we need to know when they stopped reacting to the external stimulus. This is required to prevent addiction. This is because when a player is too involved in these virtual game modes this affect aware game can reduce its attractiveness, probably helps them slip back to the real world. Appropriate involvement may not be determined specifically. A Player's reaction can be rapid, slow, quiet, fast etc.

this determines the degree of the player's engagement on that particular game. While carrying out this investigation various disturbing verbal languages, noises etc. are also used to bring out that person's real forms of emotional states.

Satisfaction level of a player is also determined after playing these gameplay characters. Sometimes these games can be too boring and too stressful if these conditions can be determined and a proper feedback can be produced then these can provide young players protection from violence and at the same time experienced players can feel the satisfactions. Adapting feedback mechanisms can also be a great help in therapy sessions in suppressing the negative situations that have occurred previously and can be prevented further by proper usage of stimulation. This is conducted to correlate affect-aware games and the satisfaction levels of a player from playing. According to the detected emotions a questionnaire is assumed depending upon the increasing or decreasing levels of difficulty in the game and this will determine the levels of satisfaction derived by the players.

1.4. Achievements

Currently (Phase 1) our project is able to meet the objectives mentioned in sections 1.2. The following points are observed.

1) On executing the program file, OpenCV captures the real time video stream through the attached web camera.
2) The captured video stream is disintegrated into 30 frames per second and passed through a haar-cascade-classifier model to identify frontal faces.
3) The frontal faces are sent one by one through the CNN classifier model to recognise the category of emotion
4) Finally, a boundary box is generated around the faces in the frame labelled with the respective emotions displayed on the face.

1.5 Organization of report.

The following chapters in this document covers related works in this field, software and hardware requirements of a system to install the program, design of a project, implementation and testing and finally a few discussions obtained based on the results. The theory described in section 1.1 (background) is applied while designing the core structure of the cnn model i.e. the layer structures, the kernel size, haar cascade classifier using open cv. The components of the program are then integrated together into one single application. The result and discussion section highlights some important drawbacks as well as ideas to overcome those drawbacks in this field.

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1. Ways in which face emotion recognition is done?

Facial expressions are the building blocks of human emotions. Interpretations of those facial expressions are required to be done effectively in order to understand the particular state of emotions of a human being. Many numerous projects and research works are done in the past to recognize the human emotions from facial expressions, many of them used many methods to implement the system effectively. Here also it is important to study what works are or techniques are already existing in the past.

1. **Facial action coding system(FACS)** — Originally adopted by Swedish anatomist Carl-Herman Hjortsjö. Later, adopted by Wallace V. Friesen and published in 1978. This system is based on a comprehensive, anatomically determined way to determine the various facial aspects which are visual from various facial movements. The individual units are broken down further to inspect in depth and details and these are called as action units.

   Whenever there is a change in the slightest facial movements these changes are reflected with the help of the FACS encoding procedure through the various muscles present in facial portions. This is a common standard practice to systematically categorize the various facial expressions of human beings. Recently, FACS has been established in such a computed automated environment so that it can detect faces in video systems and then detect the geometrical features and build temporal profiles of those features for further processing purposes. Fasel and Luettin performed an in depth research to figure out what can be the possible sources for facial expressions. Mainly it was found that the underlying muscles were responsible for the generation of expressions. For example, studies show that most primary things which are responsible for such was movement of eyes, mouth, eyebrows etc. Most of the previous works which are done in FACS become the framework of classification. For the FACIAL FEATURE EXTRACTION, FACS become a major key system.

   In addition to this, previous studies have traditionally taken two approaches to emotion classification according to Fasel and Luettin.

   - Approach based on judgement
   - Approach based on signs.

The judgment approach is developed in which six universal emotions are categorized precisely.

The sign-based approach uses a FACS system, encoding action Units are provided in order to categorize an expression based on its characteristics. Here it assigns an emotional value to a face using a combination of the key action units that create the expression. This emotional value generated, used by neural-network to recognize action units from the coordinates of facial features like lip corners or the curve of eye brows or parts of the eyes.

2. **Principal component analysis(PCA)** — Principal component analysis (PCA) is widely used in the field of data compression and feature extraction. Two basic approaches for the computation of principal components: batch and adaptive methods.

   The batch methods include the method of Eigen decomposition and the method of Singular value decomposition (SVD).

   On the other hand, the adaptive methods are mainly done by neural networks.

   The main target of PCA is to explain the variance–covariance structure of the data through a few linear combinations of the original variables. Basically it is a technique used to reduce the dimensionality of large volumes of datasets into a compact way without losing most of the important and required information of the datasets.

   The main concerning thing about PCA is that it utilizes only the global information of face images, this method is not very effective for different facial expressions.

   Mathematically if we put it, PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance value on the second coordinate, and so on.

   Step by step brief explanation: -

   a. Standardization — Standardize the ranges of the variables initially so that each of them can contribute equally for the analysis purpose. Though variances may be lost due to the standardization procedure before PCA. Even if there are large differences in initial variables ranges this will dominate over the small ranges which will lead to biases of results. So transformation of data is to be done on a comparable scale to prevent this sort of problem.

      After standardization the variables are on the same scale factor.

b. Covariance matrix computation — Here main aim is to find how the variables of input data are deviating from mean values or what can be the possible relationship between them. There may be some redundant data present and variables need to be highly correlated that time. To identify such correlations, we need to compute the covariance matrix.

c. Computation of Eigenvalues and vectors to identify principal components — From the covariance matrix these Eigenvalues and vectors are needed in order to calculate the principal components of data. Organizing the data to reduce dimensionality without losing important ones and considering remaining as your new variables. Principal components are less interpreted and don't have real meanings since construction is mostly of linear ones of initial variables. Principal components represent maximal amount of variance, i.e., the lines that capture the most of the information. The relation of variance and data, larger the variance larger the data dispersion points along the line. This provides a new axis to provide the best angle for evaluations of data and differences in observations are better visible.

3. **Aw-SpPCA** — This algorithm is another promising mechanism that has been used to detect emotion from image. This algorithm is based on the concepts of impound illumination conditions, facial expressions variations to local areas dividing a face image into several sub-images, and carrying out PCA computation on each local area independently.

   Appearance based Models is another approach to be considered. These rely on techniques from statistical analysis and machine learning to find the relevant characteristics of face images. Some appearance-based methods work in a probabilistic network.

   An image or feature vector is a random variable with some probability of belonging to a face or not.

4. **Neural networks** — This is a key tool of extracting features from an image. Many pattern recognition problems like object recognition, character recognition, etc. have been faced successfully by neural networks. These systems can be used in face detection in different ways.

   Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria.

   The concept of neural networks which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

   Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data.

They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment, for stock market price prediction varies.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. For instance, the patterns may comprise a list of quantities for technical indicators about a security; potential outputs could be “buy,” “hold” or “sell.”

Hidden layers fine-tune the input weightings until the neural network’s margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.

Some of the applications of neural networks now-a-days, Neural networks are broadly used, with applications for financial operations, enterprise planning, trading, business analytics and product maintenance. Neural networks have also gained widespread adoption in business applications such as forecasting and marketing research solutions, fraud detection and risk assessment and prediction.

A neural network evaluates price data and unearths opportunities for making trade decisions based on the data analysis. The networks can distinguish subtle nonlinear interdependencies and patterns other methods of technical analysis cannot. According to research, the accuracy of neural networks in making price predictions for stocks differs. Some models predict the correct stock prices 50 to 60 percent of the time while others are accurate in 70 percent of all instances. Some a 10 percent improvement in efficiency is all an investor can ask for from a neural network.

The most important aspect of this algorithm, like there will always be data sets and task classes that are better analysed by using previously developed algorithms. It is not so much in these algorithms that matters; it is the well-prepared input data on the targeted indicator that ultimately determines the level of success of a neural network.

