

HUMAN EMOTION RECOGNITION SYSTEM USING MULTI BRANCH CROSS CONNECTED CNN

Submitted in partial fulfilment of the requirements for
the award of

Bachelor of Engineering degree in Computer Science and Engineering

By

SANGADALA VISHNU

(Reg.No.38110651)

&

YARRAMREDDY SIVA

KRISHNA

(Reg.No.38110665)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI - 600 119

MARCH - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of "SANGADALA VISHNU(38110651), YARRAMREDDY SIVA KRISHNA (38110665) " under my supervision from December 2021 to March 2022.

DR.L.K.JOSHILA GRACE., M.E., Ph.D.
Internal Guide

DR. L. LAKSHMANAN M.E., PH.D.
Head of the Department

Submitted for Viva-voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We **SANGADALA VISHNU** (38110651) and **YARRAMREDDY SIVA KRISHNA** (38110665) hereby declare that the Project Report entitled “**HUMAN EMOTION RECOGNITION SYSTEM USING MULTI BRANCH CROSS CONNECTED CNN**” done by us under the guidance of **DR.L.K.JOSHILA GRACE** (internal) is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering / Technology degree in **Computer Science and Engineering**.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing **Dr. L. Lakshmanan M.E., Ph.D. ,** and **Dr.S.Vigneshwari M.E., Ph.D. Heads** of the Department of Computer Science and Engineering for providing me with the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.L.K.JOSHILA GRACE, M.E., Ph.D..** for her valuable guidance, suggestions and constant encouragement that paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The goal is to use a Multi-Branch Cross-Connected Convolutional neural network to construct a new model for expression detection based on a novel face decomposition. CNN, Residual Networks, and Network-in Networks are all included in this strategy. Other approaches may not be able to extract as many features as this method. This method adds a shortcut link, resulting in a highly smooth data or feature flow, as well as good feature extraction. The problem of insufficient feature extraction is addressed by the proposed strategy. It can recognise the following emotions: Angry, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral. The data flow in Single-structure CNN is not smooth, and feature extraction is insufficient. The proposed method can perform well in terms of emotion recognition and has a high level of accuracy. More fluid data transfer between networks improves feature extraction and increases emotion recognition performance.

Contents

CHAPTER 1 INTRODUCTION	1
CHAPTER 2 LITERATURE REVIEW	2
CHAPTER 3 METHODOLOGY	4 - 33
3.1 PROPOSED SYSTEM	4
3.2 EXPRESSION RECOGNITION	13
3.3 NEURAL NETWORKS	17
3.4 PROGRAMMING LANGUAGE	18
3.5 IMAGE PRE-PROCESSING	24
3.6 LIBRARIES USED	27
3.7 ARCHITECTURE AND ALGORITHM	31
CHAPTER 4 RESULTS AND DISCUSSIONS	34
CHAPTER 5 CONCLUSION AND FUTURE WORK	38
5.1 REFERENCES	39
5.2 APPENDICES	41
5.2.1 Code implementation	
41-50	

CHAPTER 1 - INTRODUCTION

Human machine interaction is very important in our near future. Now everybody has mobile phones , in the future definitely all of us will have to use robots for our household work and office too. So it's important for robots to identify our emotions and respond accordingly. Face is also an important thing in Human communication also. Emotions play a central role in social interaction . By seeing a face of a person when they can find their mood, there will be some patterns in facial muscles so that we can extract features and train the system to learn those features. So the system learns the patterns and checks for patterns in the human face , so if a pattern matches the trained or already known expression, it will give output .

Some emotions (Sadness, Anger, Fear, Joy, Disgust and Surprise). Only 7% of the communication depends on verbal interactions, 38% represent voice, 55% are facial expressions of the face. Automatic emotion recognition is helpful in fields like eLearning and affective computing. There are actually three constraints in building an automatic system : they are face detection, feature extraction, expression classification. First, face processing stage to automatically locate the face region in the input images.

The next stage is to extract and portray the facial alterations that occur as a result of different facial expressions. Finally, the facial expressions can be inferred using the categorization task. The facial expression identification technique can be separated into appearance-based and geometric-based methods depending on the existing forms of feature extraction.

CHAPTER 2 - LITERATURE REVIEW

A facial expression is a combination of an expressive component and neutral component of a person. They proposed removing valuable patterns/information from the expressive component (de-expression learning process) to recognise facial expressions (DeRL). First, a generative model is trained by cGAN. This model generates corresponding neutral face image for any input face image provided to it .[1]

Facial expression recognition system using Convolution Neural Network (CNN).It is based on LeNet Architecture. They have chosen a Kaggle facial expression dataset with seven facial expression labels as happy, sad, surprise, fear, anger, disgust, and neutral . The system achieved very less accuracy i.e., 56.77 % accuracy and 0.57 precision on testing dataset.[2]

Facial expressions Recognition model which uses FER2013 dataset. Models that were experimented with include decision trees, feed forward neural networks and the smaller convolutional networks before arriving at the proposed model. The effects of different hyper parameters on the final model was then investigated. The final accuracy of 0.60 was achieved using the Adam optimiser with modified hyper parameters. [3]

Convolution Neural Network is used to construct a facial expression recognition system (CNN). The project's CNN model is based on the LeNet Architecture. This project uses a Kaggle face expression dataset with seven facial expression labels: happy, sad, surprise, fear, rage, disgust, and neutral. The system achieved 56.77 % accuracy and 0.57 precision on testing dataset[4]

Facial recognition system using RNN, i.e. recurrent neural network . This method has very less accuracy among all others .[5]

Facial recognition using Opencv and Landmark detection, this method finds the distances between certain points on face and classifies accordingly. This method fails because of less feature extraction , it seems to be less accurate because it calculates distances and the distances are not appropriate in classifying emotion. [6]

Emotion recognition using Light -weighted Convolutional neural network, this method doesn't consider highly weighted inputs into its hidden layers, so this method also fails because it misses some patterns in feature extraction. [7]

Facial emotion recognition system using Crowd-Sourcing label distribution, this method uses deep neural networks to extract patterns from the image. This method is complex and accurate but not efficient. Its accuracy is very good but it's complex structure is not that easy to implement and understand.[8]

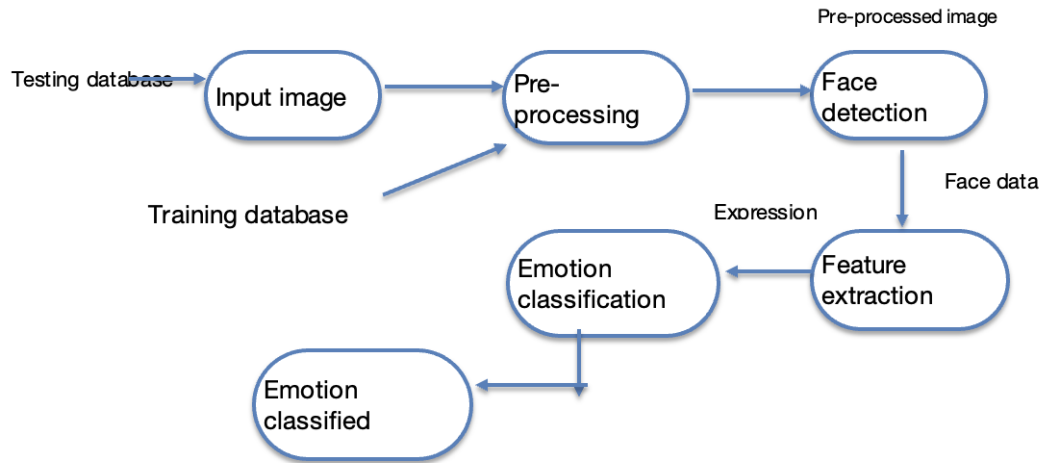
The local features are extracted in each frame using Gabor wavelets with selected scales and orientations were proposed by the author Ithaya et al . These features are passed to an ensemble classifier for detecting the location of face region. The developed method is achieved 98% accuracy using CK dataset.[9]

Ali Mollahosseini et al , an architecture called deep neural network to solve the facial expression problem over many face datasets. The proposed network has two layers called four Inception layer and max pooling. The author used variety of face datasets such as MMI, CK+,FERA, DISFA, SFEW, MultiPIE. [10]

CHAPTER 3 – METHODOLOGY

3.1 PROPOSED SYSTEM

The proposed system helps in extracting more features from the images because more networks are used , so helps us to classify the emotion accurately. We can also provide video files also so that we can get emotion classification for every frame of that video. This works main principle is that more features can lead to accurate results. Using multiple branches can take more insights or features from the audio or video file. The system takes training which consists of some input images and the model is constructed with more than one branch, then all the features from the respective branches are fused together and checks for the features that are already learnt and if fused features are mapped with that particular emotion then it outputs a particular emotion. It also shows all the emotion values in the output. If the value for a particular emotion is high than otherthen we should consider that highest valued emotion.



