LINKING FACIAL EXPRESSIONS TO EMOTICONS

A Project Report

submitted by

Abhilasha Jha	15BCE0867
Amanat Dhillon	15BCE2025
Dhanush Samdaria M	15BCE0349
Jay Paresh Mehta	15BCE0166
Payal Pankhuri	15BCE0065

CSE4014 High Performance Computing

Slot-C1+TC1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

OCTOBER, 2018

DECLARATION

We hereby declare that the Industrial Internship report entitled **“Linking Facial Expressions To Emoticons”** was submitted by Abhilasha Jha, 15BCE0867, Amanat Dhillon, 15BCE2025, Dhanush Samdaria M, 15BCE0349, Jay Paresh Mehta, 15BCE0166, Payal Pankhuri, 15BCE0065, to Vellore Institute of Technology, Vellore in partial fulfilment of the requirement for the award of the degree of **B.Tech** in **Computer Science & Engineering** is a record of bonafide project work done by us under the supervision **Dr. Narayanan Prasanth**. We further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

CERTIFICATE

This is to certify that the project work entitled "**Linking Facial Expressions To Emoticons**" that is being submitted by Abhilasha Jha, 15BCE0867, Amanat Dhillon, 15BCE2025, Dhanush Samdaria M, 15BCE0349, Jay Paresh Mehta, 15BCE0166, Payal Pankhuri, 15BCE0065 for High Performance Computing is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place : Vellore

Date :

Signature of students Signature of faculty

ACKNOWLEDGEMENT

On this opportunity we express our profound gratitude and deep regards to **Dr. Narayanan Prasanth** for his exemplary guidance, monitoring and constant encouragement throughout the internship.

The blessing, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark. Without his surveillance and direction my internship wouldn't have been successful.

Lastly, we thank almighty, our parents and our teachers for their constant encouragement without which this assignment would not have been possible

Place : Vellore

Date :

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The purpose of this project is to identify various expressions of the human face using convolutional neural networks and linking them to various emoticons as the outcome. In the given project, we also optimize our algorithm to work much faster with GPU. The libraries which will be used to create the project are TensorFlow, Keras and OpenCV.

1. INTRODUCTION

To enhance the experience of communication emoticons were developed. Emoticons are the pictorial depiction of the facial expression of human beings. They are very helpful in the facilitation of human emotional experiences. The project includes eleven human expressions namely neutral, happy, sad, wink, kiss, surprised, angry, monkey face, wink with tongue out, scared, disgusted. These expressions are the actual expressions conveyed in human beings. The purpose of this project is to identify various expressions of the human face using convolutional neural networks and link them to various emoticons as the outcome. We optimize our algorithm to work much faster with GPU. The libraries used to create the project are TensorFlow, Keras and OpenCV.

2. LITERATURE SURVEY

S.NO	PAPER TITLE	YEAR(PUBLICATION)	AUTHOR	SUMMARY
1	Emotion Recognition in the Wild via Convolutional Neural Networks and Mapped Binary Patterns	Proc. ACM International Conference on Multimodal Interaction (ICMI), November 2015.	G. Levi and T. Hassner	This paper presents a novel method for classifying emotions from static facial images. It leverages on the recent success of Convolutional Neural Networks (CNN) on face recognition problems. The method used is designed with the goal of simplifying the problem domain by removing confounding factors from the input images, with an emphasis on image illumination variations, in an effort to reduce the amount of data required to effectively train deep CNN models. These are applied to CASIA Webface images which are then used to train an ensemble of

				multiple architecture CNNs on multiple representations. Each model is then fine-tuned with limited emotion labeled training data to obtain final classification models. This method has proved to provide an improvement of 15.36% over baseline results.
2	Capturing au-aware facial features and their latent relations for emotion recognition in the wild	Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, ICMI '15	A. Yao, J. Shao, N. Ma, and Y. Chen	This paper analyzes the relations among expression-specific facial features in an explicit manner. It includes a 3 pronged approach. The first step is a pair-wise learning strategy to automatically seek a set of facial image patches which are important for discriminating two particular emotion categories. Second step uses an undirected graph structure, which takes learnt facial patches as individual vertices, to encode feature relations between any two learnt facial patches. Finally, a robust emotion representation is constructed by concatenating all task-specific graph-structured facial feature relations sequentially. The final result obtained had a 55.38% recognition accuracy outperforming the baseline .

3.	Hierarchical committee of deep convolutional neural networks for robust facial expression recognition	Journal on Multimodal User Interfaces	B. Kim, J. Roh, S. Dong, and S. Lee,	This paper includes training multiple deep convolutional neural networks (deep CNNs) as committee members and combining their decisions. To improve this committee of deep CNNs, two strategies are used : (1) in order to obtain diverse decisions from deep CNNs, the network architecture is varied, input is normalized, and random weight initialization is undergone , and (2) in order to form a better committee in structural and decisional aspects, a hierarchical architecture of the committee is constructed with exponentially-weighted decision function. On public databases, the hierarchical committee of deep CNNs yields superior performance, outperforming or competing with state-of-the-art results for these databases.

4.	Image based static facial expression recognition with multiple deep network learning	Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, ICMI '15,	Z. Yu and C. Zhang	This paper seeks to automatically classify a set of static images into 7 basic emotions. The proposed method contains a face detection module based on the ensemble of three state-of-the-art face detectors, followed by a classification module with the ensemble of multiple deep convolutional neural networks (CNN). Each CNN model is initialized randomly and pre-trained on a larger dataset. The pre-trained models are then fine-tuned on the training set. To combine multiple CNN models, the paper uses two schemes for learning the ensemble weights of the network responses: by minimizing the log likelihood loss, and by minimizing the hinge loss. The proposed method generates state-of-the-art result on the respective dataset.