5. **Support vector machines(SVM)** — SVMs are linear classifiers that maximize the margin between the decision hyperplane and the examples in the training set. So, an optimal hyperplane should minimize the classification error of the unseen test patterns. This classifier was first applied to face detection by Osuna et al. Support-Vector Machines or SVMs are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis.

If we are given some data points each belong to one of two classes, and the goal is to decide which class a 'new' data point will be in then we may use SVM. SVM views a data point as a p dimensional vector (a list of p numbers) and we want to know whether we can separate such points with a (p-1)-dimensional hyperplane. This is called a linear classifier.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. We choose that hyperplane which represents the largest separation between two classes. This means that the distance from the hyperplane to the nearest data point on each side is maximized. This type of hyperplane is called a maximum-margin hyperplane.

SVM algorithms use a set of mathematical functions known as kernels. The function of a kernel is to take data as input and transform it into the required form. The kernel functions return the inner product between two points in a suitable feature space. A feature is an individual measurable property or characteristic of a phenomenon being observed and these features are represented as feature vectors. The vector space associated with these vectors is often called the feature space. Kernel functions reduce computational cost in very high-dimensional spaces.

Support-Vector Machines have a wide ranging application. It is used to classify facial expressions. It is also used to classify textures of surfaces; SVM detect surfaces as smooth or gritty. It is also used to classify handwritten text. SVM is also used in speech recognition where we collect acoustic data using functions like Linear Prediction Coefficient (LPC)

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labelled training instances in both the standard inductive and transductive settings. Some methods for semantics parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevant feedback. This is also true for image segmentation systems.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support-vector machine weights have also been used to interpret SVM models in the past. Post-hoc interpretation of support-vector machine models in order to identify features used by the model to make predictions is a

relatively new area of research with special significance in the biological sciences.

6. **Hidden Markov model(HMM)** — It is another statistical model that has been used for face detection. The challenge is to build a proper HMM, so that the output Probability can be trusted. The states of the model would be the facial features which are often defined as strips of pixels. The probabilistic transition between states is usually the boundaries between these pixel strips. As in the case of Bayesians, HMMs are commonly used along with other methods to build detection algorithms.

   Hidden Markov models are especially known for their application in bioinformatics, reinforcement learning, temporal pattern determination like gesture, speech, musical scores, handwriting and sometimes parts of the speech also.

   The term hidden refers to the first order Markov process behind the observation.
   Observation refers to the data we know and observable.
   In machine learning sense, observation is our training data, and the number of hidden states is our hyper parameter for our model.
   HMMs are also used in all these biological aspects too. These are statistical models to capture hidden information from observable sequential symbols (e.g., a nucleotides sequence). They have many applications in sequence analysis, in particular to predict exons and introns in genomic DNA, identify functional portions in proteins (profile HMM), align two sequences (pair HMM). In a HMM, the system being modelled is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters.

   A good HMM accurately models the real world source of the observed real data and has the ability to simulate the source. A lot of Machine Learning techniques are based on HMMs have been successfully applied to problems including speech recognition, optical character recognition, computational biology and they have become a fundamental tool in bioinformatics for their robust statistical foundation, conceptual simplicity and malleability, they are adapted fit diverse classification problems.

   In Computational Biology, a Hidden Markov Model (HMM) is a statistical approach that is frequently used for modelling biological sequences. In applying it, a sequence is modelled as an output of a discrete procedure, which progresses through a series of states that are ‘hidden’ from the observer. Each such hidden state emits a symbol representing an elementary unit of the modelled data, for example, in case of a genomic sequences, an amino acid.

   We describe some important aspects of modelling HMM in order to solve real problems, giving particular emphasis in its use in biological context. In this work we illustrate, as example, applications in computational biology and bioinformatics and, in particular, the attention is on the problem to find regions of DNA that are methylated or unmethylated.

7. **Linear Discriminant Analysis(LDA)** — Method is used in statistics and pattern recognition to find a linear combination of features. The resulting combination may be used as a linear classifier or, more commonly, for dimensionality reduction before later classification. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities.

   Logistic regression is a classification algorithm traditionally limited to only two-class classification problems. If more than two classes, then Linear Discriminant Analysis(LDA) is preferred.
   Shortcomings of logistics regression problem: -
   Logistic regression is a simple and powerful linear classification algorithm. It also has limitations that suggest the need for alternate linear classification algorithms.

   - **Two-Class Problems**: Logistic regression is intended for two-class or binary classification problems. It can be extended for multi-class classification, but is rarely used for this purpose.

   - **Unstable with Well Separated Classes**: Logistic regression can become unstable when the classes are in separated formats.

   - **Unstable with Few Examples**: Logistic regression can become unstable when there are few examples from which to estimate the parameters.

   Linear Discriminant Analysis does address each of these points and is the go-to linear method for multi-class classification problems. Even with binary-classification problems, it is a good idea to try both logistic regression and linear discriminant analysis.

   The representation of LDA is straight forward.

   It consists of statistical properties of your data, calculated for each class. For a single input variable (x) this is the mean and the variance of the variable for each class. For multiple variables, this is the same property calculated over the multivariate Gaussian, namely the means and the covariance matrix. These statistical properties are estimated from your data and plug into the LDA equation to make predictions. These are the model values that you would save to file for your model.

   LDA makes some simplifying assumptions about your data:

a. That your data is Gaussian, that each variable is shaped like a bell curve when plotted.

b. That each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.

With these assumptions, the LDA model estimates the mean and variance from your data for each class. It is easy to think about this in the univariate (single input variable) case with two classes.

2.2. Algorithms used for FACE-DETECTION extraction

1. **PCA** — In principle component analysis, it uses Eigenvalues and vectors and reduces the dimensionality and projection and training of sample or data on a small feature space. Proper centred face is required for training and testing.
2. **KERNEL PCA** — Works on one dimensional data. Extension of PCA. This is used for non-linear datasets. Here kernel function is used to project the dataset into higher dimensionality feature space. Various kernel methods are linear, polynomial and Gaussian.
3. **WEIGHTED PCA** — Every image in the training set is a linear combination of weighted Eigenvalues and vectors also called Eigenfaces.
4. **LDA** — Popular feature extraction technique for face image recognition and its retrieval. But it suffers from small sample size while the face data is of high dimensional.
5. **KERNEL LDA** — Extraction of non-linear components, captures all the overall variance of all patterns which may not provide necessary significance for discrimination purpose.
6. **SEMI-SUPERVISED DISCRIMINANT ANALYSIS** — Works on both labelled and unlabelled data samples. Even if the data manifolds this function works as a discriminant one to deal with such.
7. **INDEPENDENT COMPONENT ANALYSIS** — Reveals set of hidden factors underlying random variables, measurements or signals. It is for multivariate data which is typically found in large datasets.
8. **NEURAL NETWORK BASED MODELS** — Complex non-linear relationships between the predictors and its response variables are generated.
9. **MULTIDIMENSIONAL SCALING** — Level of similarity of individual cases within a dataset or dissimilarities between sets of objects.
10. **SELF ORGANISING MAP** — High dimensional datasets are represented as 2-dimensional discrete patterns. This reduction in dimensionality is to be performed while keeping the topology of data as same or as of present in the original feature space.

