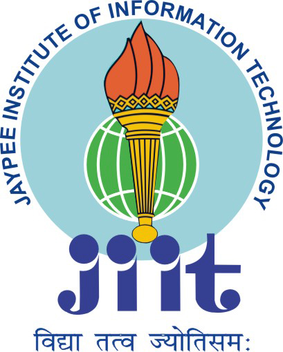
**FACIAL EMOTION RECOGNITION USING CNN**

****

**MINOR PROJECT REPORT**

**SUBMITTED TO**: MR. PAWAN KUMAR UPADYAY

1. APURVA AGGARWAL 16103031 B8 (CSE)
2. SOMIL RASTOGI 16103193 B8 (CSE)
3. AJAY KUMAR KUSHWAHA 16104013 B11 (IT)

**Abstract**

Facial expression recognition has been an active research area in the past ten years, with growing application areas including avatar animation, neuro- marketing and sociable robots. The recognition of facial expressions is not an easy problem for machine learning methods, since people can vary significantly in the way they show their expressions. Even images of the same person in the same facial expression can vary in brightness, background and pose, and these variations are emphasized if considering different subjects (because of variations in shape, ethnicity among others). Although facial expression recognition is much studied in the literature, few works perform fair evaluation avoiding mixing subjects while training and testing the proposed algorithms. Hence, facial expression recognition is still a challenging problem in computer vision. In this work, we propose a simple solution for facial expression recognition that uses a combination of Convolutional Neural Network sand specific image pre-processing steps. Convolutional Neural Networks achieve better accuracy with big data. However, there are no publicly available datasets with sufficient data for facial expression recognition with deep architectures.

Fer2013 is a challenging dataset. The images are not aligned and some of them are incorrectly labeled. Moreover, some samples do not contain faces. This makes the classification harder because the model has to generalize well and be robust to incorrect data. The best accuracy results obtained on this dataset, as far as we know, is 60%.Our proposed CNN architecture is fast to train, and it allows for real time facial expression recognition with standard computers.

**Problem Statement**

Understanding human emotions is a key area of research, as the ability to recognize one’s emotions can give one access to a plethora of opportunities and applications, ranging from friendlier human-computer interactions, to better targeted advertising campaigns, and culminating with an improved communication among humans, by improving the emotional intelligence (“EQ”) of each of us. Emotion being a subjective thing, leveraging knowledge and science behind labeled data and extracting the components that constitute it, has been a challenging problem in the industry for many years. With the evolution of deep learning in computer vision, emotion recognition has become a widely-tackled research problem.