5.	The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specifi ed expression	2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops	Patrick Lucey, Jeffrey F. Cohn, Takeo Kanade, Jason M. Saragih, Zara Ambadar, Iain A. Matthews	
6.	Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark	2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)	Abhinav Dhall, Roland Göcke, Simon Lucey, Tamás D. Gedeon	Quality data recorded in varied realistic environments is vital for effective human face related research. Currently available datasets for human facial expression analysis have been generated in highly controlled lab environments. We present a new static facial expression database Static Facial Expressions in the Wild (SFEW) extracted from a temporal facial expressions database Acted Facial Expressions in the Wild (AFEW) [9], which we have extracted from movies. In the past, many robust methods have been reported in the literature. However, these methods have been experimented on different

				databases or using different protocols within the same databases. The lack of a standard protocol makes it difficult to compare systems and acts as a hindrance in the progress of the field. Therefore, we propose a person independent training and testing protocol for expression recognition as part of the BEFIT workshop. Further, we compare our dataset with the JAFFE and Multi-PIE datasets and provide baseline results.

7.	Robust facial expression recognition using local binary patterns	IEEE International Conference on Image Processing 2005		In 2000, the Cohn-Kanade (CK) database was released for the purpose of promoting research into automatically detecting individual facial expressions. Since then, the CK database has become one of the most widely used test-beds for algorithm development and evaluation. However there are 3 limitations to the CK database 1) While AU codes are well validated, emotion labels are not, as they refer to what was requested rather than what was actually performed, 2) The lack of a common performance metric against which to evaluate new algorithms, and 3) Standard protocols for common databases have not emerged. This paper addresses the above mentioned imitations and proposes the Extended Cohn-Kanade (CK+) database. The number of sequences is increased by 22% and the number of subjects by 27%. The target expression for each sequence is fully FACS coded and emotion labels have been revised and validated. In addition to this, non-posed sequences for several types of smiles and their associated metadata have been added. This has resulted in a well-rounded database

				providing more accurate results.
8.	Emotional Recognition Using Facial Expression by Emoji in Real Time	International Journal of Innovative Research in Computer and Communication Vol. 5, Issue 9, September 2017	Mohammed Rajhi	This paper investigates emotional recognition using facial expression by emoji in real time. Moreover, it also develops the parameters of measuring the facial expression and understanding the facial emotion recognition in real time. The application developed includes six human expressions namely neutral, fear, anger, happy, sad, and surprise emotions. These expressions are the actual expressions which are being conveyed in human beings. The investigations of such expression are important because of their ability to better express human emotions and the way they facilitate communication.
9.	Comprehensive database for facial expression analysis	Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition	T. Kanade ; J.F. Cohn ; Yingli Tian	Within the past decade, significant effort has occurred in developing methods of facial expression analysis. Because most investigators have used

				relatively limited data sets, the generalizability of these various methods remains unknown. This paper describes the problem space for facial expression analysis, which includes level of description, transitions among expressions, eliciting conditions, reliability and validity of training and test data, individual differences in subjects, head orientation and scene complexity image characteristics, and relation to non-verbal behavior.
10.	A real-time automated system for the recognition of human facial expressions	IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)	Keith Anderson, Peter W. McOwan	This paper proposes a fully automated, multistage system for real-time recognition of facial expression. The system uses facial motion to characterize monochrome frontal views of facial expressions and is able to operate effectively in cluttered and dynamic scenes, recognizing the six emotions universally associated with unique facial expressions, namely happiness, sadness, disgust, surprise, fear, and anger. Faces are located using a spatial ratio template tracker algorithm. Optical flow of the face is subsequently determined using a real-time implementation of a robust gradient model. The expression recognition

				system then averages facial velocity information over

				identified regions of the face and cancels out rigid head motion by taking ratios of this averaged motion. The motion signatures produced are then classified using Support Vector Machines as either non-expressive or as one of the six basic emotions. The completed system is demonstrated in two simple affective computing applications that respond in real-time to the facial expressions of the user, thereby providing the potential for improvements in the interaction between a computer user and technology.