11. **ACTIVE SHAPE MODELS** — These are statistical models used for the shape of the objects which deform to fit into the object of new images. Repetitions of initial poses until convergence.
12. **ACTIVE APPEARANCE MODELS** — This is a computer vision algorithm where object shape and appearance are matched to form a new image. They are built in the training phase.
13. **GABOR WAVELET TRANSFORMS** — The image of the face is convolved with a set of Gavor wavelets and resulting images further processed for face recognition. This method is performed on the expression of a face image to obtain magnitude and phase characteristics.
14. **DCT (DISCRETE COSINE TRANSFORMATION)** — Separation of image parts based on different frequencies. On the quantization step, less important frequencies are discarded

2.3. Convolutional Neural Network (CNN--Brief History)

1. 1950 — From a biological experiment
2. 1959 — Russell-Kersh apparatus — Here transformation of image took place into a grid of nos.
3. 1963 — Machine perception of 3-D solids — Around this time the birth of computer vision took place. AI also became an academic discipline.
4. 1982 — David Marr — Introduced a framework for vision where low-level algorithms that detect edges, curves, corners etc. are used as stepping stones towards a high level understanding of visual data.
5. Neocognitron
6. 1989 — Early version of CNN by Yann Le Cun. Called as LENET, could recognize handwritten digits. This work led to the MNIST dataset which is a benchmark dataset of ML.
7. In late 1990s — COMPUTER VISION EMERGED
8. 2001 — First face detection framework although not based on DEEP LEARNING but has the flavour of it. [Viola jones algorithm]
9. 2009 — The deformable part model. 'Magnet' dataset becomes a benchmark in object category classification and object detection across a huge number of object categories.
10. 2010,2011 — Error rate in image classification hovered around 26%.

This has a breakthrough moment for CNNs.

2.4. Comparison of CNN w.r.t to ML (why CNN is better)?

1. Mostly because of its high accuracy CNNs are used for image classification and recognition. These CNNs mostly feed forward neural networks.

2. CNNs follow a hierarchical manner where it works as building a network which is a fully connected layer where all the neurons are interconnected and the output is also processed further.
3. Without any human supervision these CNNs can detect features as of compared to its predecessors which makes it comparatively more in use. It has that capability to learn the key features of each class by itself or given many images of cats and dogs it can differentiate.
4. CNNs are computationally more efficient and powerful than the Machine Learning algorithms.
5. Traditionally, multilayer perceptron is used for image recognition but they are fully connected via the nodes which makes it suffer from better dimensionality i.e., precisely they didn't scale well with higher resolution images. In CNNs these models which are used for image classification and recognition are the variants of multilayer perceptrons which are designed in such a manner which can provide emulation to the visual cortex layer.
6. CNN architecture is made up in such a manner that it has 3-D volumes of neurons where height, width and depth are taken into account. Here each neuron is connected via a small region prior to it which is called a receptive field. Distinct layers are stacked both in a local way and in a connected format to make this sort of architecture.
7. Local connectivity pattern is followed up between the neurons and the adjacent layers in the architecture, this leads to spatial exploitation. This type of architecture makes filters as the strongest response to this locally input pattern. To make it global these such layers are stacked up, so that the representations of small parts of input are done firstly and then representations of larger areas.
8. Each of the filters that is generated across the entire visual field are replicated in such a way that it holds the same parameterization and forms a feature map. This states that all the neurons within a convolution layer respond to the same features in a specific response field. Thus the resulting feature map is equivariant even under the changes in the location of input features due to replicating units and its presence in the visual field. CNNs grant transitional equivariant due to this shared weights concept.
9. CNNs are robust to variations wrt positions due to the presence of pooling layers. This is where the feature maps are further divided in rectangular blocks and features in each of the rectangular portions holds a single value, by taking of average or maximum value. This pooling reduces the sizes of feature maps and as well grants the degree of equivariance of the features present within it.
10. Weight sharing leads to reductions of parameters which leads to lowering of memory requirements for running the network and also allows training of larger and powerful networks.

CHAPTER 3

METHODS AND REQUIREMENTS

3.1. Summary of Requirements

RAM — 8gb DDR4

Graphics Card — NVIDIA graphics card (optional, for better results)

Processor — i5 or Amd razon 5

IDE — Jupyter notebook

Libraries used — numpy , seaborn, pandas, os, keras , matplotlib, opencv

Dataset — Facial emotion dataset. (FER-2013 Kaggle dataset)

Programming language — Python 3.8

3.2. Methods used

3.2.1. Model

- Convolutional Neural Networks used to create the model and its weights.
- The images in the dataset consists of gray scale images of size 48x48.
- Each image in the dataset displays an emotion.
- Model is trained on image pixel values.

3.2.2. User Interface

- Real time video is captured frame by frame using OpenCv.
- Frontal face is detected using Haarcascades and OpenCv.
- The emotion in the region of interest is recognized using the Model and displayed using OpenCv.

CHAPTER 4

SYSTEM DESIGN

4.1. Basic Modules

The project design can be divided into three sections.

1) Database Preparation
2) CNN Model
3) User Interface

4.1.1. Database Preparation

For our project, we are using a Facial Expression Images Dataset (FER-2013 Kaggle Dataset) containing 35907 images and 7 emotions. The dataset is split into training, testing and validation datasets each consisting of numbers of images as specified in table 4.1.

S.No	EMOTIONS	TRAINING	VALIDATION	TESTING	TOTAL
1	Happy	7215	897	895	9007
2	Neutral	4965	626	607	6198
3	Fear	4097	528	496	5121
4	Surprise	3171	416	415	4002
5	Angry	3996	491	468	4955
6	Disgust	436	55	56	547
7	Sad	4830	594	653	6077

Table 4.1 Database structure

4.1.2. CNN Model

The architecture of our CNN Model can be visualised with the help of Fig 4.1.

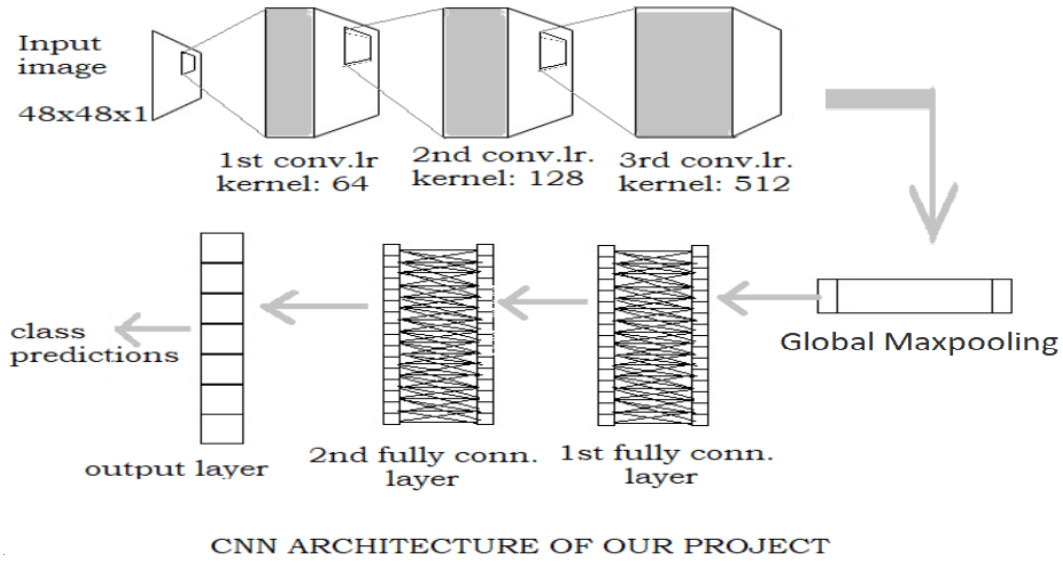


Fig 4.1- Architecture of customized CNN model

Our model consists of 3 convolution layers and 2 fully connected layers.