**Research Paper studied**

* Deep Facial Expression Recognition: A Survey By Shan Li and Weihong Deng
  + **Problem Definition -** Recent deep FER systems generally focus on two important issues: over fitting caused by a lack of sufficient training data and expression-unrelated variations, such as illumination, head pose and identity bias.
  + **Methods -** CNN is robust to face location changes and scale variations and behaves better than the multilayer perceptron (MLP) in the case of previously unseen face pose variations, The traditional DBN is built with a stack of restricted Boltzmann machines (RBMs) [114], which are two-layer generative stochastic models composed of a visible-unit layer and a hidden unit layer, DAE was first introduced in to learn efficient codings for dimensionality reduction. DAE is optimized to reconstruct its inputs by minimizing the reconstruction error.
  + **Datasets-** BU-3DFE(The Binghamton University 3D Facial Expression (BU-3DFE) database contains 606 facial expression sequences captured from 100 people.), Oulu-CASIA(The Oulu-CASIA database includes 2,880 image sequences collected from 80 subjects labeled with six basic emotion labels: anger, disgust, fear, happiness, sadness, and surprise.) are among some of the datasets used .
  + **Results - On** the Oulu-CASIA dataset CNN gave an accuracy of 78.31% and DAE gave an accuracy of 81.18%.
  + **Future Work -** The visualization results of CNNs have demonstrated a certain congruity between the learned representations; they can design filters of the deep neural networks to distribute different weights according to the importance degree of different facial muscle action parts.
* Facial Expression Recognition for ‘Wild’ images with analysis from saliency maps by Priyanka Rao and Ling Li.
  + **Problem Definition -** Problem is two-fold. First, seek to understand how and why CNNs misclassify certain images. Second, seek to understand how CNNs transfer their classification ability to “wild” images when trained on grayscale pre-processed images.
  + **Methods-** used several different architectures – a shallow 3-layer CNN, Alexnet, VGG-16, Inception, and Inception-Resnet. For the existing architectures, we train them on the Kaggle dataset and benchmark their performance against their performance when pre-trained on ImageNet. The architecture for the three-layer CNN is [conv - relu - 2x2 max pool] – [affine – relu] – [affine] [15]. For loss, they used the softmax function because they want their model to try to increase its output probability of the right emotion and decrease its output probabilities for the wrong emotions.
  + **Datasets-** used a dataset from the Kaggle “Challenges in Representation Learning: Facial Expression Recognition Challenge” competition. The provided dataset for this challenge consists of 35887 48 × 48 pixel grayscale images.
  + **Results-** used a learning rate of 1e-4 for all four models and got a test accuracy of 51%, 49.4%, 46.3% and 45.7% for Alexnet, VGG-16, Inception and Inception-Resnet respectively.
  + **Future Work-** To improve the accuracy on wild images, we need to further incorporate a dynamic searching algorithm to move the active detection bound box around or do a face segmentation first.
* Facial Emotion Detection Using Convolutional Neural Networks and Representational Autoencoder Units by Prudhvi Raj Dachapally
  + **Problem Definition-** With the transition of facial expression recognition (FER) from laboratory-controlled to challenging in-the-wild conditions and the recent success of deep learning techniques in various fields, deep neural networks have increasingly been leveraged to learn discriminative representations for automatic FER.
  + **Methods-** The first method uses Autoencoder to construct a unique representation of each emotion, while the second method is an 8-layer convolutional neural network (CNN). These methods were trained on the posed-emotion dataset (JAFFE), and to test their robustness, both the models were also tested on 100 random images from the Labeled Faces in the Wild (LFW) dataset, which consists of images that are candid than posed.
  + **Datasets-** Labeled Faces in the Wild (LFW) dataset and the Japanese Female Facial Expression (JAFFE) database which has 215 images of 10 different female models posing for 7 emotions. Seventy-five percent of this dataset was used for training, and the rest for testing.
  + **Results-** For RAU 48.57% in LFW test set and 59.25% for JAFFE test set, While CNN gave 86.38% on Jaffe test set and 67.62% on LFW test set.
  + **Future Work-** The results show that with more fine-tuning and depth, our CNN model can outperform the state-of-the-art methods for emotion recognition. They want to work more on the visualizing the learned filters in depth, and in the future, we also want to take a semi-supervised approach by using the predictions made for the LFW images, to train the network with more data, more filters, and more depth.
* Recognizing Facial Expressions Using Deep Learning by Alexandru Savoiu and James Wong.
  + **Problem Definition-** In this project they applied various deep learning methods (convolutional neural networks) to identify the key seven human emotions: anger, disgust, fear, happiness, sadness, surprise and neutrality.
  + **Methods-** The architectures we employed for our convolutional neural networks were VGG-16 and ResNet50. We used the support vector machine multiclass classifier as our baseline, which had an accuracy performance of 31.8%. To further improve our results, we leveraged ensemble and transfer learning techniques to achieve our best results. Thus, the accuracy using ensemble learning was 67.2% and with transfer learning was 78.3%.
  + **Datasets -** The Kaggle dataset (from the Facial Expression Recognition Challenge) meeting all the following attributes: 35,887 images ,Image Format: 48 x 48 pixels (8-bit grayscale)and Karolinska Directed Emotional Faces key attributes are: 4900 images , Image Format: 562 x 762 (32-bit RGB) , 70 individuals, each displaying the seven different emotional expressions, and each expression is photographed twice from five different angles
  + **Results -** On the Kaggle dataset, our baseline SVM accuracy was 31.8% while VGG-16 and ResNet50 had accuracies of 59.2% and 65.1%. The overall accuracies along with precision and recall on the KDEF dataset are greater than those on the Kaggle dataset. SVM achieved an accuracy of 37.9% while VGG16 and ResNet50 achieved accuracies of 71.4% and 73.8%.
  + **Future Work -** To further improve model performance, they wish to explore adding various facial and image features. They would also like to explore recognizing emotions in color images and to perform these predictions across the duration of a video.
* Facial Expression Recognition Using Deep Convolutional Neural Networks by Dinh Viet Sang, Nguyen Van Dat and Do Phan Thuan.
  + **Problem Definition -** Facial expressions convey non-verbal information between humans in face-to-face interactions. Automatic facial expression recognition, which plays a vital role in human-machine interfaces, has attracted increasing attention from researchers since the early nineties. Classical machine learning approaches often require a complex feature extraction process and produce poor results.
  + **Methods -** Architectures are composed of convolution layers, pooling layers, and fully connected layers. VGG is characterized by its simplicity, using only small convolutional layers stacked on top of each other in increasing depth. Unlike conventional CNNs, VGG consists of blocks, each of which contains one to four convolutional layers. The convolutional layers often have a very small filter size of 3×3, and the stride is 1. The convolutional layers in the same block have the same number of filters. And the number of filters in each convolutional layer in this block is double or equal to the number of filters in each convolutional layer of the previous block.
  + **Datasets -** The Kaggle dataset (from the Facial Expression Recognition Challenge) meets all the following attributes: 35,887 images, Image Format: 48 x 48 pixels (8-bit grayscale), various individuals across the entire spectrum of: ethnicity, race, gender and race, with all these images being taken at various angles.
  + **Results -** High accuracies are obtained with happy (89.76%), surprised (82.45%), and neutral (73.16%) classes. In fact, they are the most distinguishable emotions for human. The angry, fearful, sad classes are more often confused together, since they share many similar expressions. Finally, the disgusted class gets a pretty good accuracy 69.09%, despite the low amount of disgusted samples in the training set.
  + **Future Work -** In their work for this project, we trained the models using a pre-processed version of the raw image pixels. . They would also like to explore recognizing emotions in color images and to perform these predictions across the duration of a video.

**Algorithms/ Approach used and Implemented**

We created and deployed our own Convolution Neural Network Architecture for performing image classification task. Our task basically consisted of recognition of emotion from human images and Deep CNN architecture has been found to accomplish the task quite effectively. Moreover various researches have proved CNN to be an effective method for carrying out image classification jobs.

Firstly a brief introduction about CNN and its various layers including how a General CNN works, followed by our architectures which we have applied in order to attain our objective are mentioned below.

CNN primarily consists of 4 general layers whose numbers can be varied for making our model deeper, for example the VGG (Visual Geometry Group developed for ImageNet dataset) has a total of 16 or 19 layers which involve increased number of either convolution layers or fully connected dense layers.