3. REQUIREMENTS ANALYSIS

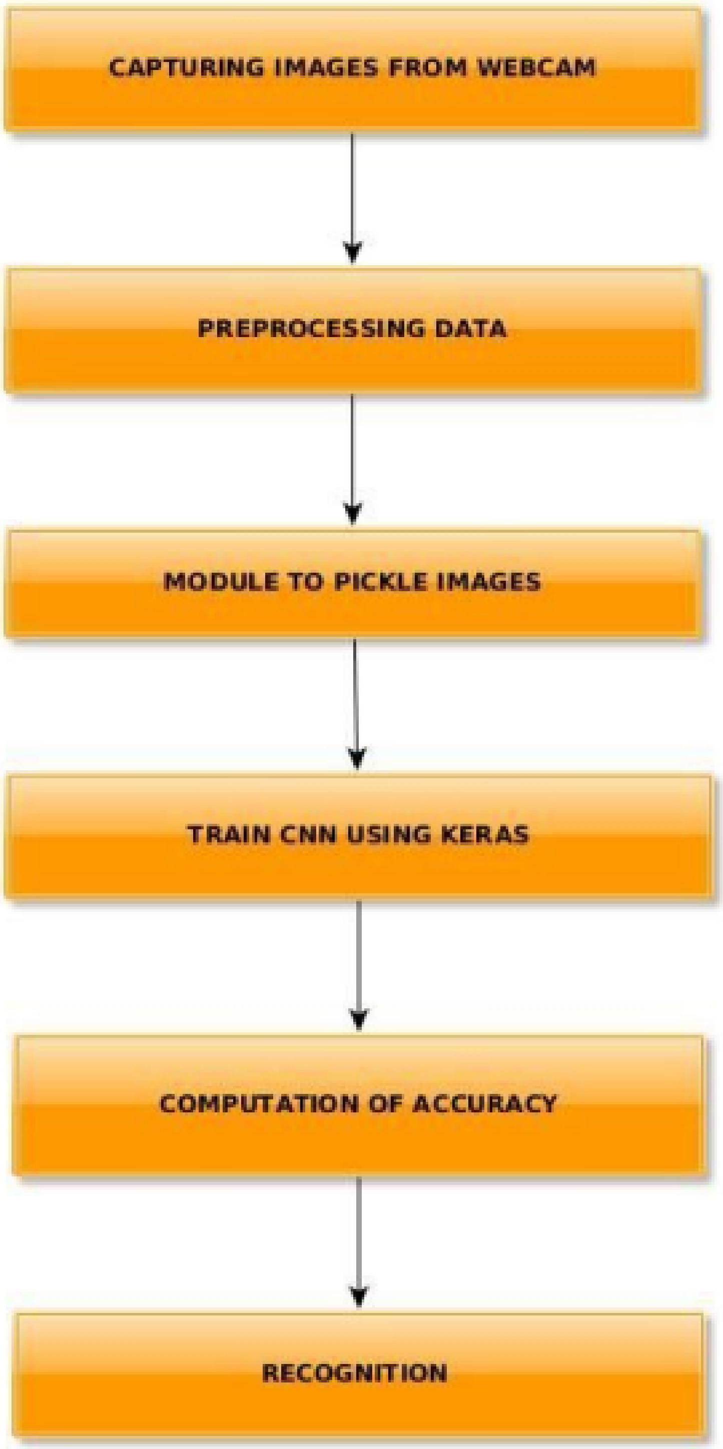
3.1 Software Requirements

1. Python 3.x
2. Tensorflow 1.5
3. Keras
4. OpenCV 3.4
5. h5py

3.2 Hardware Requirements

1. Intel i3 and higher
2. CUDA Enabled NVIDIA GPUs

4. SYSTEM DESIGN



System architecture

5. SYSTEM IMPLEMENTATION

5.1. Algorithm:

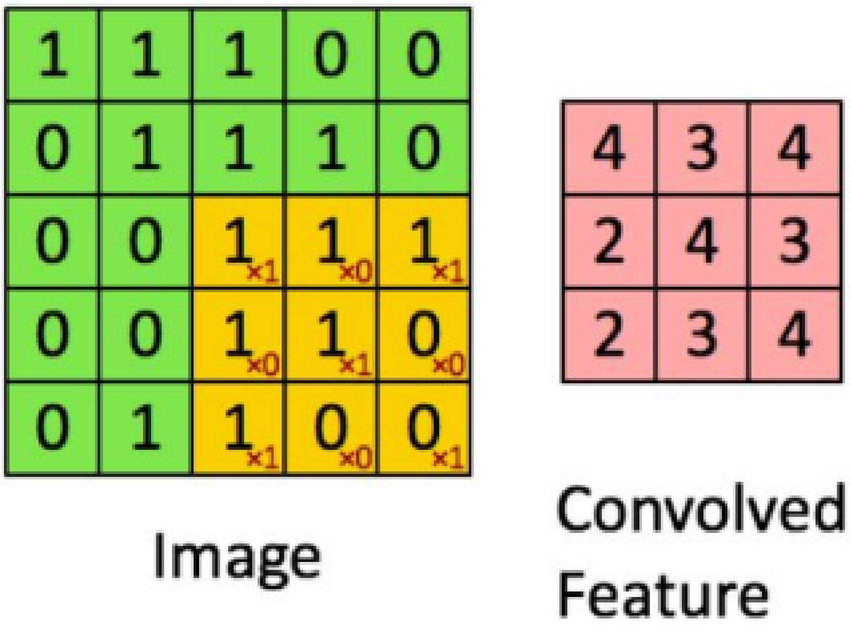
Feature Extraction using Convolution

- Natural images have the property of being ”‘stationary”’, meaning that the statistics of one part of the image are the same as any other part.
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.
- To give a concrete example, suppose you have learned features on 8x8 patches sampled from a 96x96 image. Suppose further this was done with an autoencoder that has 100 hidden units. To get the convolved features, for every 8x8 region of the 96x96 image, that is, the 8x8 regions starting at (1,1),(1,2),…(89,89), you would extract the 8x8 patch, and run it through your trained sparse autoencoder to get the feature activations. This would result in 100 sets 89x89 convolved features.

The Convolutional Neural Network

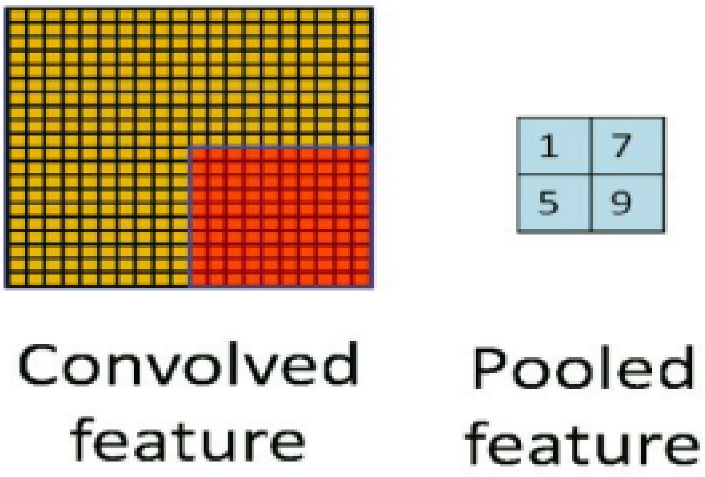
- A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3.
- The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel.
- The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1.
- Each map is then subsampled typically with mean or max pooling over p x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs.
- Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map.