- The first convolution layer or the input layer is supposed to take an input of dimension 48×48×1. This layer is supposed to have the following attributes.
    1) No of kernels - 64
    2) Filter size - 3×3
    3) Pooling (Max pooling) - 2×2
    4) Activation function used - ReLU
- In the second convolution layer the input size is kept the same as the output of the first convolution layer. The second convolution layer would have the following attributes.
    1) No of kernels - 128
    2) Filter size - 3×3
    3) Pooling (Max pooling) - 2×2
    4) Activation function used - ReLU
- In the third convolution layer the input size is kept the same as the output of the first convolution layer. The second convolution layer would have the following attributes.
    1) No of kernels - 512
    2) Filter size - 3×3
    3) Pooling (Max pooling) - 2×2
    4) Activation function used - ReLU

- We introduce a global maxpooling2D layer layer which essentially flattens the output of the third convolution layer (3d matrix) into a 1d matrix.
- Next, the flattened matrix is fed into the fully connected layers The first fully connected layer consists of the following attributes.
    1) No. of kernels -512
    2) Activation function - ReLU
- The second fully connected layer consists of the following attributes.
    1) No. of kernels - 512
    2) Activation function - ReLU
- Finally, a dense layer is added as the output layer containing the following attributes.
    1) Activation function - Softmax
    2) No of classes - 7 (No of emotions recognized in phase 1)
- Additionally, the model is optimized with ADAM having the initial lr (Learning rate) as 0.0001
- Finally, the model is compiled with loss function sparse categorical cross entropy, penalizing itself on metrics of accuracy.

Depending upon the size of data and required accuracy of the model, the number of epochs can be set to 100. After training the model with the training dataset and validation dataset, two files would be generated as follows:

1. Weights of model as “cnn_emrec_weights1.h5”
2. Model as “cnn_emrec1. json”

These files would be used in the third section of the design, the User Interface.

4.1.3. User Interface

The user interface would require the default camera to be free. Images from a USB camera or default webcam will open in a window and real time video will be acquired frame by frame as soon as the user executes the program. The following are the attributes of UI.

- The name of the window displayed is “Facial Emotion Recognition”.
- The region containing the face is bounded by a bounding box.
- Bounding box is of color blue, and thickness: 7
- Emotion detected is written in red color in Hershey simple font with thickness: 2
- The displayed image has resolution 1000×700 pixels
- To close the interface the user can press either ‘q’ or ‘Esc’ from the keyboard.