Brief Introduction on the layers of CNN

* **Convolutional Layer** – When our brain looks at an image, it looks for features and depending upon the features it sees, our brain categorises the image in certain ways. For input to our CNN we have images which can be either coloured or gray scale depending upon our requirements. These images are actually multidimensional arrays containing pixel values ranging between 0-255. For gray scale images there is one channel and the range 0-255 represents 0 for completely black pixel and 255 for completely white pixel. For coloured images there are 3 channels RGB where the values 0-255 represent the pixel intensity of the colours.

In our Convolution layers we have what we call as feature detector/filters, which is a fixed size square matrix (let’s say 3X3).We take these filters, put it on our input image and multiply each value of filter (0/1) by the value it covers(0/1).We perform element wise multiplication and add up the result. Then we slide the filter and the rate at which we slide the filter is called stride, and perform the same task.

Feature Detector/filter represents a certain pattern in it. On multiplication of a filter by image matrix values, if we get a significant number, it indicates that pattern matches up. Filters actually help us in preserving features and getting rid of unnecessary features.

We create multiple feature maps/filters as we use different filters to preserve different information. So we don’t have just one filter but fixed number which helps our CNN to decide which features are important for certain types, through its training.

* **ReLU layer** – Upon the result we obtain from our convolutional layer we apply our rectifier function as we want to increase **NON LINEARITY** in our image. The first question that arises is why do we want to increase non linearity in our image? The answer to that is image are themselves quite non linear as transition between pixels of adjacent rows and columns do not have any fixed pattern of colour change. But at the same time, from the convolution operation we risk that we might create something linear so we need to break up the linearity, which is done through relu layers. To understand let’s take an example: In the gray scale image, upon producing the feature map we see some areas going from white to gray to black, kind of a linear fashion, so what ReLU does is it removes the black part pixels and the linearity is broken while maintaining the features of the image.
* **Pooling layer** – It solves the problem of image recognition when the images aren’t oriented properly or are textured differently. If our CNN looks for features that are in the image, then it will recognise correctly, but if it looks for features that are absent in some, it will misrecognise it.

So we have to ensure a property called **Spatial Invariance**, which means our CNN doesn’t care much about the orientation or positioning of features but is rather concerned for whether the features are actually present or not.

To obtain the above property the work of pooling layers come into play.

Pooling layer is taking a fixed size matrix (eg:2X2) and moving on the convoluted feature map and only recording its maximum value(max pooling) or average value(average pooling), ultimately disregarding the 3 values and passing just 1 value to the forward layers. We were however, still able to preserve the features as the highest number we forwarded actually contained the pixel.

By pooling we are also able to reduce the overall size of the image which would ultimately be flattened and sent to further layers. We are also able to reduce the number of parameters going into the final layer, thus preventing over fitting, exactly like our brain which excludes the not so necessary features.

* **Flattening layer** – After we apply convolution operation to our input and apply pooling to the result, we get the pooled feature map, which we are just going to flatten into a column having values row by row put into columns as we want to forward it to the fully connected layer. For many pooled features, it gets transformed to a one long column vector.
* **Fully Connected layer** – This Step is actually adding up a whole artificial neural network to complete our CNN. The whole vector obtained from the flattening layer is passed to the input layer of ANN. Main purpose of doing so is to combine our features into more attributes which can predict the class even better. Cost function in ANN (MSE) is called as loss function in CNN and for that we use a cross entropy function. However our goal is same, minimizing the loss function to optimize our network. So the error here is also back propagated through the network. Weights and feature maps are adjusted while training ultimately increasing the performance. All of these operations are done through Gradient Descent Algorithm with Back Propagation.

How does the Fully Connected layer actually works-?

First we need to understand what weights to assign to a particular class and know which of the previous neurons are actually important for a particular class. So in our last fully connected layer we have a value between 0-1 where 1 means that neuron is confident to have found the feature whereas 0 means the neuron didn’t find the feature. After many iterations the class will recognise the neurons that fire up when found features belonging to the particular class, while other classes will also learn to ignore the particular features belonging to other classes. With much iteration, a feature that is useless to the network is going to be disregarded and replaced with the features that are useful.

When a new testing image is passed, the neuron values to all specific classes are passed and based upon the values of the probabilities passed we make the final prediction.

**Architecture of the CNN we have utilized in our work-**

We had a total of 35k images dataset consisting of 7 emotions, namely Anger, Fear, Disgust, Sad, Happy, Surprise and Neutral. But since our dataset consisted of very small number of disgust images we merged the disgust images with anger. Our dataset was also highly unbalanced with number if images of the class happy to a very large number. So to make our dataset balanced, we utilized the concept of over-sampling i.e. creating of copies of images of respected emotions to balance the number of emotions in each class. All this happened in our csv file format consisting the pixel values of images but then as we are using CNN we send in the images as direct input using flow\_from\_Directory. So we converted the balanced CSV file format to images and then sent the images as inputs to CNN.

We first started our work on 3 emotions namely Fear, Happy and Neutral. We first tried to create our CNN using only three emotions, we took three emotions as ours is a multiclass problem and starting with two emotions is merely binary classification.

1. We first created **ARCHI\_1** a deep network consisting of –

* 1 convolutional layer consisting of 32 filters of size (3X3) with a stride of 1 and activation function as SOFTMAX.
* 1 pooling layer involving maxpooling with a stride of 2 and pool size of (2X2).
* 1 flatten layer to flatten the result from pooling layers
* 1 fully connected layer consisting of 128 neurons and activation function as SOFTMAX.
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 3 and activation function as SOFTMAX.

We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and trained the CNN on complete dataset. We obtained accuracy of about 41% from above model. We then decided to increase the accuracy by making our model deeper, increasing the number of convolutional layers and fully connected layers.