Step 1:

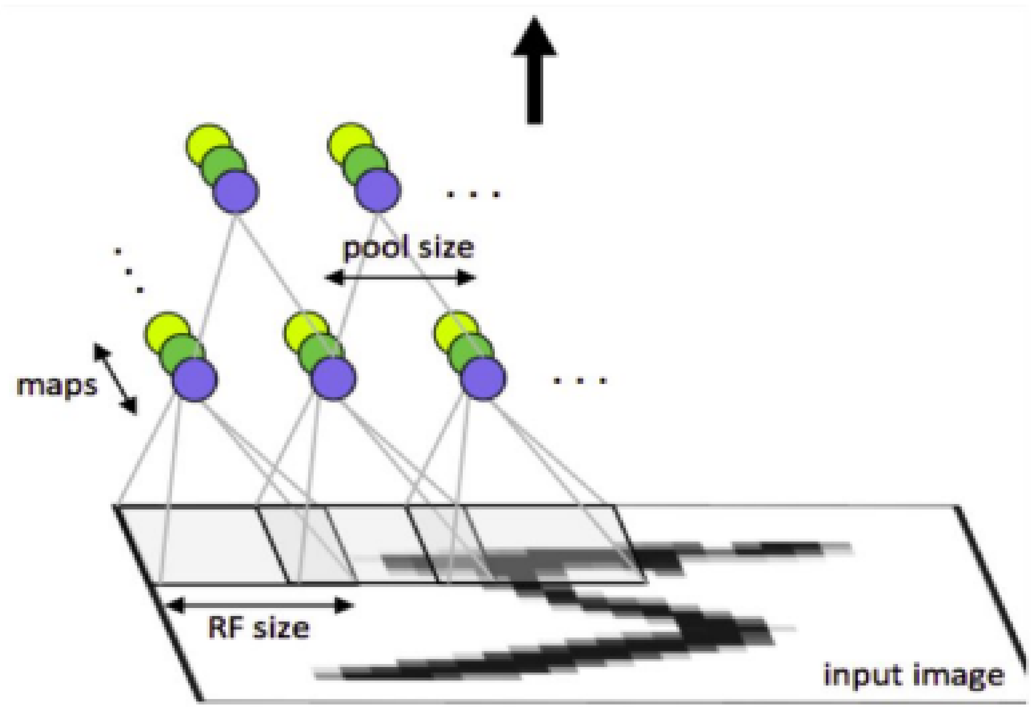


Concept of convolution

Step 2:



Concept of Convolved feature into Max Pooling



Concept of Max Pooling

5.2 Modules

Module 1- Creation of dataset
A training dataset is created with 250 images for each of the 11 expressions. The wider variety of images included will result in the creation of a better model. The images for each expression are taken with a wide variation in lighting conditions, positions etc to result in a well-rounded dataset. The capturing of images is done using a web camera.

Module 2: Modification of dataset
The captured images are processed to extract the eyebrows, eyes, nose and lips thereby ignoring all other unnecessary part of the mouth. This is done to reduce the size of the dataset, reduce the computing power and time and increase the efficiency of the model by removing the unnecessary parts of the captured images. This is followed by pickling of data where the new dataset is stored as a pickle file comprising of 6 pickles train_images, train_labels, test_images, test_labels, val_images, and val_labels.

Module 3: Training a CNN model
A new model is trained using the training data obtained from the above steps. The dataset first undergoes 2D convolution to sharpen and smoothen the images. Batch Normalization follows to improve the stability and reduce variance in dataset. Max Pooling allows us to segregate the dataset into cubes of equal dimension for better visualization of data and more accuracy. Flattening of the dataset reduces the size and allows easier processing. Another round of batch normalization allows us to

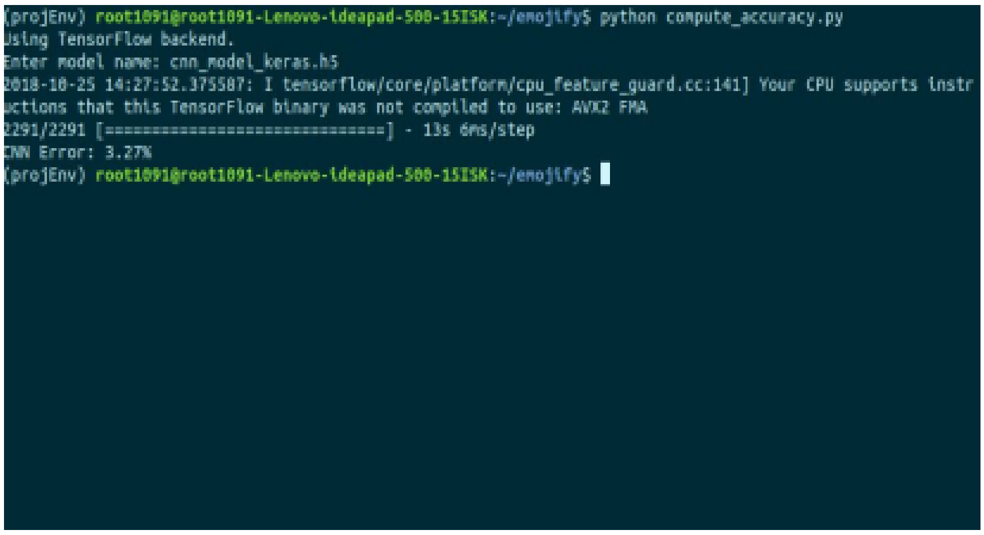
attribute the same weightage to each characteristic. This model is now optimized and ready to be tested on test data.

Module 4: Mapping to emojis

The model is now trained and ready to be tested on the created dataset. This gives the output of mapping the correct emoji on top of the human expression, thereby creating an ease of expression analysis.