4.1.4. Implementation Flowchart

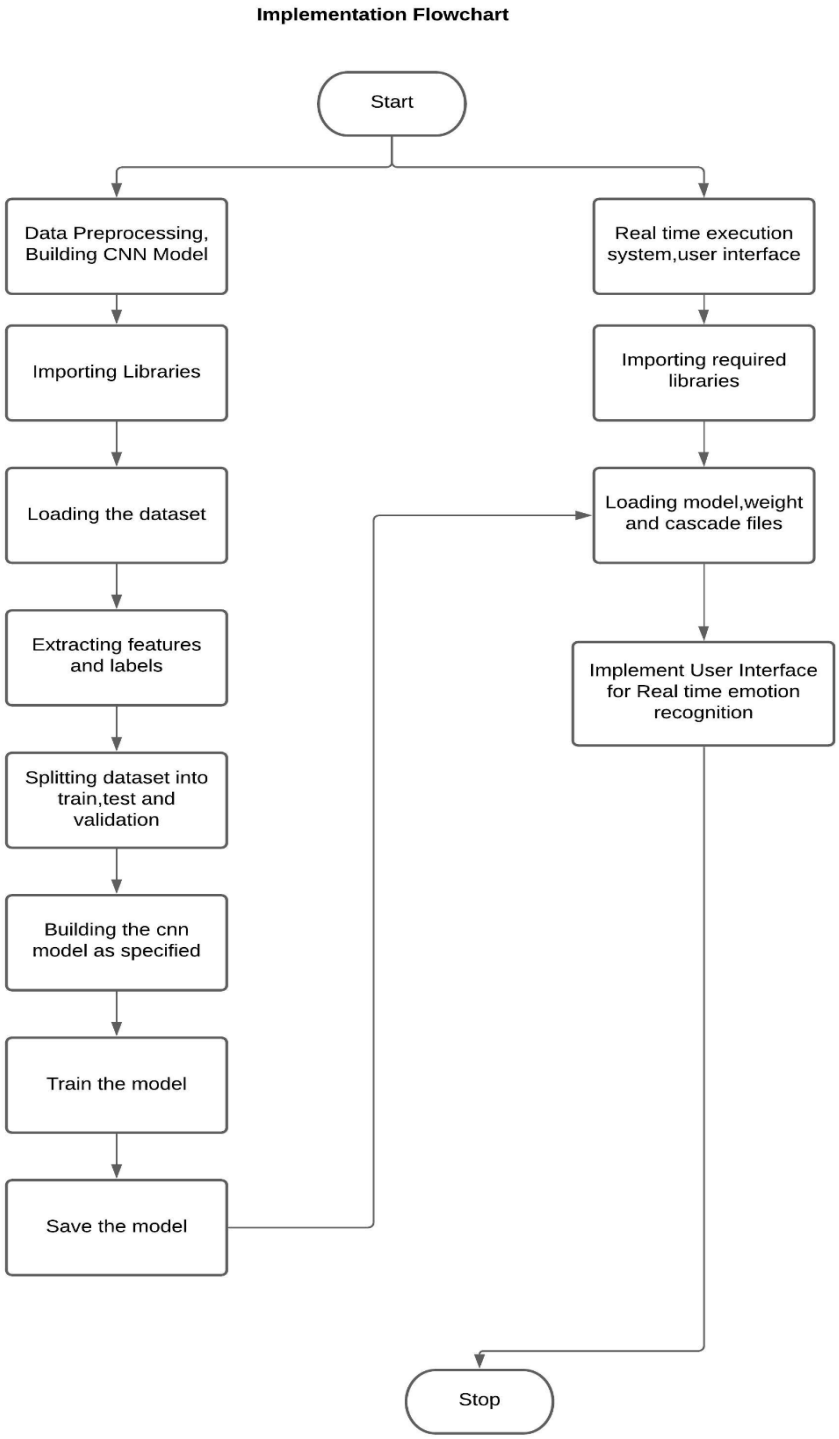


Fig 4.3 Flow of Implementation

4.1.5. Execution Flowchart

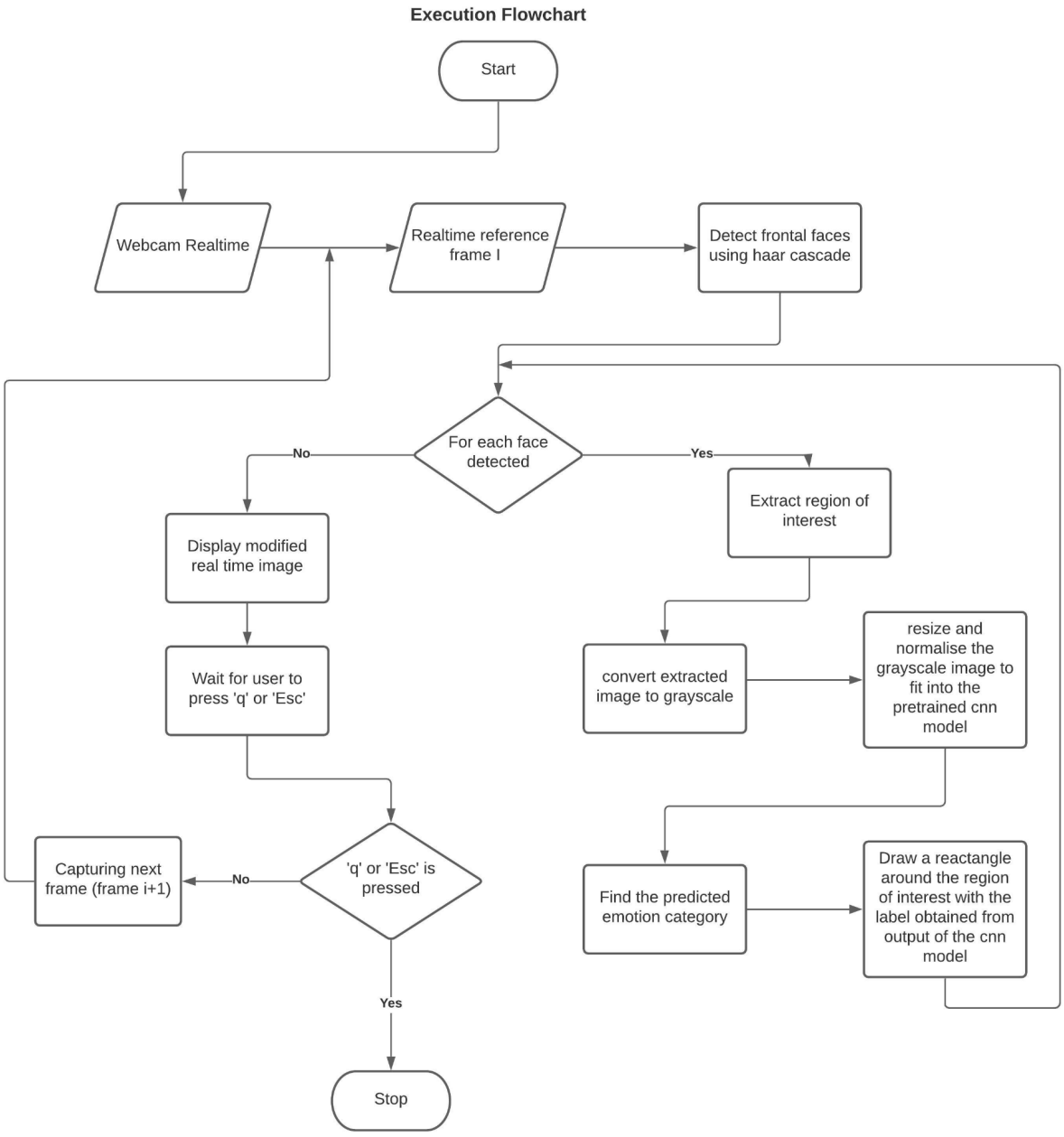


Fig 4.3- Flow of Execution

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1. Implementation

The implementation of the design mentioned in the previous chapter is done in 2 steps:

1. Data preprocessing and building CNN model
2. Real Time emotion recognition and User interface.

For both the steps, the source code is provided below.

5.1.1. PREPROCESSING OF DATA AND BUILDING THE CNN MODEL

- Language used: Python version 3.7
- Libraries used: pandas, numpy, matplotlib, tensorflow, keras, sklearn
- IDLE used: Google Colab (Colaboratory)
- Runtime used: GPU

---

1. Import the necessary libraries

```
]: from google.colab import drive
   drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
]: import pandas as pd
   import matplotlib.pyplot as plt
   import numpy as np
   import tensorflow
   from tensorflow import keras
   from keras.losses import categorical_crossentropy
   from keras.optimizers import Adam
   from tensorflow.keras import layers
   from sklearn.model_selection import train_test_split
```

2. Load the dataset as a pandas dataframe

```
#create dataframe with path to fer dataset csv file
df=pd.read_csv("/content/drive/MyDrive/Python ai course/fer2013.csv")

#printing dataframe shape and first 3 columns
print(df.shape)
df.head(3)
```

```
(35887, 3)
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training

3. Modify dataset to extract features and labels

```
#we allot each pixels to a sepaarte column
no_of_pixels=len(list(df['pixels'][0].split(' ')))

no_of_pixels #contains 48*48 pixels for each image
```

```
2304
```

```
# create a 2D numpy array where rows corresponds to images,
#and columns corresponds to pixels. Thus,
#order of matrix=(no of images) x (no of pixels)
arr=np.zeros((df.shape[0],no_of_pixels))

for index, row in df.iterrows():       #iterate rowwise in df
    pix=row['pixels'].split(' ')       #1D list of pixel values for each image
    arr[index]=np.array(pix,np.int64)  #store 1D list as array in corresponding rows
arr.shape
```

```
(35887, 2304)
```

```
#create a modified dataframe that contains the pixel values in separate columns
df_mod=pd.DataFrame(arr)
df_mod['emotion']=df['emotion']
df_mod['Usage']=df['Usage']
df_mod.head(3)
```

81	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296
0.0	78.0	21.0	11.0	61.0	144.0	168.0	173.0	157.0	138.0	150.0	148.0	132.0	159.0	182.0	183.0
5.0	159.0	150.0	139.0	128.0	116.0	125.0	133.0	109.0	130.0	147.0	130.0	121.0	105.0	108.0	95.0
3.0	90.0	106.0	123.0	146.0	155.0	148.0	130.0	141.0	119.0	69.0	54.0	89.0	104.0	138.0	152.0

```
#list out all unique emotions present that will be identified.
#the emotions are identified using numbers(or indices) instead of actual string name.
# emotions = ('angry','disgust','fear','happy','sad','surprise','neutral')
df_mod['emotion'].unique()
```

```
array([0, 2, 4, 6, 3, 5, 1])
```

4. Split dataset into train, test and validation datasets

```
#create a training dataset where usage = "Training",
#testing dataset where usage = "PublicTest",
#validation dataset where usage="PrivateTest"
df_mod.Usage.unique()
```

```
array(['Training', 'PublicTest', 'PrivateTest'], dtype=object)
```

```
dftr=df_mod[df_mod.Usage=="Training"]       #training
dfte=df_mod[df_mod.Usage=='PublicTest']     #testing
dfval=df_mod[df_mod.Usage=='PrivateTest']   #validation

#print shapes of each dataset
print(dftr.shape,"\n",dfte.shape,"\n",dfval.shape)

dftr.head(3)
```

```
(28709, 2306)
 (3589, 2306)
 (3589, 2306)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	70.0	80.0	82.0	72.0	58.0	58.0	60.0	63.0	54.0	58.0	60.0	48.0	89.0	115.0	121.0
1	151.0	150.0	147.0	155.0	148.0	133.0	111.0	140.0	170.0	174.0	182.0	154.0	153.0	164.0	173.0
2	231.0	212.0	156.0	164.0	174.0	138.0	161.0	173.0	182.0	200.0	106.0	38.0	39.0	74.0	138.0

3 rows × 2306 columns

```
#from each dataset, extract the features and labels
#features(x-part) = the pixel values (dropping the 'Usage' and 'emotion' columns)
#labels(y-part) = the 'emotion' column

#training
xtr=dftr.drop(['Usage','emotion'],axis=1)
ytr=dftr['emotion']

#testing
xte=dfte.drop(['Usage','emotion'],axis=1)
yte=dfte['emotion']

#validation
xval=dfval.drop(['Usage','emotion'],axis=1)
yval=dfval['emotion']

print(xtr.shape,ytr.shape,xte.shape, yte.shape,xval.shape, yval.shape)
```

```
(28709, 2304) (28709,) (3589, 2304) (3589,) (3589, 2304) (3589,)
```

```
#to match the feature array dimension with the model input size, (which is 48,48,1),
#resize the feature 2D arrays into 4D array, i.e. 3D for each image

xtr=np.array(xtr).reshape(28709,48,48,1)
xte=np.array(xte).reshape(3589,48,48,1)
xval=np.array(xval).reshape(3589,48,48,1)

#normalise the pixel values in all feature datasets
xtr=xtr/255
xte=xte/255
xval=xval/255
```