(fig-1 BLOCK DIAGRAM)

Testing and training database :-

We are considering some datasets like FER+, CK+, FER2013, RAF for training and testing. The datasets Ck+ and RAF are used for testing and FER+ and FER2013 for training. These datasets have many images and their classification according to expression. So we train the model using training datasets and test the model after training using testing datasets.

Data Pre-processing :-

The first step before training the model is to clean or pre-process the images in the database. Pre-processing in the sense of cropping the images, removing some images which are not useful etc..

Because of data preprocessing we can be able to get accurate and efficient results. If images are not cropped then the model has to cover more regions which takes some extra time. So pre-processing helps to make pictures more ready for training and testing.

Model :-

After the pre-processing step we provide the training images to the model so that it can extract features from the images and learns which pattern or combination of features gives what expression. By giving more and more pictures the model improves its learning and it will be good at predicting and classifying the image. We will check the model with testing data to ensure whether the model has learnt things in a proper manner and giving desired outputs.

Feature Extraction :-

Convolutional neural networks map features to features whereas ANN maps pixel to pixel. So the model detects patterns in an input image and tries to map to the trained images and classifies the image and returns the respective emotion or expression.

Classification :-

First we need to train the model using the training data, before this the training data has to be pre-processed. The model extracts the patterns or features from the images and classifies accordingly i.e. emotion classification. Whenever we provide a new image to the model, then the model finds the features from that image and classifies the input image depending upon the already classified features.

Face detection involves separating image windows into two classes; one containing faces (turning the background (clutter)). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of

any face under any set of lighting conditions, upon any background. The face detection task can be broken down into two steps.

The first step is a classification task that takes some arbitrary image as input and outputs a binary value of yes or no, indicating whether there are any faces present in the image. The second step is the face localization task that aims to take an image as input and output the location of any face or faces within that image as some bounding box with (x, y, width, height).

THE FACE DETECTION SYSTEM CAN BE DIVIDED INTO THE FOLLOWING STEPS:-

1. Pre-Processing:

To reduce the variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping images with frontal faces to include only the front view. All the cropped images are then corrected for lighting through standard algorithms.

2. Classification:

Neural networks are implemented to classify the images as faces or non faces by training on these examples. We use both our implementation of the neural network and the neural network toolbox for this task. Different network configurations are experimented with to optimize the results.

3. Localization:

The trained neural network is then used to search for faces in an image and if present localize them in a bounding box. Various Feature of Face on which the work has done on:- Position Scale Orientation Illumination

Face detection is a computer technology that determines the location and size of human face in arbitrary (digital) image. The facial features are detected and any other objects like trees, buildings and bodies etc are ignored from the digital image. It can be regarded as a 'specific' case of object-class detection, where the task is finding the location and sizes of all objects in an image that belong to a given class. Face detection, can be regarded as a more 'general' case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). Basically there are two types of approaches to detect facial part in the given image i.e. feature base and image base approach. Feature base approach tries to extract features of the image and match it against the knowledge of the face features. While image base approach tries to get best match between training and testing images.

FEATURE BASE APPROCH:

Active Shape Model Active shape models focus on complex non-rigid features like actual physical and higher level appearance of features Means that Active Shape Models (ASMs) are aimed at automatically locating landmark points that define the shape of any statistically modelled object in an image. When of facial features such as the eyes, lips, nose, mouth and eyebrows. The training stage of an ASM involves the building of a statistical

a) facial model from a training set containing images with manually annotated landmarks. ASMs is classified into three groups i.e. snakes, PDM, Deformable templates

b) 1.1)Snakes:The first type uses a generic active contour called snakes, first introduced by Kass et al. in 1987 Snakes are used to identify head boundaries [8,9,10,11,12]. In order to achieve the task, a snake is first initialized at the proximity

around a head boundary. It then locks onto nearby edges and subsequently assume the shape of the head. The evolution of a snake is achieved by minimizing an energy function, E_{snake} (analogy with physical systems), denoted as $E_{snake} = E_{internal} + E_{external}$. Where $E_{internal}$ and $E_{external}$ are internal and external energy functions. Internal energy is the part that depends on the intrinsic properties of the snake and defines its natural evolution

The typical natural evolution in snakes is shrinking or expanding. The external energy counteracts the internal energy and enables the contours to deviate from the natural evolution and eventually assume the shape of nearby features—the head boundary at a state of equilibria. Two main consideration for forming snakes i.e. selection of energy terms and energy minimization. Elastic energy is used commonly as internal energy. Internal energy is vary with the distance between control points on the snake, through which we get contour an elastic-band characteristic that causes it to shrink or expand. On other side external energy relay on image features. Energy minimization process is done by optimization techniques such as the steepest gradient descent. Which needs highest computations. Huang and Chen and Lam and Yan both employ fast iteration methods by greedy algorithms. Snakes have some demerits like contour often becomes trapped onto false image features and another one is that snakes are not suitable in extracting non convex features.

DEFORMABLE TEMPLATES:

Deformable templates were then introduced by Yuille et al. to take into account the a priori of facial features and to better the performance of snakes. Locating a facial feature boundary is not an easy task because the local evidence of facial edges is difficult to organize into a sensible global entity using generic contours. The low brightness contrast around some of these features also makes the edge detection process. Yuille et al. took the concept of snakes a step further by incorporating global information of the eye to improve the reliability of the extraction process Deformable templates approaches are developed to solve this problem. Deformation is based on local valley, edge, peak, and brightness .Other than face boundary, salient feature (eyes, nose, mouth and eyebrows) extraction is a great challenge of face recognition. $E = E_v + E_e + E_p + E_i + E_{internal}$; where E_v , E_e , E_p , E_i , $E_{internal}$ are external energy due to valley, edges, peak and image brightness and internal energy

PDM (Point Distribution Model):

Independently of computerized image analysis, and before ASMs were developed, researchers developed statistical models of shape . The idea is that once you represent shapes as vectors, you can apply standard statistical methods to them just like any other multivariate object. These models learn allowable constellations of shape points from training example and use principal components to build what is called a Point Distribution Model. These have been used in diverse ways, for example for categorizing Iron Age broaches. Ideal Point Distribution Models can only deform in ways that are characteristic of the object. Cootes and his colleagues were seeking models which do exactly that so if a beard, say, covers the chin, the shape model can "override the image" to approximate the position of the chin under the beard. It was therefore natural (but perhaps only in retrospect) to adopt Point Distribution Models. This synthesis of ideas from image processing and statistical shape modelling led to the Active Shape Model. The first parametric statistical shape model for image analysis based on principal components of inter-landmark distances was presented by Cootes and Taylor in. On this approach, Cootes, Taylor, and their colleagues, then released a series of papers that cumulated in what we call the classical Active Shape Model.