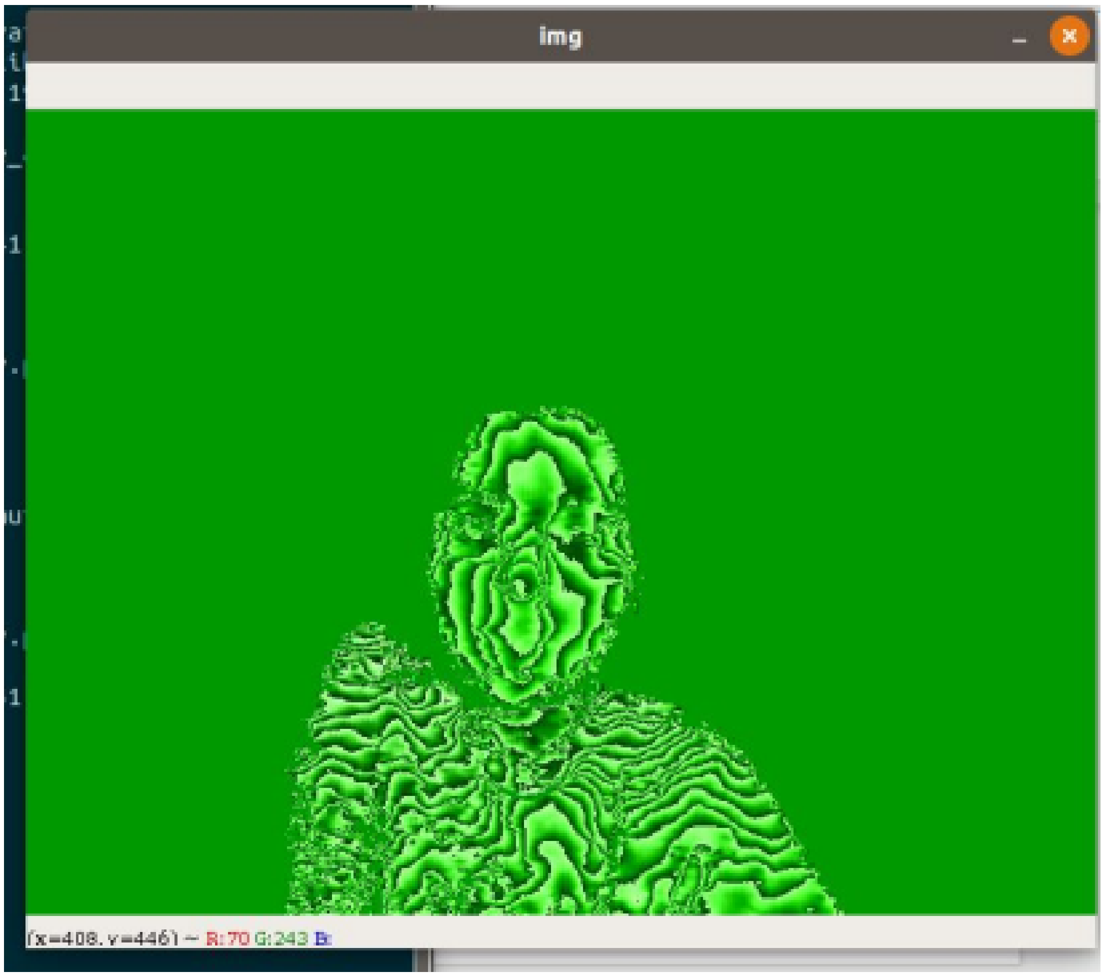
6. TESTING AND RESULT ANALYSIS

6.1 Accuracy and Benchmarking



```
(projEnv) root1091@root1091-Lenovo-ideapad-500-15ISK:~/emojify$ python compute_accuracy.py
Using TensorFlow backend.
Enter model name: cnn_model_keras.h5
2018-10-25 14:27:52.375587: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instr
uctions that this TensorFlow binary was not compiled to use: AVX2 FMA
2291/2291 [==============================] - 13s 6ms/step
CNN Error: 3.27%
(projEnv) root1091@root1091-Lenovo-ideapad-500-15ISK:~/emojify$ 
```

Screenshot 1: Computation of accuracy



Screenshot 2: GUI tool for face to emoji conversion

6.2 Speed

Without parallelization and all images dataset, it took four days to obtain the result.

With CUDA parallelization it took 2 days to train the same system.

6.3 Limitations

The limitations of the system are enumerated as follows:

1. **Acquiring dataset:** Currently the dataset is the one that has been generated by us, and in no sense is universal. Thus the results are unreliable.

2. **Time of training** the convolutional neural network, even when GPU acceleration is supported,
3. **Platform dependence** on Nvidia Graphic card for algorithm acceleration, which is a limitation of other GPUs in field of deep learning.

7. CONCLUSION AND FUTURE WORK

This study explores a possibility of human-computer interaction, based on facial expression recognition as a non-conventional interaction modality. The development of a library which is easy to use allows the adoption, by any kind of application, of the functionalities which have been exposed in this paper.

In the case of distinction between happy and sad facial expressions, a hit-rate of over 80% has been achieved. Among five different expressions (angry, happy, neutral, sad and surprised), it has been of about 55%. Despite the source code not being optimised and the library having been generated without any optimisations at the compiler level, the algorithms which have been implemented are light and allow a good temporal performance. The current implementation took slightly less than 1 minute to classify a facial image. With an optimised code and compilation process, it will be possible to achieve real-time performances.

Tests performed with the classification of multiple frames have also shown promising results. The hit-rate for “happy” vs. “sad” was of 85% and for the scenario with the five emotion classes was of almost 60%. Further studies are needed in order to better understand what the correct number of frames is to use in this case is.

There are certain aspects which may be improved in the future. It would be desirable to achieve a 90% hit-rate for distinction between two facial expressions. It will be necessary to adjust some parameters in order to achieving a system which is more robust. In the process of feature extraction, eye and mouth detection are the least reliable. In the first case, the solution will be to train a new Haar cascade classifier, which was outside the scope of this study. In the second case, it is imperative to improve the mouth-detecting algorithm. It would be interesting to use an approach which responded to variations of illumination or colour. Such an approach may bring a new robustness to the system. Also, the use of the differentials of the face associated with the capture of video was outside the scope of this study. Indeed, the whole classification process is based in the information of a single image. The use of information related to the alteration of facial features throughout a video capture may not only simplify the feature extraction process, but also provide additional information and, thus, improve its performance.