5. Building CNN model

```
#creating the cnn model as a sequential model
model = keras.Sequential()

#FIRST CONVOLUTION LAYER
model.add(layers.Conv2D(64, kernel_size=(3,3), activation='relu', input_shape=(48,48,1)))
model.add(layers.Conv2D(64, kernel_size= (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2)))

#SECOND CONVOLUTION LAYER
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2)))

#THIRD CONVOLUTION LAYER
model.add(layers.Conv2D(512, (3,3), activation='relu'))
model.add(layers.Conv2D(512, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2)))
```

```
#Global MAXPOOLING LAYER
model.add(layers.Flatten())

#FIRST FULLY CONNECTED LAYER
model.add(layers.Dense(512, activation='relu'))

#SECOND FULLY CONNECTED LAYER
model.add(layers.Dense(512, activation='relu'))

#OUTPUT LAYER
model.add(layers.Dense(7, activation="softmax"))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 44, 44, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 512)         590336
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 512)         2359808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
=================================================================
Total params: 4,524,487
Trainable params: 4,524,487
Non-trainable params: 0
_________________________________________________________________
```

```
#Compliling the model
model.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

```
history=model.fit(xtr,ytr, validation_data=(xval,yval), batch_size=64, epochs=100)
```

```
Epoch 1/100
449/449 [==============================] - 12s 28ms/step - loss: 1.8045 - accuracy: 0.2542 - val_loss: 1.7546 - val_accuracy: 0.2666
Epoch 2/100
449/449 [==============================] - 12s 28ms/step - loss: 1.6313 - accuracy: 0.3486 - val_loss: 1.5394 - val_accuracy: 0.3920
Epoch 3/100
449/449 [==============================] - 12s 28ms/step - loss: 1.4286 - accuracy: 0.4468 - val_loss: 1.3906 - val_accuracy: 0.4539
Epoch 4/100
449/449 [==============================] - 12s 28ms/step - loss: 1.2874 - accuracy: 0.5055 - val_loss: 1.2844 - val_accuracy: 0.4946
Epoch 5/100
449/449 [==============================] - 12s 28ms/step - loss: 1.1876 - accuracy: 0.5453 - val_loss: 1.2141 - val_accuracy: 0.5266
Epoch 6/100
```

```
Epoch 92/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0326 - accuracy: 0.9885 - val_loss: 4.8477 - val_accuracy: 0.5681
Epoch 93/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0609 - accuracy: 0.9817 - val_loss: 4.0727 - val_accuracy: 0.5556
Epoch 94/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0542 - accuracy: 0.9837 - val_loss: 4.0911 - val_accuracy: 0.5687
Epoch 95/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0463 - accuracy: 0.9852 - val_loss: 4.5388 - val_accuracy: 0.5464
Epoch 96/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0544 - accuracy: 0.9832 - val_loss: 3.7016 - val_accuracy: 0.5620
Epoch 97/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0430 - accuracy: 0.9855 - val_loss: 4.4900 - val_accuracy: 0.5525
Epoch 98/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0540 - accuracy: 0.9838 - val_loss: 3.9052 - val_accuracy: 0.5614
Epoch 99/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0390 - accuracy: 0.9875 - val_loss: 4.5307 - val_accuracy: 0.5639
Epoch 100/100
449/449 [==============================] - 13s 28ms/step - loss: 0.0463 - accuracy: 0.9853 - val_loss: 4.2598 - val_accuracy: 0.5609
```

```
cnn = model.to_json()
with open("cnn_emrec1.json", "w") as json_file:
    json_file.write(cnn)
model.save_weights("cnn_emrec_weights1.h5")
```

After successful execution of this code, the model is saved as “cnn_emrec1.json” and the weights of the model is saved as “cnn_emrec_weights1.h5” files.
Next we shall implement step 2 of the design.

5.1.2. REAL-TIME EMOTION RECOGNITION AND USER INTERFACE

- Language used: Python version 3.7
- Libraries used:os, numpy, OpenCV, tensorflow, keras
- IDLE used: Jupyter notebook
- Runtime used: CPU runtime

---

1. Import required libraries

```
]:  import os
    import cv2
    import numpy as np
    from keras.models import model_from_json
    from keras.preprocessing import image
```

2. Loading files

```
#loading the cnn model
model = model_from_json(open("./cnn_emrec.json", "r").read())

#loading the weights
model.load_weights("./cnn_emrec_weights.h5")

#loading haar cascade frontal face file
face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

3. Real-time emotion recognition

```
#defining opencv object with path to default web camera
cap=cv2.VideoCapture(0)

#defining emotions in the form of a tuple
emotions = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')

#run an infinite loop till the user ends the application
while True:

    #capture frame  as test_img and returned boolean value as ret (true if image is captured)
    ret,test_img=cap.read()

    if not ret:
        continue

    #convert rgb image into gray scale image
    gray_img= cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
```

```
    #detect all the faces in the gray_img using haar-cascades and store each image as a
    #tuple( x-y coordinate of starting pixel position as x,y , width and height of face as w,h )
    faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.32, 5)


    for (x,y,w,h) in faces_detected: #for each face

        #draw rectangle bounding box in the test_img
        cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)

        #cropping region of interest i.e. face area from image
        roi_gray=gray_img[y:y+w,x:x+h]

        #resize cropped image into 48 x 48
        roi_gray=cv2.resize(roi_gray,(48,48))

        #preprocess image pixel values into an 2-D array
        img_pixels = image.img_to_array(roi_gray)

        #expand dimension of image to match predefined model input size, 48 x 48 x 1
        img_pixels = np.expand_dims(img_pixels, axis = 0)
```

```
        #normalise image pixel values
        img_pixels /= 255

        #perform emotion recognition on the normalised image.
        #the output is a list containing probability values for all classes of emotions.
        #the class containing maximum probability(class-score) is chosen
        predictions = model.predict(img_pixels)

        #find max scored class.
        max_index = np.argmax(predictions[0])

        #This index is used to fetch emotion from the emotion array
        predicted_emotion = emotions[max_index]

        #Put the detected emotion label above the bounding box
        cv2.putText(test_img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

        #each face in the detected image is labelled and contained in respective bounding boxes.

    #after labelling is done on image frame, the image is resized back to formal resolution.
    resized_img = cv2.resize(test_img, (1000, 700))

    #the fimal image is displayed in the user interface window, named: Facial Emotion Recognition
    cv2.imshow('Facial Emotion Recognition ',resized_img)

    #the user interface will be closed when esc or 'q' is pressed. (ASCII of esc=27)
    if cv2.waitKey(10) == 27 or cv2.waitKey(10) == ord('q'):
        break

#release default camera
cap.release()

#destroy user interface window(s)
cv2.destroyAllWindows
```

Primary Takeaways:

1. Model accuracy: 98.53%
2. Model and weights file generated in Step1 are loaded in the second stage in Step2.
3. After every logical fragment of the code, related output is provided for better clarification.

5.2 TESTING

In our project we have implemented both unit as well as integrated testing.

5.2.1: Unit Testing

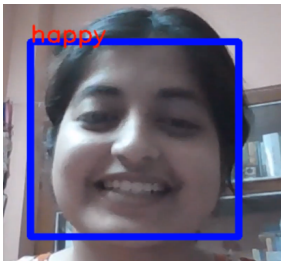
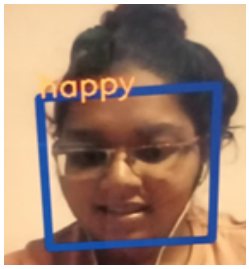
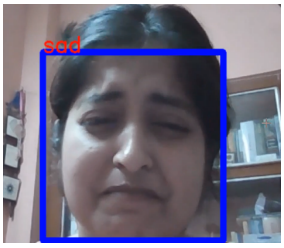
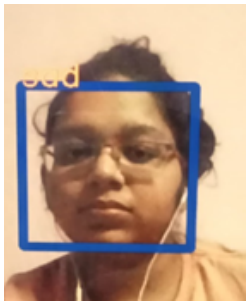
While implementation, each code block is executed and tested. The outputs are provided along with the implementation snippets.