LOW LEVEL ANALYSIS:

Based on low level visual features like colour, intensity, edges, motion etc. Skin Colour BaseColor is vital feature of human faces. Using skin-color as a feature for tracking a face has several advantages. Colour processing is much faster than processing other facial features. Under certain lighting conditions, colour is orientation invariant. This property makes motion estimation much easier because only a translation model is needed for motion estimation. Tracking human faces using colour as a feature has several problems like the colour representation of a face obtained by a camera is influenced by many factors (ambient light, object movement, etc Majorly three different face detection algorithms are available based on RGB, YCbCr, and HIS colour space models. In the implementation of the algorithms there are three main steps viz.

- (1) Classify the skin region in the colour space,**
- (2) Apply threshold to mask the skin region and**
- (3) Draw bounding box to extract the face image.**

Crowley and Coutaz suggested simplest skin colour algorithms for detecting skin pixels. The perceived human colour varies as a function of the relative direction to the illumination The pixels for skin region can be detected using a normalized colour

histogram, and can be normalized for changes in intensity on dividing by luminance. Converted an $[R, G, B]$ vector is converted into an $[r, g]$ vector of normalized colour which provides a fast means of skin detection. This algorithm fails when there are some more skin region like legs, arms, etc. Cah and Ngan suggested skin colour classification algorithm with YCbCr colour space. Research found that pixels belonging to skin region having similar Cb and Cr values. So that the thresholds be chosen as $[Cr1, Cr2]$ and $[Cb1, Cb2]$, a pixel is classified to have skin tone if the values $[Cr, Cb]$ fall within the thresholds. The skin colour distribution gives the face portion in the color image algorithm is also having the constraint that the image should be having only face as the skin region. Kjeldson and Kinder defined a color predicate in HSV color space to separate skin regions from background. Skin color classification in HSI colour space is the same as YCbCr colour space but here the responsible values are hue (H) and saturation (S). Similar to above the threshold be chosen as $[H1, S1]$ and $[H2, S2]$, and a pixel is classified to have skin tone if the values $[H, S]$ fall within the threshold and this distribution gives the localized face image. Similar to above two algorithm this algorithm is also having the same constraint

MOTION BASE:

When use of video sequence is available, motion information can be used to locate moving objects. Moving silhouettes like face and body parts can be extracted by simply thresholding accumulated frame differences. Besides face regions, facial features can be located by frame differences.

GRAY SCALE BASE:

Gray information within a face can also be treat as important features. Facial features such as eyebrows, pupils, and lips appear generally darker than their surrounding facial regions. Various recent feature extraction algorithms search for local gray minima within segmented facial regions. In these algorithms, the input images are first enhanced by contrast-stretching and gray-scale morphological routines to improve the quality of local dark patches and thereby make detection easier. The extraction of dark patches is achieved by low-level gray-scale thresholding. Based method and consist three levels. Yang and huang presented new approach i.e. faces gray scale behaviour in pyramid (mosaic) images. This system utilizes hierarchical Face location consist three levels. Higher two level based on mosaic images at different resolution.

In the lower level, edge detection method is proposed. Moreover this algorithms gives fine response in complex background where size of the face is unknown

EDGE BASE:

Face detection based on edges was introduced by Sakai et al. This work was based on analysing line drawings of the faces from photographs, aiming to locate facial features. Than later Craw et al. proposed a hierarchical framework based on Sakai et al.'s work to trace a human head outline. Then after remarkable works were carried out by many researchers in this specific area. Method was very simple and fast. They proposed frame work which consist three stepwise. Initially the images are enhanced by applying median filter for noise removal and histogram equalization for contrast adjustment. In the second step the edge images constructed from the enhanced image by applying sober operator. Then a novel edge tracking algorithm is applied to extract the sub windows from the enhanced image based on edges. Further they used Back propagation Neural Network (BPN) algorithm to classify the sub-window as either face or non-face.

FEATURE ANALYSIS

These algorithms aim to find structural features that exist even when the pose, viewpoint, or lighting conditions vary, and then use these to locate faces. These methods are designed mainly for face localization

FEATURE SEARCHING

VIOLA JONES METHOD:

Paul Viola and Michael Jones presented an approach for object detection which minimizes computation time while achieving high detection accuracy. Paul Viola and Michael Jones proposed a fast and robust method for face detection which is 15 times quicker than any technique at the time of release with 95% accuracy at around 17 fps. The technique relies on the use of simple Haar-like features that are evaluated quickly through the use of a new image representation. Based on the

concept of an —Integral Image it generates a large set of features and uses the boosting algorithm AdaBoost to reduce the overcomplete set and the introduction of a degenerative tree of the boosted classifiers provides for robust and fast interferences. The detector is applied in a scanning fashion and used on gray-scale images, the scanned window that is applied can also be scaled, as well as the features evaluated.

GABOR FEATURE METHOD:

Sharif et al proposed an Elastic Bunch Graph Map (EBGM) algorithm that successfully implements face detection using Gabor filters. The proposed system applies 40 different Gabor filters on an image. As a result of which 40 images with different angles and orientation are received. Next, maximum intensity points in each filtered image are calculated and mark them as fiducially points. The system reduces these points in accordance to distance between them. The next step is calculating the distances between the reduced points

Using distance formula. At last, the distances are compared with database. If match occurs, it means that the faces in the image are detected

CONSTELLATION METHOD

All methods discussed so far are able to track faces but still some issue like locating faces of various poses in complex background is truly difficult. To reduce this difficulty investigator form a group of facial features in face-like constellations using more robust modelling approaches such as statistical analysis. Various types of face constellations have been proposed by Burl et al. . They establish use of statistical shape theory on the features detected from a multiscale Gaussian derivative filter. Huang et al. also apply a Gaussian filter for pre-processing in a framework based on image feature analysis. Image Base Approach

3.2 EXPRESSION RECOGNITION

Over the last few decades many techniques have been proposed for face recognition. Many of the techniques proposed during the early stages of computer vision cannot be considered successful, but almost all of the recent approaches to the face recognition problem have been creditable. According to the research by Brunelli and Poggio (1993) all approaches to human face recognition can be divided into two strategies:

- (1) Geometrical features and
- (2) Template matching.

FACE RECOGNITION USING GEOMETRICAL FEATURES

This technique involves computation of a set of geometrical features such as nose width and length, mouth position and chin shape, etc. from the picture of the face we want to recognize. This set of features is then matched with the features of known individuals. A suitable metric such as Euclidean distance (finding the closest vector) can be used to find the closest match. Most pioneering work in face recognition was done using geometric features (Kanade, 1973), although Craw et al. (1987) did relatively recent work in this area.

must be recognized. The possible results when projecting a face into face space

THERE ARE FOUR POSSIBILITIES:

- 1. Projected image is a face and is transformed near a face in the face database
- 2. Projected image is a face and is not transformed near a face in the face database
- 3. Projected image is not a face and is transformed near a face in the face database
- 4. Projected image is not a face and is not transformed near a face in the face database

While it is possible to find the closest known face to the transformed image face by calculating the Euclidean distance to the other vectors, how does one know whether the image that is being transformed actually contains a face? Since PCA is a many-to-one transform, several vectors in the image space (images) will map to a point in face space (the problem is that even non-face images may transform near a known face image's faces space vector). Turk and Pentland (1991a), described a simple way of checking whether an image is actually of a face. This is by transforming an image into face space and then transforming it back (reconstructing) into image space. Using the previous notation

With these calculations it is possible to verify that an image is of a face and recognise that face. O'Toole et al. (1993) did some interesting work on the importance of eigen faces with large and small eigenvalues. They showed that the eigen vectors with larger eigenvalues convey information relative to the basic shape and structure of the faces. This kind of information is most useful in categorising faces according to sex, race etc. Eigen vectors with smaller eigenvalues tend to capture information that is specific to single or small subsets of learned faces and are useful for distinguishing a particular face from any other face. Turk and Pentland (1991a) showed that about 40 eigen faces were sufficient for a very good description of human faces since the reconstructed image have only about 2% RMS. pixel-by-pixel errors.