8. REFERENCES

[1] Y. Tian, T. Kanade, and J.F. Cohn, Facial Expression Analysis, Handbook of Face Recognition, Springer, October 2003.
[2] B. Fasel and J. Luettin, "Automatic facial expression analysis: a survey," Pattern Recognition, vol. 36, pp. 259–275, 2003.
[3] M. J. Lyons, J. Budynek, and S. Akamatsu, "Automatic classification of single facial images," IEEE PAMI, 1999.
[4] I. Cohen, N. Sebe, Garg A., L. Chen, and T. Huang, "Facial expression recognition from video sequences: Temporal and static modeling," CVIU, vol. 91, pp. 160–187, 2003.
[5] M.S. Bartlett, G. Littlewort, I. Fasel, and R. Movellan, "Real time face detection and facial expression recognition: Development and application to human computer interaction," in CVPR Workshop on CVPR for HCI, 2003.
[6] Y. Tian, "Evaluation of face resolution for expression analysis," in IEEE Workshop on Face Processing in Video, 2004.
[7] Z. Zhang, M. J. Lyons, M. Schuster, and S. Akamatsu, "Comparison between geometry-based and gabor-wavelets-based facial expression recognition using multi-layer perceptron," in IEEE FG, April 1998.
[8] T. Ojala, M Pietikinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distribution," Pattern Recognition, vol. 29, no. 1, 1996.
[9] T. Ojala, M. Pietikinen, and T. Menp, "Multiresolution grayscale and rotation invariant texture classification with local binary patterns," IEEE PAMI, vol. 24, no. 7, July 2002.
[10] T. Ahonen, A. Hadid, and M. Pietikinen, "Face recognition with local binary patterns," in ECCV, 2004, pp. 469–481 facial expression," in IEEE PETS, Australia, March 2003. [15] V. N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

APPENDIX I: SOURCE CODE

Python code for creating dataset from webcam

create_dataset_webcam.py

```
import cv2
import numpy as np
import dlib
import pickle
import os
from imutils import face_utils
from imutils.face_utils import FaceAligner
from random import shuffle, randint
from        preprocess_img        import        create_mask,        get_bounding_rect

SHAPE_PREDICTOR_68 = "shape_predictor_68_face_landmarks.dat"

shape_predictor_68  =  dlib.shape_predictor(SHAPE_PREDICTOR_68)  detector  =
dlib.get_frontal_face_detector()

cam = cv2.VideoCapture(1)
if cam.read()[0]==False:
        cam = cv2.VideoCapture(0)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
fa = FaceAligner(shape_predictor_68, desiredFaceWidth=250)


dataset = 'new_dataset/'
label = int(input('Enter label: '))
num_of_images = int(input('Enter number of images that you want to be taken:
'))
starting_num  =  int(input('Enter  starting  image  number:  '))  count_images  =
starting_num
is_capturing = False
if not os.path.exists(dataset+str(label)):
        os.mkdir(dataset+str(label))
while True:
        img = cam.read()[1]
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = detector(gray)
        rand = randint(0, 10)
        if len(faces) > 0:
                face = faces[0]
shape_68 = shape_predictor_68(img, face)
shape = face_utils.shape_to_np(shape_68)
                mask = create_mask(shape, img)
                masked = cv2.bitwise_and(gray, mask)
maskAligned = fa.align(mask, gray, face)


faceAligned = fa.align(masked, gray, face)
(x0, y0, x1, y1) = get_bounding_rect(maskAligned)
faceAligned = faceAligned[y0:y1, x0:x1]
faceAligned = cv2.resize(faceAligned, (100, 100))
(x, y, w, h) = face_utils.rect_to_bb(face)
cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255,
        0), 2)
if count_images-starting_num < int(num_of_images):
                                            if is_capturing:
cv2.putText(img,
        str(count_images-starting_num), (50, 50), cv2.FONT_HERSHEY_TRIPLEX, 1.5, (255, 255, 0)
        )
if rand%2 == 0:
faceAligned =
        cv2.flip(faceAligned, 1)

        cv2.imwrite(dataset+str(label)+'/'+str(count_images)+'.jpg', faceAligned)
count_images += 1
                                   else:
                                            break
cv2.imshow('faceAligned', faceAligned)
                          cv2.imshow('img', img)
                          keypress = cv2.waitKey(1)
                          if keypress == ord('q'):
                                   break
                          elif keypress == ord('c'):
                                   if is_capturing:
                                            is_capturing = False
                                   else:
                                            is_capturing = True
```