1. We then Created **ARCHI\_2** a deep network consisting of-

* 3 convolutional layers consisting of 32 filters of size (3X3) with a stride of 1 and activation function as SOFTMAX.
* 1 pooling layer involving **maxpooling** with a stride of 2 and pool size of (2X2).
* 1 flatten layer to flatten the result from pooling layers
* 1 fully connected layer consisting of 128 neurons and activation function as SOFTMAX
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 3 and activation function as SOFTMAX.

We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and training the CNN on complete dataset. We obtained accuracy of about 62% from above model.

1. We then Created **ARCHI\_3** a deep network consisting of-

* 3 convolutional layers consisting of 32 filters of size (3X3) with a stride of 1 and activation function as SOFTMAX.
* 1 pooling layer involving **averagepooling** with a stride of 2 and pool size of (2X2).
* 1 flatten layer to flatten the result from pooling layers
* 1 fully connected layer consisting of 128 neurons and activation function as SOFTMAX.
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 3 and activation function as SOFTMAX.

We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and training the CNN on complete dataset.

We obtained accuracy of about 56% from above model. Hence inferring that maxpooling performs better in our case. Now our model was working with nearly expected accuracy so we tried to increase our number of classes and also applied some of the image\_augmentation techniques as upon doing some research we came to the conclusion that heavy image augmentation increases the accuracy.

Also we did the above mentioned work on an unbalanced dataset which contained a high proportion of images belonging to “happy” class, clearly indicating that the accuracy obtained was because of high overfitting. We balanced the dataset by oversampling and on running the 3 architectures mentioned above, received an accuracy of 30 to 45%. The accuracy clearly fell.

1. We then Created **ARCHI\_4** a deep network consisting of-

* 3 convolutional layers consisting of 32 filters of size (3X3) with a stride of 1 and activation function as SOFTMAX.
* 1 pooling layer involving Maxpooling with a stride of 2 and pool size of (2X2).
* 1 flatten layer to flatten the result from pooling layers
* 1 fully connected layer consisting of 128 neurons and activation function as SOFTMAX
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 6 and activation function as SOFTMAX.

We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and training the CNN on complete dataset.

We obtained accuracy of about 36% from above model. Worried on the performance we started doing even more research and came across a deeper neural network architecture in a research paper and applied it to our 6 emotions.

1. We then Created **ARCHI\_5** a deep network consisting of-

* 1 convolutional layer consisting of 64 filters of size (5X5) with a stride of 1.
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 pooling layer involving Maxpooling with a stride of 2 and pool size of (5X5).
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 convolutional layer consisting of 64 filters of size (3X3) with a stride of 1.
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 convolutional layer consisting of 64 filters of size (3X3) with a stride of 1.
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 pooling layer involving Averagepooling with a stride of 2 and pool size of (3X3).
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 convolutional layer consisting of 128 filters of size (3X3) with a stride of 1.
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 convolutional layer consisting of 128 filters of size (3X3) with a stride of 1.
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 Zero padding layer to add zeroes in the input image in the border of image.
* 1 pooling layer involving Averagepooling with a stride of 2 and pool size of (3X3).
* 1 flatten layer to flatten the result from pooling layers
* 1 fully connected layer consisting of 1024 neurons
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 dropout layer which randomly deactivates passed probability neurons to avoid overfitting. Dropout prob as 0.2.
* 1 fully connected layer consisting of 1024 neurons
* 1 Parametric RELU layer as it has small slope for negative values instead of altogether 0 as in Relu.
* 1 dropout layer which randomly deactivates passed probability neurons to avoid overfitting. Dropout prob as 0.2.
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 6, with activation as Softmax.

We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and training the CNN on complete dataset.

We obtained accuracy of about 58% from above model. Seemingly happy we tried to test it on real time dataset and that’s when it showed its poor performance. The CNN seems to have been heavily overtrained on two of the emotions namely angry and sad and generally classifies every emotion as those two emotions only.

With further research we found out that to reduce overfitting we could use two techniques namely **Batch Normalization** and Adding of **Dropout** layers.

**Batch Normalization** - To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

**Dropout** - Dropout is a technique used to tackle overfitting. The Dropout method in keras.layers module takes in a float between 0 and 1, which is the fraction of the neurons to drop.

We also came across a paper which validated our suspicion of poor performance of ARCHI\_4 due to heavy image augmentation. It turns out image augmentation is necessary but the techniques we were using were actually decreasing the accuracy.

1. Finally we created our Final architecture which is **ARCHI\_6** a deep network consisting of-

* 1 convolutional layer consisting of 64 filters of size (3X3) with a stride of 1.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 pooling layer using MaxPooling wit filter size and stride of 2.
* 1 dropout layer with dropout of 0.25.
* 1 convolutional layer consisting of 128 filters of size (5X5) with a stride of 1.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 pooling layer using MaxPooling wit filter size and stride of 2.
* 1 dropout layer with dropout of 0.25.
* 1 convolutional layer consisting of 512 filters of size (3X3) with a stride of 1.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 pooling layer using MaxPooling wit filter size and stride of 2.
* 1 dropout layer with dropout of 0.25.
* 1 convolutional layer consisting of 512 filters of size (3X3) with a stride of 1.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 pooling layer using MaxPooling wit filter size and stride of 2.
* 1 dropout layer with dropout of 0.25.
* 1 flatten layer to flatten the result from pooling layers.
* 1 fully connected layer consisting of 256 neurons.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 dropout layer with dropout of 0.25.
* 1 fully connected layer consisting of 512 neurons.
* 1 BatchNormalization layer.
* 1 Activation layer with activation function as RELU.
* 1 dropout layer with dropout of 0.25.
* 1 fully connected layer consisting the number of neurons equal to number of classes we had that is 6, with activation as Sigmoid.