IMPROVING FACE DETECTION USING RECONSTRUCTING

Reconstruction cannot be used as a means of face detection in images in near real-time since it would involve resizing the face detection *window* area and large matrix multiplication, both of which are computationally expensive. However, reconstruction can be used to verify whether potential face locations identified by the deformable template algorithm actually contain a face. If the reconstructed image differs greatly from the face detection *window* then the *window* probably does not contain a face. Instead of just identifying a single potential face location, the face detection algorithm can be modified to output many high 'faceness' locations which can be verified using reconstruction. This is especially useful because occasionally

the best 'faceness' location found by the deformable template algorithm may not contain the ideal frontal view face pixel area. potential face locations that have been identified by the face detection system (the best face locations it found on its search) are checked whether they contain a face. If the threshold level (maximum difference between reconstruction and original for the original to be a face) is set correctly this will be an efficient way to detect a face. The deformable template algorithm is fast and can reduce the search space of potential face locations to a handful of positions. These are then checked using reconstruction. The number of locations found by the face detection system can be changed by getting it to output, not just the best face locations it has found so far but any location, which has a 'faceness' value, which for example is, at least 0.9 times the best heuristic value that has been found so far. Then there will be many more potential face locations to be checked using reconstruction. This and similar speed versus accuracy trade-off decisions have to be made keeping in mind the platform on which the system is implemented.

Similarly, instead of using reconstruction to check the face detection system's output, the output's correlation with the average face can be checked. The segmented areas with a high correlation probably contains a face. Once again a threshold value will have to be established to classify faces from non-faces. Similar to reconstruction, resizing the segmented area and calculating its correlation with the average face is far too expensive to be used alone for face detection but is suitable for verifying the output of the face detection system.

POSE INVARIANT FACE RECOGNITION

Extending the frontal view face recognition system to a pose-invariant recognition system is quite simple if one of the proposed specifications of the face recognition system is relaxed. Successful pose-invariant recognition will be possible if many images of a known individual are in the face database. Nine images from each known individual can be taken as shown below. Then if an image of the same individual is submitted within a 30o angle from the frontal view he or she can be identified.

Nine images in face database from a single known individual Pose invariant face recognition highlights the generalisation ability of PCA. For example, when an

individual's frontal view and 30° left view known, even the individual's 15° left view can be recognised.

It is process of identifying different parts of human faces like eyes, nose, mouth, etc... this process can be achieved by using MATLAB code. In this project the author will attempt to detect faces in still images by using image invariants. To do this it would be useful to study the grey-scale intensity distribution of an average human face. The following 'average human face' was constructed from a sample of 30 frontal view human faces, of which 12 were from females and 18 from males. A suitably scaled colormap has been used to highlight grey-scale intensity differences.

The grey-scale differences, which are invariant across all the sample faces are strikingly apparent. The eye-eyebrow area seem to always contain dark intensity (low) gray-levels while nose forehead and cheeks contain bright intensity (high) grey-levels. After a great deal of experimentation, the researcher found that the following areas of the human face were suitable for a face detection system based on image invariants and a deformable template.

The above facial area performs well as a basis for a face template, probably because of the clear divisions of the bright intensity invariant area by the dark intensity invariant regions. Once this pixel area is located by the face detection system, any particular area required can be segmented based on the proportions of the average human face. After studying the above images it was subjectively decided by the author to use the following as a basis for dark intensity sensitive and bright intensity sensitive templates. Once these are located in a subject's face, a pixel area 33.3% (of the width of the square window) below this

Note the slight differences which were made to the bright intensity invariant sensitive template which were needed because of the pre-processing done by the system to overcome irregular lighting (chapter six). Now that a suitable dark and bright intensity invariant templates have been decided on, it is necessary to find a way of using these to make 2 A-units for a perceptron, i.e. a computational model is needed to assign neurons to the distributions displayed

3.3 NEURAL NETWORKS

Neural networks gaining much more attention in many pattern recognition problems, such as OCR, object recognition, and autonomous robot driving. Since face detection can be treated as a two class pattern recognition problem, various neural network algorithms have been proposed. The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one demerit is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance. In early days most hierarchical neural network was proposed by Agui et al. [43]. The first stage having two parallel sub networks in which the inputs are filtered intensity values from an original image. The inputs to the second stage network consist of the outputs from the sub networks and extracted feature values. An output at the second stage shows the presence of a face in the input region. Propp and Samal developed one of the earliest neural networks for face detection. Their network consists of four layers with 1,024 input units, 256 units in the first hidden layer, eight units in the second hidden layer, and two output units. Feraud and Bernier presented a detection method using auto associative neural networks. The idea is based on which shows an auto associative network with five layers is able to perform a nonlinear principal component analysis. One auto associative network is used to detect frontal-view faces and another one is used to detect faces turned up to 60 degrees to the left and right of the frontal view. After that Lin et al. presented a face detection system using probabilistic decision-based neural network (PDBNN). The architecture of PDBNN is similar to a radial basis function (RBF) network with modified learning rules and probabilistic interpretation.

3.4 PROGRAMMING LANGUAGE

PYTHON (PROGRAMMING LANGUAGE)

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation

HISTORY

Python was conceived in the late 1980s, and its implementation began in December 1989 [28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL). About the origin of Python, Van Rossum wrote in 1996: Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much

else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x and 2.7.x version series. The End Of Life date (EOL, sunset date) for Python 2.7 was initially set at 2015, then postponed to 2020 out of concern that a large body of existing code cannot easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life but Google cited performance under concurrent workloads as their only motivation. Features and philosophy Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other 6/13/2017 Python (programming language) paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers some support for functional programming in the Lisp tradition. The language has `map()`, `reduce()` and `filter()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools

borrowed from Haskell and Standard ML. The core philosophy of the language is summarized by the document The Zen of Python (PEP 20), which includes aphorisms such as: Beautiful is better than ugly Explicit is better than implicit Simple is better than complex Complex is better than complicated Readability counts Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it". Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar. A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming

language is called unpythonic. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example: List comprehensions vs. for-loops Conditional expressions vs. if blocks The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements. 6/13/2017 Python (programming language) - Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python. Methods Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Typing Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them. Python allows programmers to define their own types using classes, which are most often used for objectoriented programming. New instances of classes are constructed by calling

the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta programming and reflection. Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0. The long term plan is to support gradual typing and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named `mypy` supports compile-time type checking.

Mathematics Python has the usual C arithmetic operators (+, -, *, /, %). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a new matrix multiply `@` operator is included in version 3.5. [71] Additionally, it has a unary operator `~`, which essentially inverts all the bytes of its one argument. For integers, this means `~x=-x-1`. Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)`, and `x >> y`, which shifts `x` to the right `y` places, the same as `x/(2**y)`. [73] The behavior of division has changed significantly over time: Python 2.1 and earlier use the C division behavior. The `/` operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. `7 / 3 == 2` and `-7 / 3 == -2`. Python 2.2 changes integer division to round towards negative infinity, e.g. `7 / 3 == 2` and `-7 / 3 == -3`. The floor division `//` operator is introduced. So `7 // 3 == 2`, `-7 // 3 == -3`, `7.5 // 3 == 2.0` and 6/13/2017 Python (programming language) - Wikipedia [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) 8/19 -7.5 // 3 == -3.0. Adding `from __future__ import division` causes a module to use Python 3.0 rules for division (see next). Python 3.0 changes `/` to be always floating-point division. In Python terms, the pre-3.0 `/` is classic division, the version-3.0 `/` is real division, and `//` is floor division. Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it

means that the equation $(a+b) // b == a // b + 1$ is always true. It also means that the equation $b * (a // b) + a \% b == a$ is valid for both positive and negative values of a . However, maintaining the validity of this equation means that while the result of $a \% b$ is, as expected, in the half-open interval $[0, b)$, where b is a positive integer, it has to lie in the interval $(b, 0]$ when b is negative. Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: `round(0.5)` is 1.0, `round(-0.5)` is -1.0. Python 3 uses round to even: `round(1.5)` is 2, `round(2.5)` is 2. Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression $a < b < c$ tests whether a is less than b and b is less than c . C-derived languages interpret this expression differently: in C, the expression would first evaluate $a < b$, resulting in 0 or 1, and that result would then be compared with c . Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type `int`, to arbitrary precision, belonging to the python type `long`, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the `int` and `long` types was eliminated; this behavior is now entirely contained by the `int` class.) The `Decimal` type/class in module `decimal` (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes. The `Fraction` type in module `fractions` (since version 2.6) provides arbitrary precision for rational numbers. Due to Python's extensive mathematics library, and the third-party library NumPy which further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation. Libraries Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it.

