Smart Selfie Capture using Facial Landmark Plot Point

Ashish Chopra
Multimedia part
Samsung R&D Institute
Noida, India

Yash Awasthi
Multimedia part
Samsung R&D Institute
Noida, India

Madhavi
Multimedia part
Samsung R&D Institute
Noida, India

Vinita Sharma
Multimedia part
Samsung R&D Institute
Noida, India

***Abstract*— This paper investigates the situations for automatic selfie capture when the user selects an image with some facial expressions as theme for template and then user try to imitate the facial expressions from template, camera click automatically gets triggered for a specific period and captures the best face which matches the expression of the selected template. We plan on doing this using facial recognition by extracting the plot points of facial expressions of the template and comparing them to the frame with face captured by camera sensor. We will be comparing different parts of face which involve movements and making of expressions like eyes, eyebrows, mouth, etc. for comparing the expressions and the corner boundary points of face for the orientation of face. We will be assigning different weight balance to different parameters depending on importance of it for the expression and the average of all the scores will lead to the final score for comparison to find the best match from frames. If score is better than the previous captured image the frame will be send for processing and that image will be captured.**

***Keywords – Automatic Selfie, Image Processing, Neural Networks, Facial Points, Feature Extraction*.**

I. INTRODUCTION

The usage of mobile phones has increased multi-fold in the recent decades mostly because of its utility in most of the aspects of daily life, such as communications, entertainment, financial transactions, etc. This has led to the explosion of multimedia content. With the advancement in storage space of mobile phones more photos and videos are captured every day. It offers an immense scope in Image Processing. In the current scenario some models are known to extract the facial features or the plot points of an image. In[1] plot points of an image are extracted and prediction about facial beauty were made. Paper[2] extracts facial features for face recognition. Same target was achieved by [3]whereas [4]extracted facial features for name tagging only.

In the current scenario there is no such model or method, such that a template image is provided to a user and using facial points of that image to match with the face of user captured by the camera sensor and capturing the image with the highest normalised score of accuracy to the image selected as template for expressions.

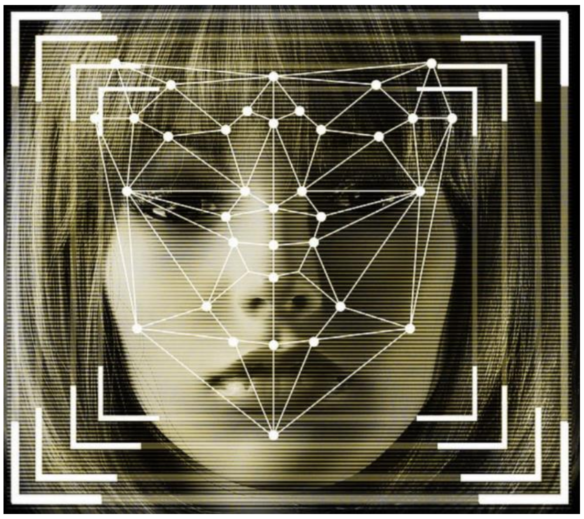


Fig 1: Shows basic facial plot points are map of facial points used in security, entertainment and other applications

There are 3 basic stages of how a facial feature extraction[5] model works: face location determination stage, facial landmark detection stage and feature extraction stage. The first stage deals with face detection algorithms that are to be implemented for the facial recognition, which mainly deals with image pre-processing and normalizing the images to eliminate redundant areas. In the second stage, the plot points are extracted from the face. In the third stage, it is our facial fiducial landmarks: eyebrows, eyes, nose and mouth, are identified as the critical features for points are hence extracted

II. RELATED CONCEPTS

A. *Face Detection*

The first process in automatic face recognition is face detection[6], which is a specific type of object detection. The accurate detection of human faces is much more important process. The face detection is a computer vision technology to detect the frontal faces of the human from digital images or videos. This paper uses computer vision technology for frontal faces detection[7] by facial landmark detection of nose and upper lips. Nose is the center point of the facial landmarks.

B. Dlib Face Alignment

Dlib[8][9] is an open source library which provides best environment for developing software based on machine learning in C++. The core of the Dlib is linear algebra with Basic Linear Algebra Subprograms (BLAS)[10]. It is mainly used for implementation of Bayesian networks and Kernel-based algorithms for classification, clustering, anomaly detection, regression and feature ranking. Dlib library has two main components, linear algebra and machine learning tools.[11].

1) Linear Algebra

The linear algebra component is based on the template expression techniques stated in the Blitz++ numerical software by Veldhuizen and Ponnambalam (1996). Dlib used with BLAS gains the performance and speed of the code as optimized libraries. Dlib can perform any transformations on all expressions by invoking the appropriate BLAS which enables user to write equations in the most intuitive form, thus leaving the details of the software optimization to the library.

2) Machine Learning Tools

The main goal of this component is to provide simple and high modular architecture for kernel based algorithms. Dlib can be implementing on images, column vectors or any form of structured data. Implementation of the algorithm is entirely different from data on which they operate. The flexibility of the Dlib is direct operation on any objects which making it to apply custom kernels where kernel operates on object than fixed length vector. This paper deployed the Dlib for face alignment of the frontal faces; the alignment could be effective for 45^0 of the alignment[12].