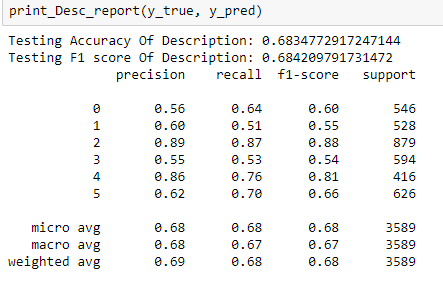
We compiled our CNN using ADAM optimizer with loss function as categorical\_crossentropy and evaluation metrics as accuracy, Batch\_size as 32 and training the CNN on complete dataset.

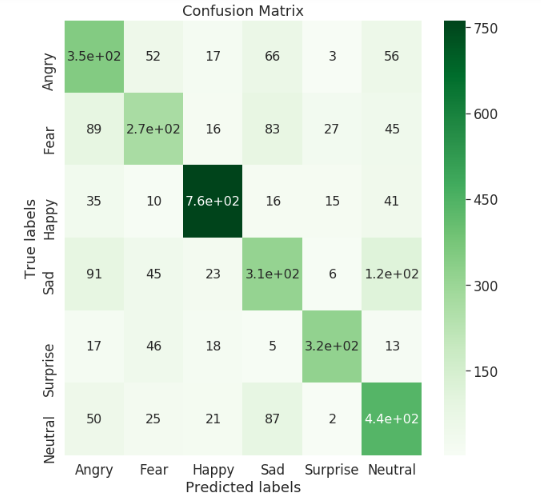
We obtained a test set accuracy of 68% and validation set accuracy around 66%.So we see not only our accuracy increased from the previous all models but also when tested on new images it gave fairly decent results.

We came to the conclusion that our neural network architecture of ARCHI\_6 is the best and can be used to obtain pretty good results.

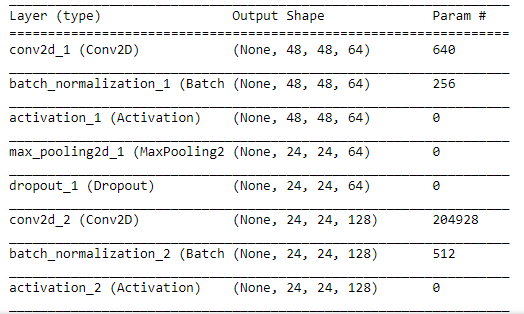
**Results**

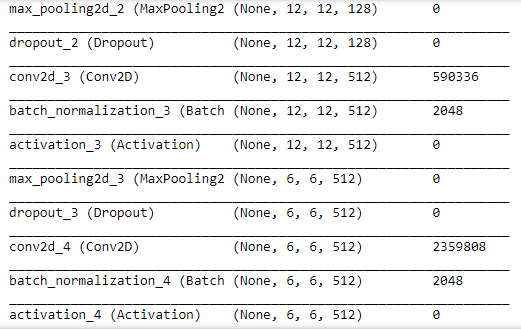
In our self made CNN architecture for emotion classification into 6 categories (namely Anger, Happy, Fear, Sad, Surprise and Neutral), an accuracy of 68.55% was achieved.