It is a type of signal dispensation in which input is an image, like video frame or photograph and output may be image or characteristics associated with that image.

Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

3.5 IMAGE PRE-PROCESSING

ADVANTAGES OF IMAGE PROCESSING:

The purpose of image processing is divided into 5 groups.

They are:

VISUALIZATION - Observe the objects that are not visible.

IMAGE SHARPENING AND RESTORATION - To create a better image.

IMAGE RETRIEVAL - Seek for the image of interest.

MEASUREMENT OF PATTERN – Measures various objects in an image.

IMAGE RECOGNITION – Distinguish the objects in an image.

FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:

1. IMAGE ACQUISITION:

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves pre-processing, such as scaling etc.

2. IMAGE ENHANCEMENT:

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. IMAGE RESTORATION:

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

4. COLOR IMAGE PROCESSING

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modelling and processing in a digital domain etc.

5. WAVELETS AND MULTIREOLUTION PROCESSING

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

6. COMPRESSION

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

7. MORPHOLOGICAL PROCESSING

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. SEGMENTATION

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

9. REPRESENTATION AND DESCRIPTION

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

As CNN based learning algorithm shows better performance on the classification issues, the rich labeled data could be more useful in the training stage. 3D object classification and pose estimation is a jointed mission aiming at separate different posed apart in the descriptor form.

In the training stage, we prepare 2D training images generated from our module with their class label and pose label. We fully exploit the information lies in their labels by using a triplet and pair-wise jointed loss function in CNN training.

As CNN based learning algorithm shows better performance on the classification issues, the rich labeled data could be more useful in the training stage. 3D object classification and pose estimation is a jointed mission aiming at separate different pose apart in the descriptor form.

In the training stage, we prepare 2D training images generated from our module with their class label and pose label. We fully exploit the information that lies in their labels by using a triplet and pair-wise jointed loss function in CNN training.

Both class and pose label are in consideration in the triplet loss. The loss score will be smaller when features from the same class and same pose is more similar and features from different classes or different poses will lead to a much larger loss score.

3.6 LIBRARIES USED

OPENCV

About the training and feature extraction process, it is a rough implementation by using OpenCV and Caffe from the idea of Paul Wohlhart. The principal purpose of this API is constructing a well labeled database from .ply models for CNN training with triplet loss and extracting features with the constructed model for prediction or other purpose of pattern recognition, algorithms into two main Class:

OpenCV is a library for real-time computer vision. It has very powerful functions that make the art of processing images and getting information about them easy.

In this post, we will review some of the functions that we used for making a 3D-reconstruction from an image in order to make an autonomous robotic arm.

OpenCV uses a pinhole camera model. This model works by projecting 3D points onto the image plane using a perspective transformation.

There are some functions of OpenCV that help us accomplish our goal. These functions work with a chess board model to calibrate the model, so the first thing is to get a chess board and take some photos of it. We took several photographs in order to get a better calibration.

It has come to my attention that most 3D reconstruction tutorials out there are a bit lacking. Don't get me wrong they're great, but they're fragmented or go too deep into the theory or a combination of both.

Worse yet they use specialized datasets (like Tsukuba) and this is a bit of a problem when it comes to using the algorithms for anything outside those datasets (because of parameter tuning).

I believe that the cool thing about 3D reconstruction (and computer vision in general) is to reconstruct the world around you, not somebody else's world (or dataset). This tutorial is a humble attempt to help you recreate your own world using the power of OpenCV.

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe

HISTORY:

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray

tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months¹ and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV

APPLICATIONS:

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Motion understanding
- Object identification

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. The API for these interfaces can be found in the online documentation..

Since version 3.4, **OpenCV.js** is a javascript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface

Numpy : NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones: ♣ A powerful N-dimensional array object ♣ Sophisticated (broadcasting) functions ♣ Tools for integrating C/C++ and Fortran code ♣ Useful linear algebra, Fourier Transform, and random number capabilities.

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

o SciPy : SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others. NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is

obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is build on the code of Numeric and the features of Numarray. o The Python Alternative To Matlab : Python in combination with Numpy, Scipy and Matplotlib can be used as a replacement for MATLAB. The combination of NumPy, SciPy and Matplotlib is a free (meaning both "free" as in "free beer" and "free" as in "freedom") alternative to MATLAB. Even though MATLAB has a huge number of additional toolboxes available, NumPy has the advantage that Python is a more modern and complete programming language and - as we have said already before - is open source. SciPy adds even more MATLAB-like functionalities to Python. Python is rounded out in the direction of MATLAB with the module Matplotlib, which provides MATLAB-like plotting functionality.

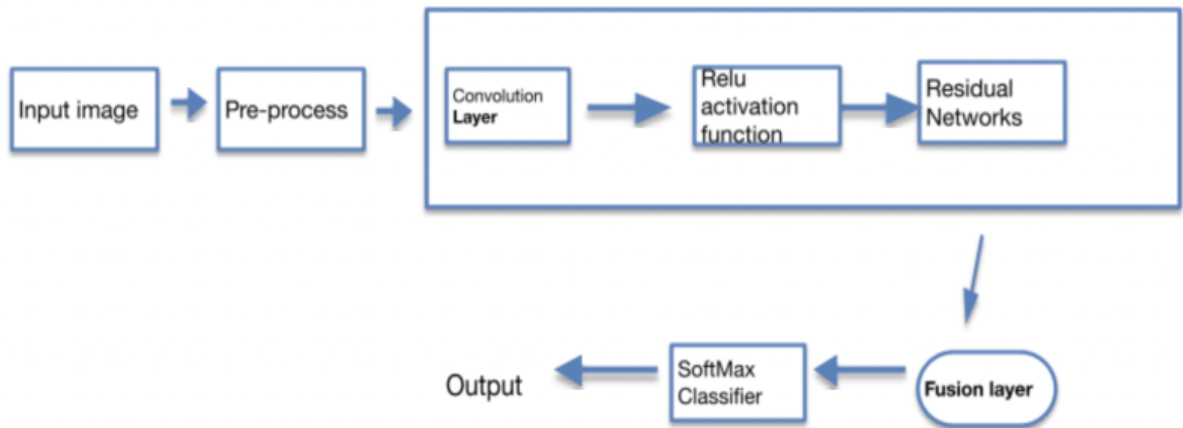
3.7 ARCHITECTURE AND ALGORITHM

A.DATABASE PREPROCESSING

We are considering datasets : Fer 2013, CK+, FER+ and RAF . Firstly we have to normalize the expression data to avoid different distributions of test and training data sets. This will improve the ability of the network for better feature extraction.Flipping, rotation and translation is done.

It is like cleaning the data for better and proper training.

B. MBCC CNN Model .



(Fig-2) Structure of MB-CC-CNN

After the input image gets pre-processing the model extracts features from the images and combines all those features into a single unit and then depending on the closest match the emotion gets classified.

Convolution Layer and ReLU activation function :-

The convolution layer contains filters which are used to extract features and they are small when compared with the real image.

If the input given is positive then the relu activation function returns the input as the output , if not it returns zero as output.

Residual Networks :-

Residual networks create a shortcut and transfer input directly and we have networks in networks which can extract more features. A single network will not be able to extract the patterns in a deep manner, so implementing some cross-connected networks helps for more feature extraction, therefore producing more accurate results

Fusion Layer :-

Fusion layer fuses all the patterns / features and fuses it together to make a single well structured feature.