5.2.2: Integrated Testing

The entire model and UI blocks are integrated and displayed in a OpenCv execution window and the following test result is observed.

5.2.2.1 Testing Model Performance

The implementation was tested on all 7 emotions on 3 subjects. The following are the snippets during execution of the program.

Emotions	Subject 1	Subject 2	Subject 3
HAPPY			
SAD			

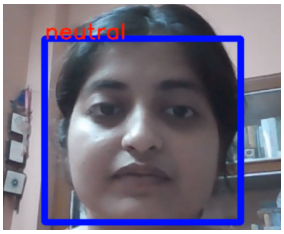
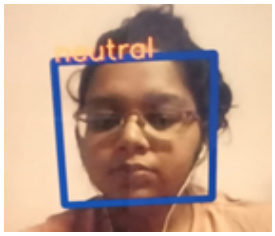
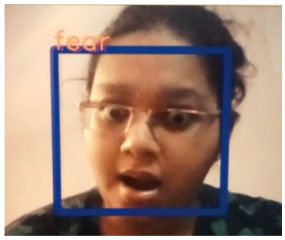
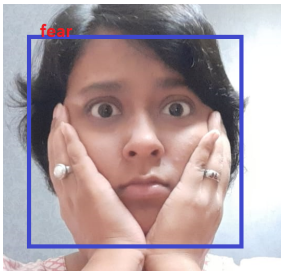
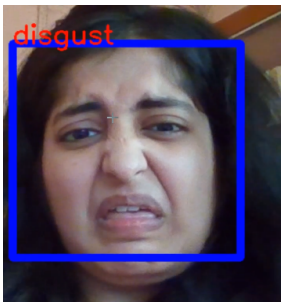
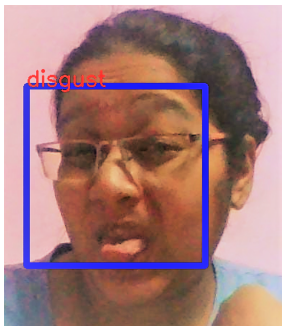
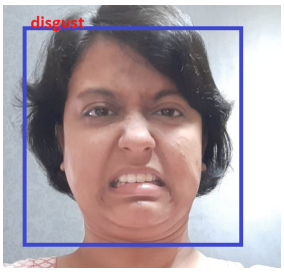
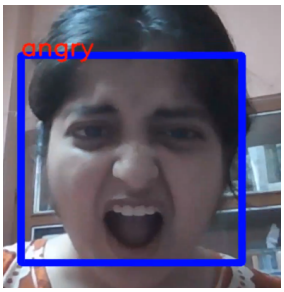
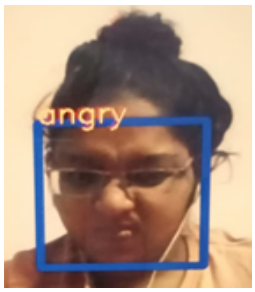
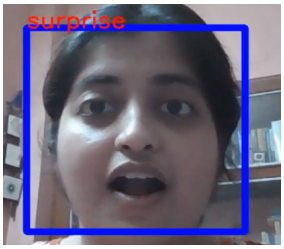
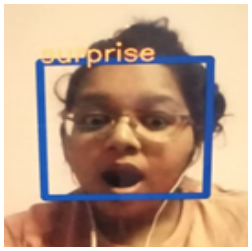
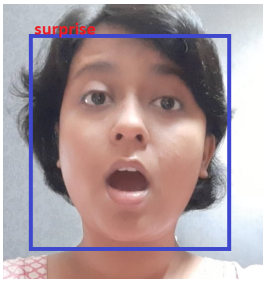
NEUTRAL			
FEAR			
DISGUST			
ANGRY			
SURPRISE			

Table 5.1- Model performance on different emotions of 3 subjects.

5.2.2.2. Testing User Interface

The following are sample screenshots captured during the execution of the code. A few such samples are provided.

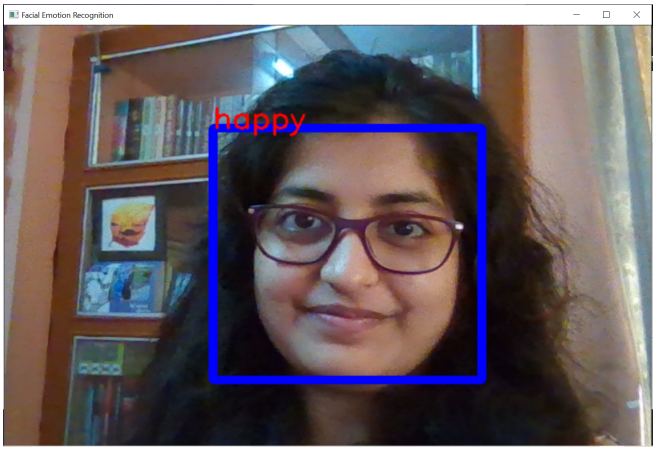


Fig 5.1-User Interface Sample 1

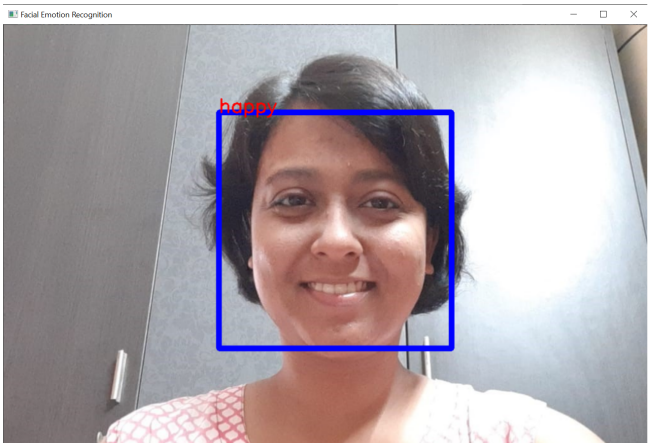


Fig 5.2-User Interface Sample 2

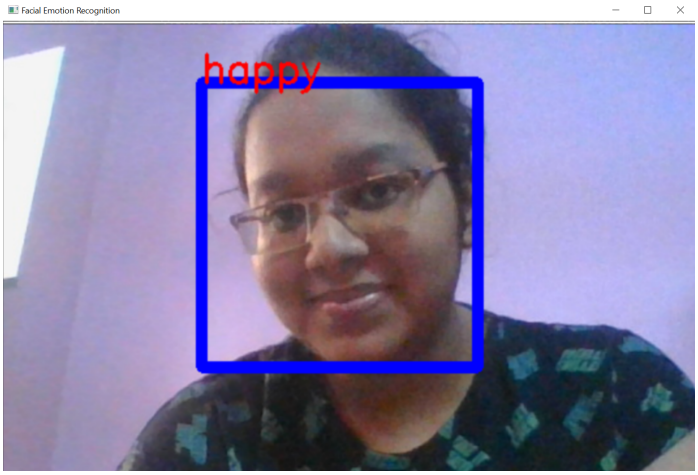


Fig 5.3-User Interface Sample 3

CHAPTER 6

Results and Discussion

6.1. Result

The accuracy of the model was recorded to be 98.53%.