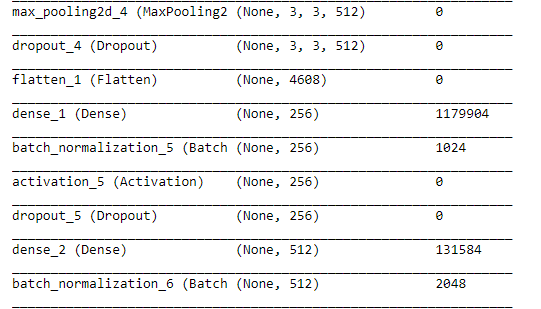


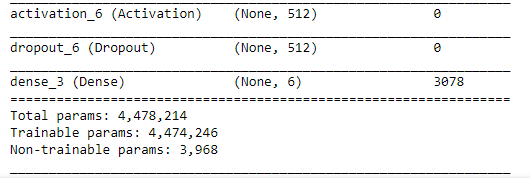


**Model Summary of Our Architecture**

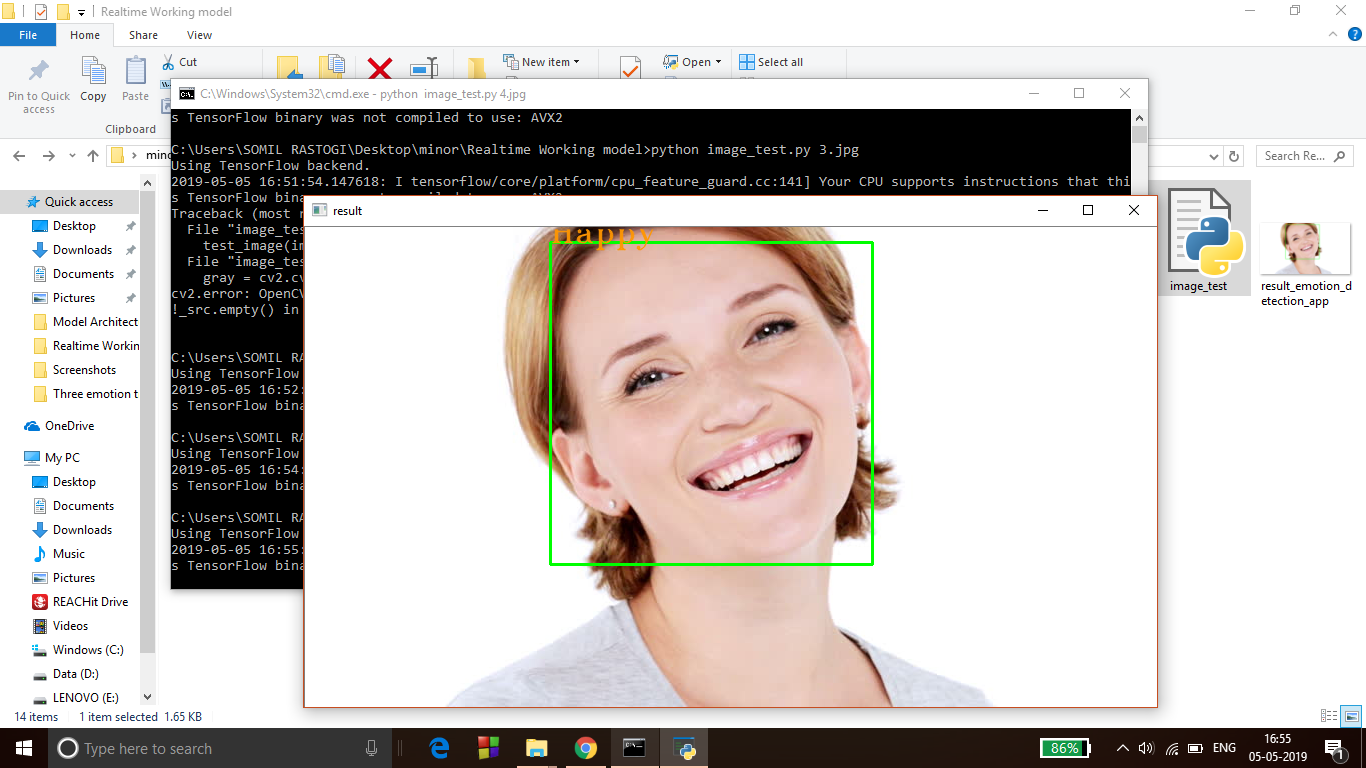


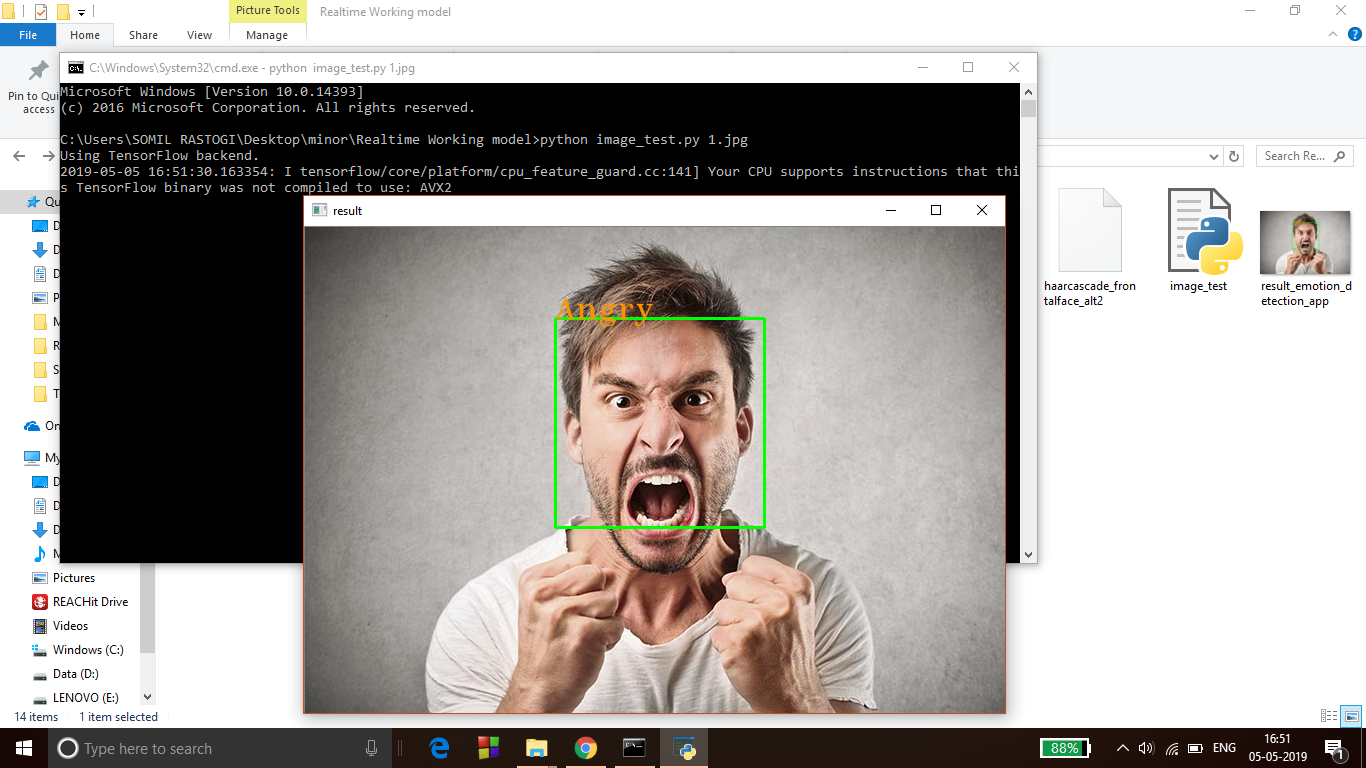


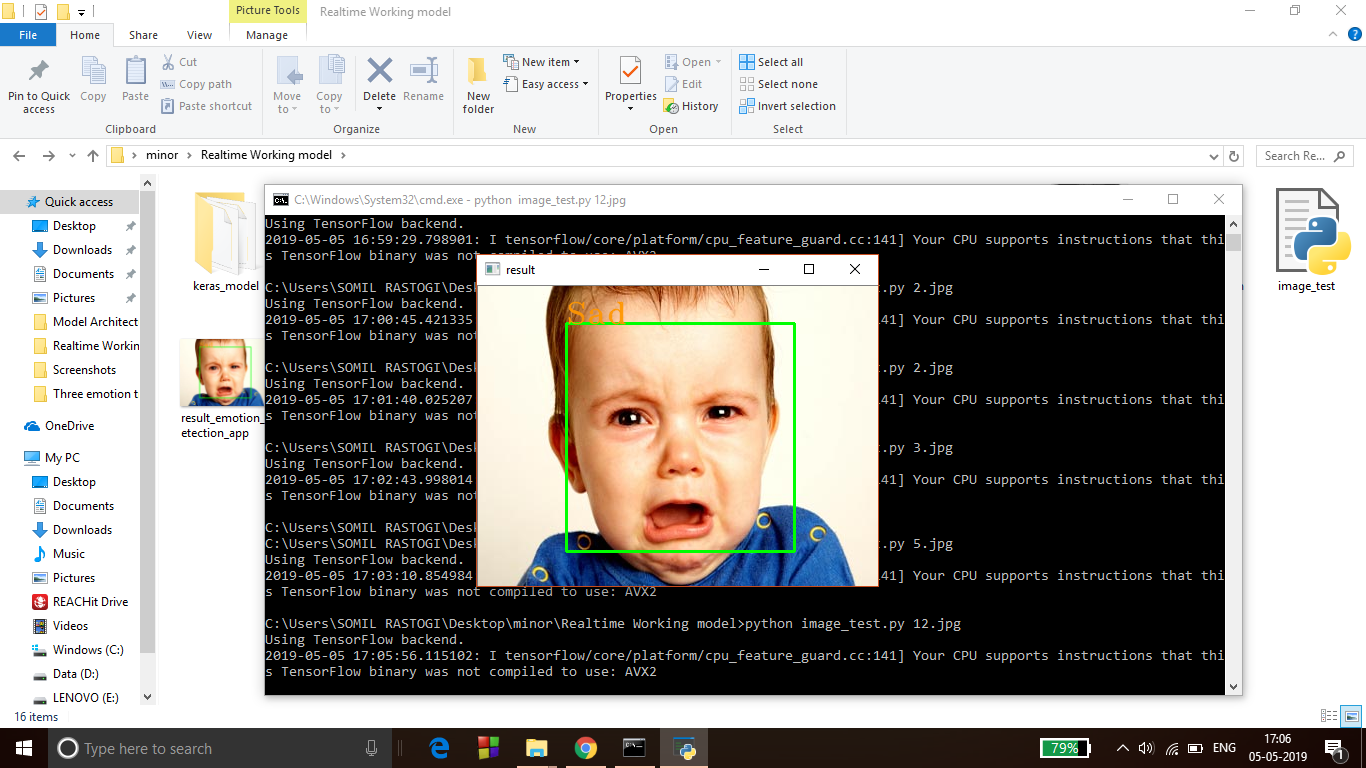


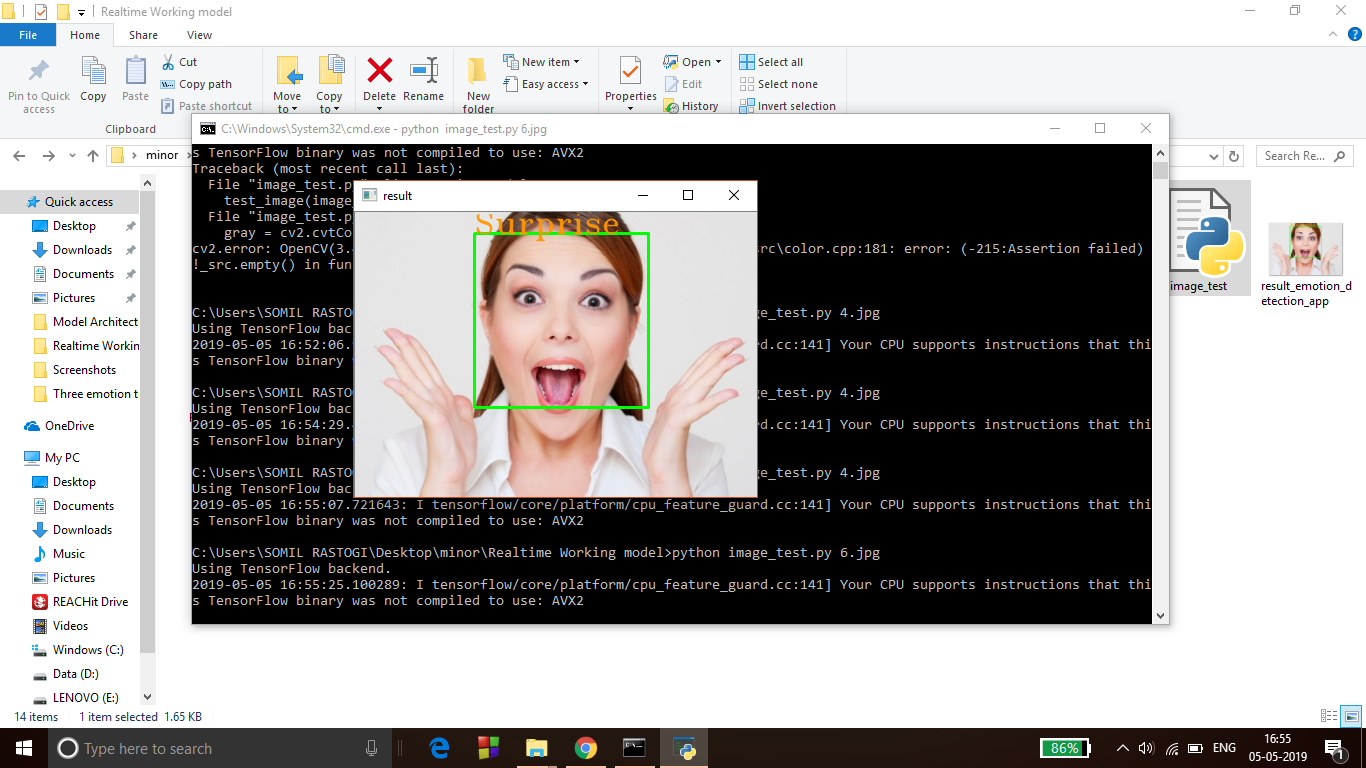


**Result screenshots**

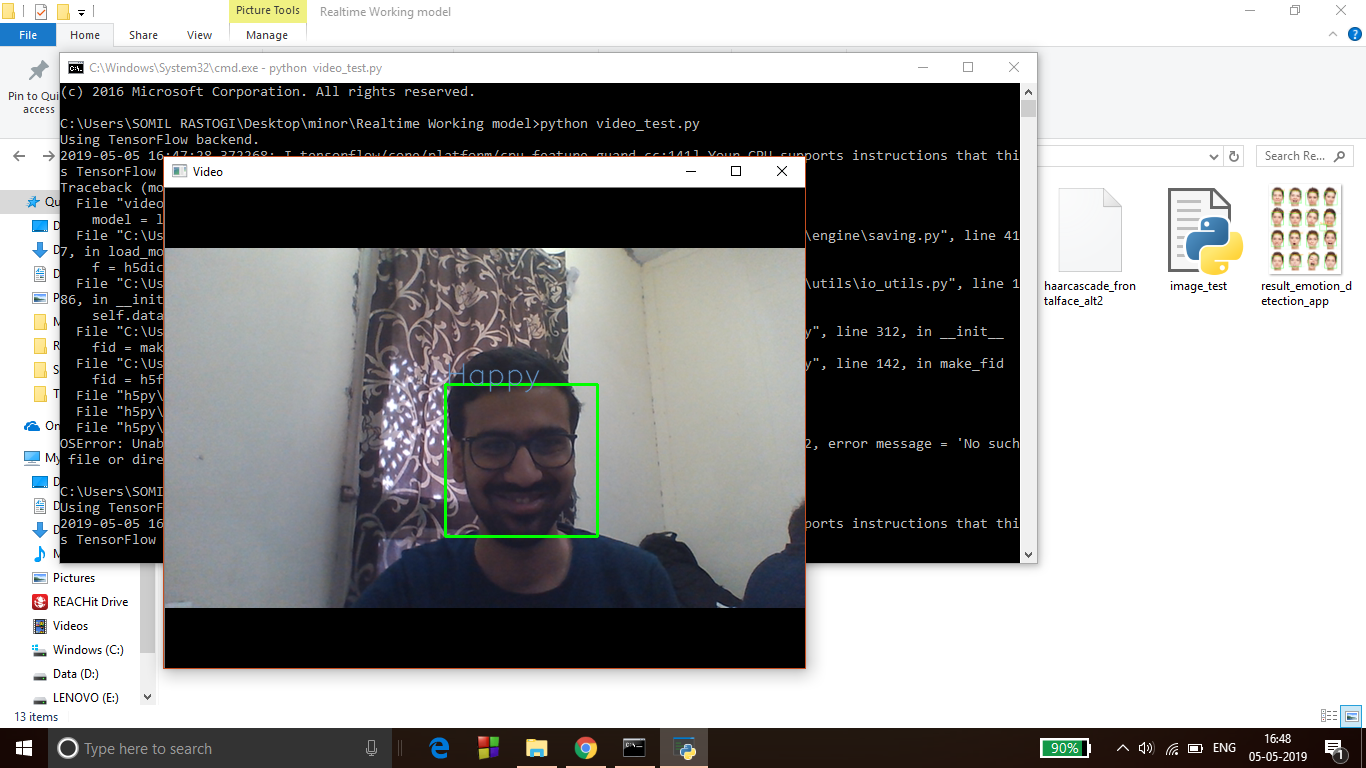








**Real time emotion detection**



**References**

* Raj Dachapally, Prudhvi. (2017). Facial Emotion Detection Using Convolutional Neural Networks and Representational Autoencoder Units.
* Savoiu, A., & Wong, J. Recognizing Facial Expressions Using Deep Learning.
* Sang, Dinh & Van Dat, Nguyen & Phan Thuan, Do. (2017). Facial expression recognition using deep convolutional neural networks. 130-135. 10.1109/KSE.2017.8119447.
* Matsugu, M., Mori, K., Mitari, Y., & Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. Neural Networks, 16(5-6), 555-559.
* Ma, L., & Khorasani, K. (2004). Facial expression recognition using constructive feedforward neural networks. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(3), 1588-1595.
* Wu, J. (2017). Introduction to Convolutional Neural Networks.
* Kuo, C.-C. Jay. “Understanding Convolutional Neural Networks with a Mathematical Model.” J. Visual Communication and Image Representation 41 (2016): 406-413.