SoftMax Classifier :-

It is used to classify the emotion according to the extracted features from the image.

Max pooling layer is used to get the maximum element of the feature. Convolutional layer maps feature to feature but ANN maps pixel to pixel , for any kind of image related projects it's good to go with CNN.

Implementation includes a facial data set that is pre-processed then Multi-branch-Cross-connected CNN is constructed for more feature extraction. Then the model is prepared for more facial feature extraction.

ALGORITHM :

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

Step 2 :Pre-processing of images.

Step 3 :Detection of a face from each image.

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

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

Step 6 :The numpy array gets passed into the Convolution2D layer.

Step 7 :Convolution generates feature maps.

Step 8 :Pooling method called MaxPooling2D that uses (2, 2) windows across the feature map only keeping the maximum pixel value.

Step 9 :During training, Neural network Forward propagation and Backward propagation performed on the pixel values.

Step 10 :The Softmax function presents itself as a probability for each emotion class. The model is able to show the detail probability composition of the emotions in the face

CHAPTER 4 RESULTS AND DISCUSSIONS

THE CK+ FACIAL EXPRESSION DATA SET The CK+ data set is an extension of the Cohn-Kanade data set and was published in 2010. The CK+ data set used contains seven types of expressions, including 135 angry images, 54 normal

images, 177 disgusted images, 75 fearful images, 207 happy images, 84 sad images, and 249 surprised images. For the CK+ data set, 10-fold cross validation is used for evaluation. The data set is randomly divided into 10 subsets, and the sampling interval is 10. Two subsets are used for testing and the rest are used for training..To check the performance we are considering some expression data sets

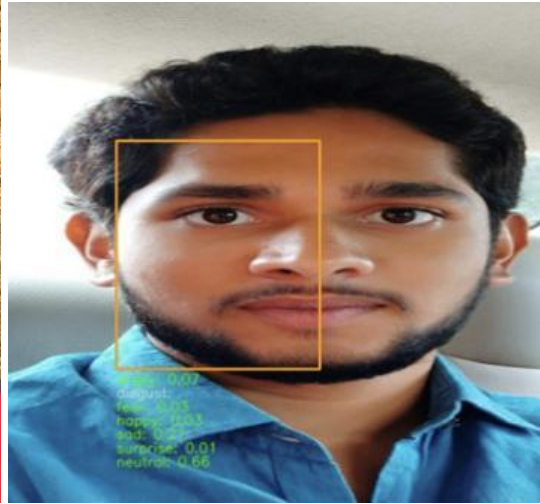
The proposed MBCC-CNN model is based on the fusion of the residual connection, Network in Network, and tree structure approaches. This model adds a data flow short-cut between networks. The Network in Network approach increases the receptive field of the convolution layer and improves the feature extraction ability of the model. In the multibranch network, the image features extracted from different branches are combined together, which effectively avoids the problem of insufficient feature extraction. The proposed MBCC-CNN method can extract image features effectively and improve the expression recognition accuracy. To fully verify the effectiveness of the proposed method, 32 related expression recognition methods are chosen for comparison under the same conditions. The comparison results of recognition accuracy on the Fer2013, CK+, FER+ and RAF data sets

1. CK+
2. FER2013
3. RAF

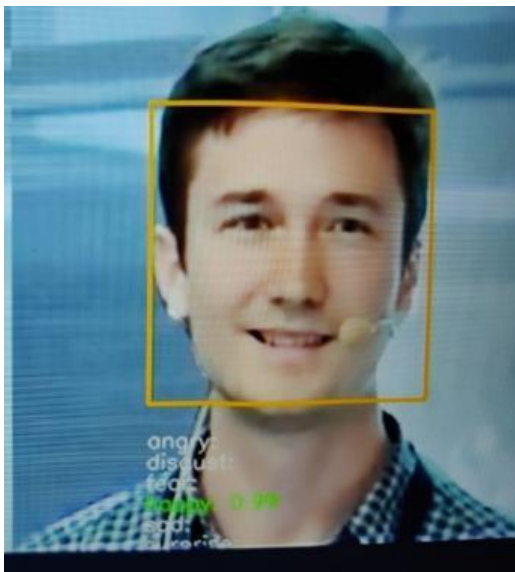
Output Screenshots



(fig-3)



(fig-4)



(fig-5)



(fig-6)

Figure 3 is showing higher value for happy i.e.0.83 , whereas fig-4 is neutral.

In the similar way the fig 5 is happy as the value for happy attribute is high when compared with other. Fig 6 is neutral .

Classification Performance analysis

Accuracy = (No.of correctly predicted / Total no.of samples) X 100

Accuracy comparison between other different methods with respect to FER2013 dataset

Model	Accuracy
AlexNet	66.67%
VGGNet	69.45%
CNN	70.71%
MB-CNN	72.53%

Accuracy comparison between other different methods with respect to CK+ dataset

Model	Accuracy
WM CNN-LSTM	97.46%

AlexNet	96.43%
VGGNet	95.28%
CNN	96.21%
MB-CNN	98.53%

Accuracy comparison between other different methods with respect to RAF dataset

Model	Accuracy
ResNet	74.76%
AlexNet	85.32%
CNN	85.01%
MB-CNN	87.51%

The above results show that the proposed model is more accurate in feature extraction with high accuracy.

So we can clearly say that the MB-CNN is very accurate and can extract more features than other models.

In this part, first, the expression data sets used in this paper are introduced. Then, the confusion matrix analysis, recognition performance analysis, and thermodynamic diagram analysis are carried out for the proposed MBCC-CNN model. Then, the expression recognition system based on the MBCC-CNN model is designed. Finally, the proposed method is compared with some recent works with different evaluation indicators. All the experiments in this paper are performed on JetBrains PyCharm2017.1 x64, Kera2.1.4, and GeForce 940MX GPU. The parameter settings of our model.

CHAPTER 5 CONCLUSION AND FUTURE WORK

The proposed method combines residual networks, network in networks and multi branch tree structure approaches, which extracts features in a better way. Actually we are using multi networks so that each network can extract some feature independently and finally fuse altogether to a single feature, because of this we can be able to extract more features. Results are sent to the SoftMax classifier which classifies according to the features extracted. The results have shown that the model is good in feature extraction and accuracy.

The proposed method combines residual networks, network in networks and multi branch tree structure approaches, which extracts features in a better way. Actually we are using multi networks so that each network can extract some feature independently and finally fuse altogether to a single feature, because of this we can be able to extract more features. Results are sent to the SoftMax classifier which classifies according to the features extracted. The results have shown that the model is good in feature extraction and accuracy.

5.1 REFERENCES

- [1] M.-I. Georgescu, R. T. Ionescu, and M. Popescu, "Local learning for face expression identification using deep and handmade features," *Access, IEEE*, vol.7, pp. 64827–64836, 2019.
- [2] X. Sun and M. Lv, "FFacial expression recognition based on a hybrid model combining deep and shallow features," *Cognit. Comput.*, vol. 11, no. 4, pp. 587–597, Aug. 2019.
- [3]"Improved face expression detection based on DWT feature for deep CNN," R. Bendjillali, M. Beladgham, K. Merit, and A. Taleb-Ahmed, *Electronics*, vol. 8, no. 3, p. 324, Mar. 2019.
- "Facial expression recognition with deep learning," *Proc. 10th Int. Conf. Internet Multimedia Comput. Service*, 2018, p. 10.
- [4] X. M. Wang, J. Huang, J. Zhu, and M. Yang, "Facial expression recognition with deep learning," *Proc. 10th Int. Conf. Internet Multimedia Comput. Service*, 2018, p. 10.
- [5]M. H. A. Latif, H. M. Yusof, S. N. Sidek, and N. N. Sidek are the acronyms for M. H. A. Latif, H. M. Yusof, S. N. Sidek, and N. N. Sidek, respectively. 2015 IEEE International Symposium on Robotics and Intelligent Sensors, pages 214–219 (IRIS). 214–219 in IEEE International Symposium on Robotics and Intelligent Sensors (IRIS), 2015.
- [6] Y. Li, J. Zeng, S. Shan, and X. Chen, "Occlusion aware face emotion detection using CNN with attention mechanism," *IEEE Transactions on Image Processing*, vol. 28, no. 5, pp. 2439–2450, May 2019.
- [7] S. Shi, H. Si, J. Liu, and Y. Liu, "Facial expression recognition based on Gabor features of salient patches and ACI-LBP," *J. Intell. Fuzzy Syst.*, vol. 34, no. 4, pp. 2551–2561, Apr. 2018.