- The result of the model structure is given below.:-

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 44, 44, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 512)         590336
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 512)         2359808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
=================================================================
Total params: 4,524,487
Trainable params: 4,524,487
Non-trainable params: 0
_________________________________________________________________
```

Fig 6.1- Model structure summary

- The result of the User Interface window displaying a emotion is given below:-

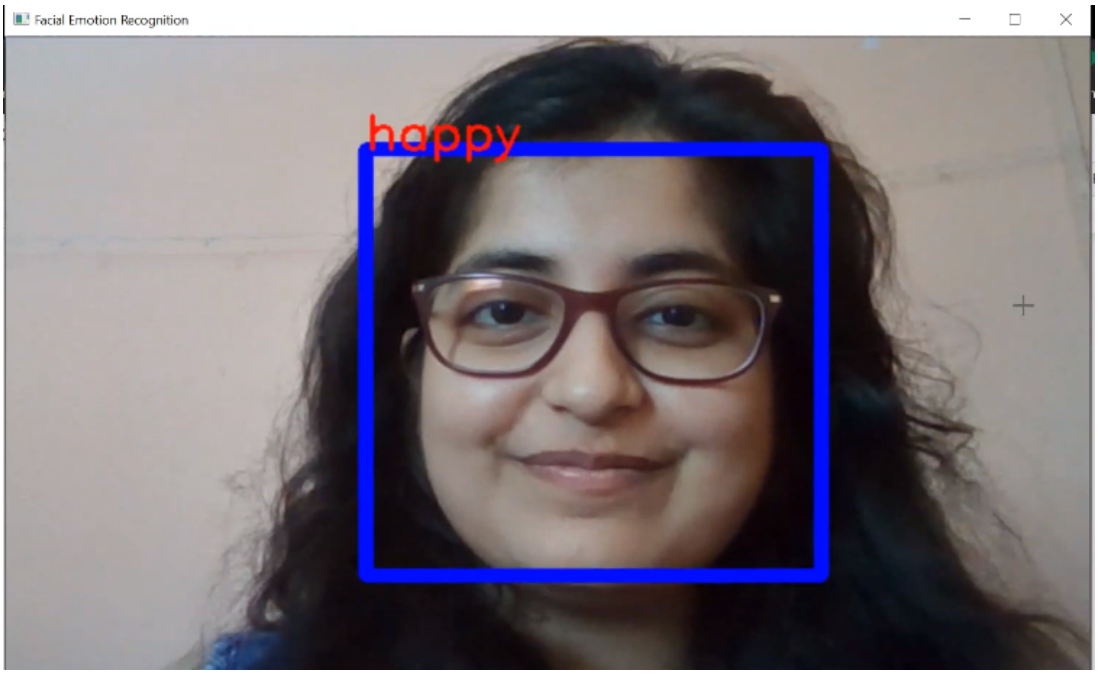


Fig 6.2- User interface window

6.2. Discussion

- The emotions happy, neutral and sad could be identified very well. The emotions disgust could not be identified easily. This is due to the inhomogeneous distribution of the number of images.
- Executing the program with a pre-trained model would require executing the UI code. One has to make sure that the files, cnn_emrec1.json (json model), cnn_emrec_weights1.h5 (the weight file), haarcascade_frontalface_default.xml (haar cascade file) is contained in the same directory as the UI code.
- To change the video path in such a way that the emotion is recognised not from the webcam, but from any pre-recorded video, the UI code can be modified to change the video path from '0' (default- web camera) to the video URL using the cv2.VideoCapture constructor.
- Since emotion is recognised in a frame wise manner, we can implement emotion recognition from image files with minimal changes.

CHAPTER 7

CONCLUSION

7.1. Limitations of the system

During the execution of the system, the following limitations were observed.

- The number of epochs was set to 100, which took a considerable time to converge on GPU system and inconveniently a long time to converge on the CPU system, in other words, if we use a powerful system of high GPU, we get our results quick and accurate which does not make it cost efficient.
- Although increasing the number of epochs helped in making the model more and more accurate, training the model required a lot of time. This is, thus, a trade-off between accuracy and building cost of the model.
- As ‘fear’ and ‘disgust’ emotions have a lesser number of training images, recognising these two emotions is comparatively difficult for the system.
- Emotions cannot be detected for side-face and occlusions due to limitations in the dataset.

7.2. Future scope of the project

Contextual emotion recognition

The project can be extended to identify the mood of a reference frame instead of just identifying the facial emotions in the frame. This would require object recognition, the position and engagement in the reference frame, and optionally a natural language processing on the text or subtitles in the image.

One of the related works on this topic is emotion recognition in context by Kosti, Alvarez and two other scientists at MIT [13]. In their research they have created a multi labelled classification system that shows the percentage of emotions that are felt or expressed in the reference frame. Emotions such as excitement, engagement, anticipation, happiness, yearning ,pleasure, peace, annoyance,esteem,etc. They have used the concept of VAD model that describes the emotions using three numerical dimensions. Valence(V) that measures how positive or pleasant an emotion is, Arousal(A) that measures the activity of the person, Dominance(D) that measures the control level of the person. In every reference frame the set of multi labelled predicted emotions as well as the VAD components are displayed.

7.3. Conclusion

Our project addresses the problem of identifying emotions in the frontal face of a person in real time using a CNN model. The limitations as discussed above can be overcomed using a high-end GPU system and a better dataset containing approximately equal images for all emotions. Even with these limitations, our model was able to achieve an accuracy score of 98.53% on the FER-2013 Dataset.

References

[1] Kołakowska A., Landowska A., Szwoch M., Szwoch W., Wróbel M.R. (2014) Emotion Recognition and Its Applications. In: Hippe Z., Kulikowski J., Mroczek T., Wtorek J. (eds) Human-Computer Systems Interaction: Backgrounds and Applications 3. Advances in Intelligent Systems and Computing, vol 300. Springer, Cham. https://doi.org/10.1007/978-3-319-08491-6_5

[2] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization, http://arxiv.org/abs/1412.6980, (2014).

[3] Emotion Detection from Frontal Facial Image by Sakib Hussain http://hdl.handle.net/10361/2931

[4] Ismail, Nurulhuda & Idayu, Mas & Md Sabri, Mas Idayu. (2009). Review of existing algorithms for face detection and recognition.

[5] Cai, Deng & He, Xiaofei & Hu, Yuxiao & Han, Jiawei & Huang, Thomas. (2007). Learning a Spatially Smooth Subspace for Face Recognition. 1-7. 10.1109/CVPR.2007.383054.

[6] Marchewka, Artur & Zurawski, Lukasz & Jednoróg, Katarzyna & Grabowska, Anna. (2014). The Nencki Affective Picture System (NAPS): Introduction to a novel, standardized, wide-range, high-quality, realistic picture database. Behavior research methods. 10.3758/s13428-013-0379-1.

[7] Riegel, Monika & Moslehi, Abnoos & Michalowski, Jaroslaw & Zurawski, Łukasz & Horvat, Marko & Wypych, Marek & Jednoróg, Katarzyna & Marchewka, Artur. (2017). Nencki Affective Picture System: Cross-Cultural Study in Europe and Iran. Frontiers in Psychology. 8. 10.3389/fpsyg.2017.00274.

[8]Weng, Lilian, Object Detection for Dummies Part 2: CNN, DPM and Overfeat, 2017 , http://lilianweng.github.io/lil-log/2017/12/15/object-recognition-for-dummies-part-2.html, Accessed: 12 Dec 2020

[9]CHI Conference Extended Abstracts on Human Factors in Computing Systems—CHI EA 2020

[10] Friesen, W.V.; Ekman, P.; University of California at San Francisco, San Francisco, CA, USA. EMFACS-7: Emotional Facial Action Coding System, Unpublished manuscript. 1983.

[11] Ekman, P.; Friesen, W.V. Measuring facial movement. Environ. Psychol. Nonverbal Behav. 1976

[12] Viola, Paul & Jones, Michael. (2001). Rapid object detection using a boosted cascade of simple features. Comput. Vis. Pattern Recog. 1.

[13] R. Kosti, J. M. Alvarez, A. Recasens and A. Lapedriza, "Emotion Recognition in Context," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1960-1968, doi: 10.1109/CVPR.2017.212.