Image preprocessing

1. **To generate dataset from webcam images:** create_dataset_from_faces.py

```
import numpy as np

import pickle

import cv2, os

from keras import optimizers

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D from keras.utils

import np_utils

from keras.callbacks import ModelCheckpoint

from keras.layers.normalization import BatchNormalization from keras import

backend as K
```

```
from keras.callbacks import TensorBoard

from keras.models import load_model

from time import time

K.set_image_dim_ordering('tf')


os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'


def get_image_size():

        img = cv2.imread('dataset/0/1.jpg', 0)

        return img.shape


def get_num_of_classes():

        return len(os.listdir('dataset/'))



image_x, image_y = get_image_size()


def cnn_model():

        num_of_classes = get_num_of_classes()

        model = Sequential()

        model.add(Conv2D(32, (5,5), input_shape=(image_x, image_y, 1), activation='relu'))

        model.add(BatchNormalization())

        model.add(MaxPooling2D(pool_size=(10, 10), strides=(10, 10), padding='same'))

        model.add(Flatten())

        model.add(Dense(1024, activation='relu'))

        model.add(BatchNormalization())

        model.add(Dropout(0.6))

        model.add(Dense(num_of_classes,     activation='softmax'))     sgd     =

        optimizers.SGD(lr=1e-2)

        model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

        filepath="cnn_model_keras.h5"

        checkpoint1   =   ModelCheckpoint(filepath,   monitor='val_acc',   verbose=1,
save_best_only=True, mode='max')

        callbacks_list = [checkpoint1]

        from keras.utils import plot_model
```

```
            plot_model(model,   to_file='model.png',   show_shapes=True)   return   model,
            callbacks_list


        def train():
            with open("train_images", "rb") as f:
                train_images = np.array(pickle.load(f))
            with open("train_labels", "rb") as f:
                train_labels       =       np.array(pickle.load(f),       dtype=np.uint8)       with


            open("test_images", "rb") as f:



                test_images = np.array(pickle.load(f))
            with open("test_labels", "rb") as f:
test_labels = np.array(pickle.load(f), dtype=np.uint8)


            with open("val_images", "rb") as f:
                val_images = np.array(pickle.load(f))
            with open("val_labels", "rb") as f:
val_labels = np.array(pickle.load(f), dtype=np.uint8)


            train_images = np.reshape(train_images,
        (train_images.shape[0], image_x, image_y, 1))
            test_images = np.reshape(test_images, (test_images.shape[0], image_x, image_y, 1))
            val_images  =  np.reshape(val_images,  (val_images.shape[0],  image_x,  image_y,
        1))


            train_labels     =     np_utils.to_categorical(train_labels)     test_labels     =
            np_utils.to_categorical(test_labels)
            val_labels = np_utils.to_categorical(val_labels)


            model, callbacks_list = cnn_model()
            tensorboard                =                TensorBoard(log_dir="./logs/{}".format(time()))
            callbacks_list.append(tensorboard)
        model.fit(train_images, train_labels,
validation_data=(test_images,    test_labels),    epochs=15,    batch_size=100,
callbacks=callbacks_list)

        model = load_model('cnn_model_keras.h5')

        scores  =  model.evaluate(val_images,  val_labels,  verbose=1)  print("CNN  Error:

        %.2f%%" % (100-scores[1]*100))


train()
```

2. **To generate relevant data from images:** preprocess_image.py

```
import cv2

import numpy as np

from imutils import contours


def euclidean_distance(a, b):

          dist = 0

          if len(a) == len(b):

                    for i in range(len(a)):

                              dist += (a[i]-b[i])**2

                    dist = np.sqrt(dist)

          return int(dist)


def highest_euclidean_distance(shape, fixed_point, *other_points): num = len(other_points)

          largest_distance = 0

          for point in other_points:

                    dist    =    euclidean_distance(fixed_point,    shape[point])    if    dist    >

                    largest_distance:

                              largest_distance = dist

          return largest_distance


def centroid(shape, *points):

          num_of_points = len(points)

          x, y = 0, 0

          for point in points:
                x += shape[point][0]

                y += shape[point][1]



        centroid = (int(x/num_of_points), int(y/num_of_points)) return centroid


def get_points(face_part, shape):

        points = [shape[point] for point in face_part]

        return np.array(points)


def create_mask(shape, img):

        height, width, channels = img.shape

        mask = np.zeros((height, width), dtype=np.uint8)

        right_eye = (36, 37, 38, 39, 40, 41)

        left_eye = (42, 43, 44, 45, 46, 47)

        mouth = (48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59) nose = (27, 31, 33, 35)

        left_eyebrow = (17, 18, 19, 20, 21)

        right_eyebrow = (22, 24, 25, 26, 26)


        middle_right_eye = centroid(shape, *right_eye)

        middle_left_eye = centroid(shape, *left_eye)

        radius_left_eye  =  highest_euclidean_distance(shape,  middle_left_eye,
*left_eye)

        radius_right_eye = highest_euclidean_distance(shape, middle_right_eye,
*right_eye)

        mask = cv2.circle(mask, middle_right_eye, radius_right_eye, 255, -1)

        mask = cv2.circle(mask, middle_left_eye, radius_left_eye, 255, -1)


        middle_mouth = centroid(shape, *mouth)

        radius_mouth = highest_euclidean_distance(shape,
middle_mouth, *mouth)

mask = cv2.circle(mask, middle_mouth, radius_mouth, 255, -1)


        mask = cv2.fillPoly(mask, [get_points(left_eyebrow, shape)], 255)
        mask = cv2.fillPoly(mask, [get_points(right_eyebrow, shape)], 255)

        mask = cv2.fillPoly(mask, [get_points(nose, shape)], 255) return mask


def get_bounding_rect(img):

        cnts = cv2.findContours(img.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[1]

        boundingBoxes = contours.sort_contours(cnts,
method='left-to-right')[1]


        x0, y0 = boundingBoxes[0][0], boundingBoxes[0][1] x1, y1 = 0, 0

        for i, (x,y,w,h) in enumerate(boundingBoxes):

                if i == 0:

                        continue

                if x+w > x1:

                        x1 = x+w

                if y+h > y1:

                        y1 = y+h

        return x0,y0,x1,y1
```