- [8] M. Z. Uddin, M. M. Hassan, A. Almogren, A. Alamri, M. Alrubaian, and G. "Identification of facial expressions utilising local direction-based robust features and a deep belief network," *Fortino explains.. Access, IEEE*, vol. 5, pp. 4525–4536, Mar. 2017.
- [9] S.Zhao,H.Cai,H.Liu,J.Zhang,andS.Chen,“Featureselectionmechanism in CNNs for facial expression recognition,” in *Proc. BMVC*, 2018, p. 317.
- [10] E. Vezzetti, S. Tornincasa, S. Moos, and F. Marcolin, “3D human face analysis: Automatic expression recognition,” *Biomed. Eng., Calgary*, pp. 24–30, Feb. 2016.
- [11] G. Zeng, J. Zhou, X. Jia, W. Xie, and L. Shen, “Hand-crafted feature guided deep learning for facial expression recognition,” in *Proc. 13th IEEE Int. Conf. Autom. Face Gesture Recognit. (FG)*, Xi’an, China, May 2018, pp. 423–430.
- [12] M.-I. Georgescu, R. T. Ionescu, and M. Popescu, “Local learning with deep and handcrafted features for facial expression recognition,” *IEEE Access*, vol. 7, pp. 64827–64836, 2019.]
- [13] X. Sun and M. Lv, “FFacial expression recognition based on a hybrid model combining deep and shallow features,” *Cognit. Comput.*, vol. 11, no. 4, pp. 587–597, Aug. 2019.
- [14] R. Bendjillali, M. Beladgham, K. Merit, and A. Taleb-Ahmed, “Improved facial expression recognition based on DWT feature for deep CNN,” *Electronics*, vol. 8, no. 3, p. 324, Mar. 2019.
- [15] X. M. Wang, J. Huang, J. Zhu, and M. Yang, “Facial expression recognition with deep learning,” *Proc. 10th Int. Conf. Internet Multimedia Comput. Service*, 2018, p. 10.

5.2 APPENDICES

5.2.1 Code implementation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import csv
import os
import re
import time
from pathlib import Path
from typing import Optional, Union
from zipfile import ZipFile

import cv2
import numpy as np
import pandas as pd

from tqdm import tqdm

from .logger import log

class Video(object):
    def __init__(
        self, video_file:str, outdir:str="output", first_face_only:bool=True,
        tempfile:Optional[str]=None
    ):
        """Video class for extracting and saving frames for emotion detection.
        :param video_file - str
        :param outdir - str
        :param tempdir - str
        :param first_face_only - bool
        :param tempfile - str
        """
        assert os.path.exists(video_file), "Video file not found at {}".format(
            os.path.abspath(video_file)
        )
        self.cap = cv2.VideoCapture(video_file)
        if not os.path.isdir(outdir):
            os.makedirs(outdir, exist_ok=True)
        self.outdir = outdir

        if not first_face_only:
            log.error("Only single-face charting is implemented")
        self.first_face_only = first_face_only
        self.tempfile = tempfile
        self.filepath = video_file
        self.filename = "".join(self.filepath.split("/")[-1])
```

```

@staticmethod
def get_max_faces(data: list) -> int:
    """Get max number of faces detected in a series of frames, eg 3"""
    max = 0
    for frame in data:
        for face in frame:
            if len(face) > max:
                max = len(face)
    return max

@staticmethod
def _to_dict(data: Union[dict, list]) -> dict:
    emotions = []

    frame = data[0]
    if isinstance(frame, list):
        try:
            emotions = frame[0]["emotions"].keys()
        except IndexError:
            raise Exception("No data in 'data'")
    elif isinstance(frame, dict):
        return data

    dictlist = []

    for data_idx, frame in enumerate(data):
        rowdict = {}
        for idx, face in enumerate(list(frame)):
            if not isinstance(face, dict):
                break
            rowdict.update({"box" + str(idx): face["box"]})
            rowdict.update(
                {emo + str(idx): face["emotions"][emo] for emo in emotions}
            )
        dictlist.append(rowdict)
    return dictlist

def to_pandas(self, data: Union[pd.DataFrame, list]) -> pd.DataFrame:
    """Convert results to pandas DataFrame"""
    if isinstance(data, pd.DataFrame):
        return data

    if not len(data):
        return pd.DataFrame()
    datalist = self._to_dict(data)
    df = pd.DataFrame(datalist)
    if self.first_face_only:
        df = self.get_first_face(df)
    return df

```

```

@staticmethod
def get_first_face(df: pd.DataFrame) -> pd.DataFrame:
    assert isinstance(df, pd.DataFrame), "Must be a pandas DataFrame"
    try:
        int(df.columns[0][-1])
    except ValueError:
        # Already only one face in df
        return df

    columns = [x for x in df.columns if x[-1] == "0"]
    new_columns = [x[:-1] for x in columns]
    single_df = df[columns]
    single_df.columns = new_columns
    return single_df

@staticmethod
def get_emotions(df: pd.DataFrame) -> list:
    """Get emotion columns from results."""
    columns = [x for x in df.columns if "box" not in x]
    return df[columns]

def to_csv(self, data, filename="data.csv"):
    """Save data to csv"""
    def key(item):
        key_pat = re.compile(r"^(D+)(d+)$")
        m = key_pat.match(item)
        return m.group(1), int(m.group(2))

    dictlist = self._to_dict(data)
    columns = set().union(*(d.keys() for d in dictlist))
    columns = sorted(columns, key=key) # sort by trailing number (faces)

    with open("data.csv", "w", newline="") as csvfile:
        writer = csv.DictWriter(csvfile, columns, lineterminator="\n")
        writer.writeheader()
        writer.writerows(dictlist)
    return dictlist

def analyze(
    self,
    detector, # fer.FER instance
    display: bool = False,
    output: str = "csv",
    frequency: Optional[int] = None,
    max_results: int = None,
    save_fps: Optional[int] = None,
    video_id: Optional[str] = None,
    save_frames: bool = True,
    save_video: bool = True,
    annotate_frames: bool = True,

```

```

zip_images: bool = True,
detection_box: Optional[dict] = None
) -> list:
    """Recognize facial expressions in video using `detector`.

```

Args:

```

    detector (fer.FER): facial expression recognizer
    display (bool): show images with cv2.imshow
    output (str): csv or pandas
    frequency (int): inference on every nth frame (higher number is faster)
    max_results (int): number of frames to run inference before stopping
    save_fps (bool): inference frequency = video fps // save_fps
    video_id (str): filename for saving
    save_frames (bool): saves frames to directory
    save_videos (bool): saves output video
    annotate_frames (bool): add emotion labels
    zip_images (bool): compress output
    detection_box (dict): dict with bounding box for subimage (xmin, xmax, ymin, ymax)

```

Returns:

```

    data (list): list of results

```

```

    """
    data = []
    if frequency is None:
        frequency = 1
    else:
        frequency = int(frequency)

    results_nr = 0

    # Open video
    assert self.cap.open(self.filepath), "Video capture not opening"
    self.__emotions = detector._get_labels().items()
    self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    pos_frames = self.cap.get(cv2.CAP_PROP_POS_FRAMES)
    assert int(pos_frames) == 0, "Video not at index 0"

    frameCount = 0
    height, width = (
        int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)),
        int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
    )

    fps = self.cap.get(cv2.CAP_PROP_FPS)
    length = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
    assert fps and length, "File {} not loaded".format(self.filepath)

```