3. Training CNN: train_CNN_keras.py

```
import numpy as np
import pickle
import cv2, os
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D from
keras.layers.convolutional import MaxPooling2D from keras.utils import
np_utils
from keras.callbacks import ModelCheckpoint from keras.layers.normalization
import BatchNormalization from keras import backend as K
from keras.callbacks import TensorBoard
from keras.models import load_model
from time import time
K.set_image_dim_ordering('tf')

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

def get_image_size():
        img = cv2.imread('dataset/0/1.jpg', 0)
        return img.shape

def get_num_of_classes():
        return len(os.listdir('dataset/'))

image_x, image_y = get_image_size()

def cnn_model():

        num_of_classes = get_num_of_classes()
```

```
            model = Sequential()
            model.add(Conv2D(32, (5,5), input_shape=(image_x, image_y, 1), activation='relu'))
            model.add(BatchNormalization())
            model.add(MaxPooling2D(pool_size=(10, 10), strides=(10, 10), padding='same'))
            model.add(Flatten())
            model.add(Dense(1024, activation='relu'))
            model.add(BatchNormalization())
            model.add(Dropout(0.6))
            model.add(Dense(num_of_classes,        activation='softmax'))        sgd        =
            optimizers.SGD(lr=1e-2)
            model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
            filepath="cnn_model_keras.h5"
            checkpoint1    =    ModelCheckpoint(filepath,    monitor='val_acc',    verbose=1,
save_best_only=True, mode='max')
            callbacks_list = [checkpoint1]
            from keras.utils import plot_model
            plot_model(model,   to_file='model.png',   show_shapes=True)   return   model,
            callbacks_list


def train():
            with open("train_images", "rb") as f:
                        train_images = np.array(pickle.load(f))
            with open("train_labels", "rb") as f:
                        train_labels = np.array(pickle.load(f),
dtype=np.uint8)


            with open("test_images", "rb") as f:
                        test_images = np.array(pickle.load(f))
            with open("test_labels", "rb") as f:
                        test_labels = np.array(pickle.load(f), dtype=np.uint8) 33



            with open("val_images", "rb") as f:
                        val_images = np.array(pickle.load(f))
```

```
                    with open("val_labels", "rb") as f:
val_labels = np.array(pickle.load(f), dtype=np.uint8)


                    train_images = np.reshape(train_images,
        (train_images.shape[0], image_x, image_y, 1))
                    test_images = np.reshape(test_images, (test_images.shape[0], image_x, image_y, 1))
                    val_images = np.reshape(val_images, (val_images.shape[0], image_x, image_y, 1))


                    train_labels = np_utils.to_categorical(train_labels)
                    test_labels = np_utils.to_categorical(test_labels)
                    val_labels = np_utils.to_categorical(val_labels)


                    model, callbacks_list = cnn_model()
                    tensorboard                    =                    TensorBoard(log_dir="./logs/{}".format(time()))
                    callbacks_list.append(tensorboard)
                    model.fit(train_images, train_labels,
        validation_data=(test_images, test_labels), epochs=15, batch_size=100, callbacks=callbacks_list)
                    model = load_model('cnn_model_keras.h5')
                    scores = model.evaluate(val_images, val_labels, verbose=1) print("CNN Error:
                    %.2f%%" % (100-scores[1]*100))


        train()
```

4. Computation of accuracy:

compute_accuracy.py

```
from keras.models import load_model
import numpy as np
import pickle
from keras.utils import np_utils


with open("train_images", "rb") as f:
            train_images = np.array(pickle.load(f))
with open("train_labels", "rb") as f:
        train_labels = np.array(pickle.load(f), dtype=np.uint8)

train_images = np.reshape(train_images, (train_images.shape[0], 100, 100, 1))
train_labels = np_utils.to_categorical(train_labels)

model_name = input('Enter model name: ')
model = load_model(model_name)
scores = model.evaluate(train_images, train_labels, verbose=1) print("CNN Error:
%.2f%%" % (100-scores[1]*100))
```

5. GUI based tool: emojify.py

```
import cv2
import numpy as np
import dlib, os
from imutils import face_utils
from imutils.face_utils import FaceAligner
from keras.models import load_model
from preprocess_img import create_mask, get_bounding_rect from blend
import blend

CNN_MODEL = 'cnn_model_keras.h5'
SHAPE_PREDICTOR_68 = "shape_predictor_68_face_landmarks.dat"
```

```
cnn_model = load_model(CNN_MODEL)
shape_predictor_68 = dlib.shape_predictor(SHAPE_PREDICTOR_68) detector =
dlib.get_frontal_face_detector()

cam = cv2.VideoCapture(1)
if cam.read()[0]==False:
        cam = cv2.VideoCapture(0)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
fa = FaceAligner(shape_predictor_68, desiredFaceWidth=250)

def get_emojis():
        emojis_folder = 'emojis/'
        emojis = []
        for emoji in range(len(os.listdir(emojis_folder))): print(emoji)

emojis.append(cv2.imread(emojis_folder+str(emoji)+'.png', -1)) return emojis



def get_image_size():
        img = cv2.imread('dataset/0/100.jpg', 0)
        return img.shape

image_x, image_y = get_image_size()

def keras_process_image(img):
        img = cv2.resize(img, (image_x, image_y))
        img = np.array(img, dtype=np.float32)
        img = np.reshape(img, (1, image_x, image_y, 1))
        return img

def keras_predict(model, image):
```

```
                processed = keras_process_image(image)

                pred = model.predict(processed)

                pred_probab = pred[0]

                pred_class      =      list(pred_probab).index(max(pred_probab))      return

                max(pred_probab), pred_class


        def fun_util():

                emojis = get_emojis()

                disp_probab, disp_class = 0, 0

                while True:

                        img = cam.read()[1]

                        img = cv2.flip(img, 1)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

                        faces = detector(gray)

                        if len(faces) > 0:

for i, face in enumerate(faces):

shape_68 = shape_predictor_68(img,
        face)

                                        shape =
        face_utils.shape_to_np(shape_68)

mask = create_mask(shape, img)



masked = cv2.bitwise_and(gray, mask)

maskAligned = fa.align(mask, gray,
        face)

faceAligned = fa.align(masked, gray,
        face)

(x0, y0, x1, y1) =
        get_bounding_rect(maskAligned)

faceAligned = faceAligned[y0:y1,
        x0:x1]

faceAligned = cv2.resize(faceAligned,
         (100, 100))

(x, y, w, h) =
        face_utils.rect_to_bb(face)

#cv2.rectangle(img, (x, y), (x+w,
        y+h), (255, 255, 0), 2)
```

```
cv2.imshow('faceAligned', faceAligned)

cv2.imshow('face #{}'.format(i),
        img[y:y+h, x:x+w])

pred_probab, pred_class =
        keras_predict(cnn_model, faceAligned)

img = blend(img, emojis[pred_class],
         (x, y, w, h))

                              cv2.imshow('img', img)

                              if cv2.waitKey(1) == ord('q'):

                                        break


        keras_predict(cnn_model, np.zeros((100, 100, 1), dtype=np.uint8)) fun_util()
```