```

if save_fps is not None:
    frequency = fps // save_fps
    log.info("Saving every {} frames".format(frequency))

log.info(
    "{:.2f} fps, {} frames, {:.2f} seconds".format(fps, length, length / fps)
)

capture_duration = 1000 / fps

if save_frames:
    os.makedirs(self.outdir, exist_ok=True)
    log.info(f"Making directories at {self.outdir}")
    root, ext = os.path.splitext(os.path.basename(self.filepath))
    outfile = os.path.join(self.outdir, f"{root}_output{ext}")

if save_video:
    videowriter = self._save_video(outfile, fps, width, height)

pbar = tqdm(total=length, unit='frames')
while self.cap.isOpened():
    start_time = time.time()
    ret, frame = self.cap.read()
    if not ret: # end of video
        break
    if frameCount % frequency != 0:
        frameCount += 1
        continue

    if detection_box is not None:
        try:
            frame = self._crop(frame, detection_box)
        except Exception as e:
            log.error(e)
            break

    padded_frame = detector.pad(frame)
    try:
        # Get faces with emotions
        faces = detector.detect_emotions(padded_frame)
        if detection_box is not None:
            try:
                for face in faces:
                    original_box = face.get("box")
                    face["box"] = (original_box[0] + detection_box.get("x_min"), original_box[1] +
detection_box.get("y_min"), original_box[2], original_box[3])
            except Exception as e:
                log.error(e)

    except Exception as e:

```

```

log.error(e)
break

# Save images to `self.outdir`
imgpath = os.path.join(
    self.outdir, (video_id or root) + str(frameCount) + ".jpg"
)
if save_frames and not annotate_frames:
    cv2.imwrite(imgpath, frame)

if display or save_video or annotate_frames:
    assert isinstance(faces, list), type(faces)
    for face in faces:
        bounding_box = face["box"]
        emotions = face["emotions"]

        cv2.rectangle(
            frame,
            (bounding_box[0] - 40, bounding_box[1] - 40),
            (
                bounding_box[0] - 40 + bounding_box[2],
                bounding_box[1] - 40 + bounding_box[3],
            ),
            (0, 155, 255),
            2,
        )

        for idx, (emotion, score) in enumerate(emotions.items()):
            color = (211, 211, 211) if score < 0.01 else (0, 255, 0)
            emotion_score = "{}: {}".format(
                emotion, "{:.2f}".format(score) if score > 0.01 else ""
            )
            cv2.putText(
                frame,
                emotion_score,
                (
                    bounding_box[0] - 40,
                    bounding_box[1] - 40 + bounding_box[3] + 30 + idx * 15,
                ),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5,
                color,
                1,
                cv2.LINE_AA,
            )
        if display:
            cv2.imshow("Video", frame)
        if save_frames and annotate_frames:
            cv2.imwrite(imgpath, frame)
        if save_video:

```

```

        videowriter.write(frame)
        results_nr += 1

    if display or save_video:
        remaining_duration = max(
            1, int((time.time() - start_time) * 1000 - capture_duration)
        )
        if cv2.waitKey(remaining_duration) & 0xFF == ord("q"):
            break
    else:
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    frameCount += 1
    if faces:
        data.append(faces)
    if max_results and results_nr > max_results:
        break

    pbar.update(1)

pbar.close()
self.cap.release()
if display or save_video:
    videowriter.release()
if save_video:
    log.info(
        "Completed analysis: saved to {}".format(self.tempfile or outfile)
    )
    if self.tempfile:
        os.replace(self.tempfile, outfile)

if save_frames and zip_images:
    log.info("Starting to Zip")
    outdir = Path(self.outdir)
    zip_dir = outdir / 'images.zip'
    images = sorted(list(outdir.glob("*.jpg")))
    total = len(images)
    i = 0
    with ZipFile(zip_dir, 'w') as zip:
        for file in images:
            zip.write(file, arcname=file.name)
            os.remove(file)
            i += 1
            if i%50 == 0: log.info(f"Compressing: {i*100 // total}%")
    log.info("Zip has finished")

if output == "csv":
    return self.to_csv(data)
elif output == "pandas":

```

```

        return self.to_pandas(data)
    else:
        raise NotImplementedError(f"{output} is not supported")
    return data

def _save_video(self, outfile:str, fps: int, width: int, height: int):
    if os.path.isfile(outfile):
        os.remove(outfile)
        log.info("Deleted pre-existing {}".format(outfile))
    if self.tempfile and os.path.isfile(self.tempfile):
        os.remove(self.tempfile)
    fourcc = cv2.VideoWriter_fourcc("m", "p", "4", "v")
    videowriter = cv2.VideoWriter(
        self.tempfile or outfile, fourcc, fps, (width, height), True
    )
    return videowriter

    @staticmethod
    def _crop(self, frame, detection_box):
        crop_frame = frame[
            detection_box.get("y_min"): detection_box.get("y_max"),
            detection_box.get("x_min"): detection_box.get("x_max")]
        return crop_frame

    def __del__(self):
        try:
            cv2.destroyAllWindows()
        except:
            pass

#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs
import os
import re

from setuptools import find_packages, setup

NAME = "fer"
PACKAGES = find_packages(where="src")
HERE = os.path.abspath(os.path.dirname(__file__))

def read(*parts):
    """
    Build an absolute path from *parts* and return the contents of the
    resulting file. Assume UTF-8 encoding.

```

```

"""
with codecs.open(os.path.join(HERE, *parts), "rb", "utf-8") as f:
    return f.read()

META_PATH = os.path.join("src", NAME, "__init__.py")
META_FILE = read(META_PATH)
KEYWORDS = ["facial expressions", "emotion detection", "faces", "images"]

def find_meta(meta):
    """
    Extract __*meta*__ from META_FILE.
    """
    meta_match = re.search(
        r"__{meta}__ = [\"]([^\"]*)[\"]".format(meta=meta), META_FILE,
        re.M)
    if meta_match:
        return meta_match.group(1)
    raise RuntimeError("Unable to find __{meta}__ string.".format(meta=meta))

URL = find_meta("url")
PROJECT_URLS = {
    "Documentation": URL,
    "Bug Tracker": "https://github.com/hynek/argon2_cffi/issues",
    "Source Code": "https://github.com/hynek/argon2_cffi",
}
CLASSIFIERS = [
    "Intended Audience :: Developers",
    "License :: OSI Approved :: MIT License",
    "Intended Audience :: Education",
    "Intended Audience :: Science/Research",
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.6",
    "Topic :: Scientific/Engineering :: Mathematics",
    "Topic :: Software Development :: Libraries :: Python Modules",
    "Topic :: Software Development :: Libraries",
]

PYTHON_REQUIRES = ">= 3.6"

INSTALL_REQUIRES = [
    "matplotlib", "opencv-contrib-python", "keras>=2.0.0", "pandas",
    "requests", "mtcnn>=0.1.1", "tqdm"
]

EXTRAS_REQUIRE = {"docs": ["sphinx"], "tests": ["coverage", "pytest"]}
EXTRAS_REQUIRE["dev"] = (EXTRAS_REQUIRE["tests"] + EXTRAS_REQUIRE["docs"] +
    ["wheel", "pre-commit"])

```

```

VERSION = find_meta("version")

# README.md
LONG = open('README.md').read()

setup(
    name=NAME,
    version=find_meta("version"),
    author=find_meta("author"),
    author_email=find_meta("email"),
    maintainer=find_meta("author"),
    maintainer_email=find_meta("email"),
    description=find_meta("description"),
    license=find_meta("license"),
    keywords=KEYWORDS,
    url=URL,
    packages=PACKAGES,
    long_description=LONG,
    classifiers=CLASSIFIERS,
    install_requires=INSTALL_REQUIRES,
    extras_require=EXTRAS_REQUIRE,
    python_requires=PYTHON_REQUIRES,
    include_package_data=True,
    package_dir={"": "src"},
    zip_safe=False,
)

```