III. PROPOSED METHODOLOGY

A. Facial Feature Extraction

There are multiple models which are used for detection of face and extraction of facial features from it. One such example of facial landmark detection[13] model present in open source is Dlib's68-facelandmark: "shape_predictor_68_face_landmarks.dat"[14]. Dlib's 68-face landmark model shows how we can access the face features like eyes, eyebrows, nose, etc. But sometimes, we don't want to access all features of the face and want only some features likes, lips for lipstick application. So that is also possible using custom training of the Dlib's 68-landmark models.

There are multiple other models with different number of facial plot points and as we increase or decrease the number of plot points the accuracy also increases or decreases. Some of the models used by us in this research paper are-

- 12-point model
- 35-point model
- Dlib's 68-point model

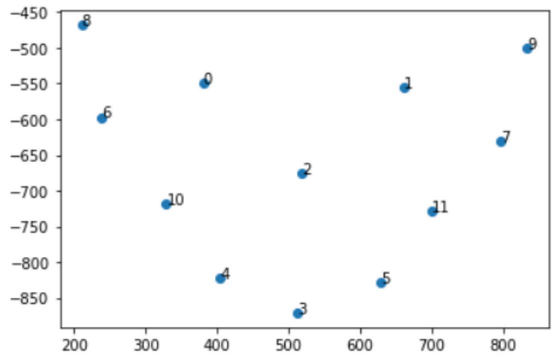


Fig 2: A 12-point model where 0, and 1 represents eyes, 2 nose tip, 3, 4 and 5 represent lips and so on.

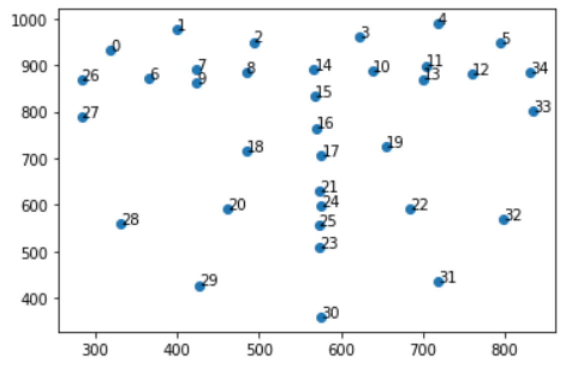


Fig 3: A 35-point facial model

Left eyebrow	[0, 2]
Right eyebrow	[3, 5]
Left eye	[6, 9]
Right eye	[10, 13]
Nose bridge	[14, 16]
Nose tip	[17, 19]
Top lip	[20, 22] + 24
Bottom lip	20 + 22 + 23 + 25
Chin / Jaw line	[26, 34]

Table 1: Plot points numberings of face features for the 35-point model

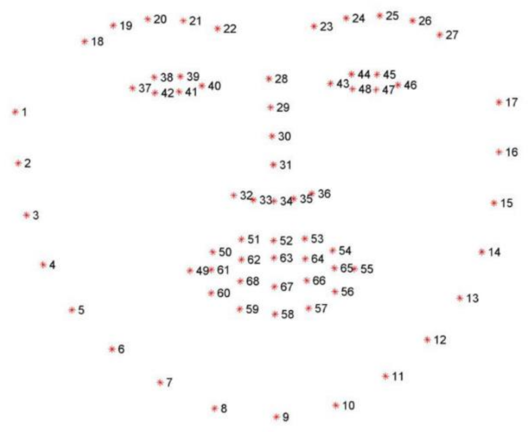


Fig 4: Dlib 68 point model

Left eyebrow	[18, 22]
Right eyebrow	[23, 27]
Left eye	[37, 42]
Right eye	[43, 48]
Nose bridge	[28, 31]
Nose tip	[32, 36]
Top lip	[49, 55] + [61, 65]
Bottom lip	49 + 55 + [66, 68]
Chin / Jaw line	[1, 17]

Table 2: Plot points numberings of face features for the 68-point model

```
#initialize dlib's face detector and create facial landmark predictor

Detector = dlib.get_frontal_face_detector ()
Predictor = dlib.shape_predictor (predictor_path)

#load the input image resize it and convert into grayscale

#detect faces in grayscale image
Rects = Detector(gray,1)

#loop over the face detection
for(i,rect) in enumerate(rects):

#determine the facial landmarks for the face region,
Shape = Predictor(gray,rect)

#array of landmark coordinates
Shape = face_utils.shape_to_np(Shape)
```

Fig 5: Code for face landmark detection

B. Face Points vector Computation

The next step after facial points extraction was to normalize the facial feature points, so as the scale for both template image points as well as the bitmap captured by camera sensor were on same scale and comparable for next process of similarity computation.

We used point 17 of the 35-point model to compute the vectors for all points.

$$V_{xi} = X_i – X_{17} \quad \text{and} \quad V_{Yi} = Y_i – Y_{17}$$

e.g. ,

$$V_{x0} = XX_0 – X_{17} \quad \text{and} \quad V_{Y0} = Y_0 – Y_{17}$$
$$V_{x1} = X_1 – X_{17} \quad \text{and} \quad V_{Y1} = Y_1 – Y_{17}$$
$$V_{x2} = X_2 – X_{17} \quad \text{and} \quad V_{Y2} = Y_2 – Y_{17}$$
$$V_{x3} = X_3 – X_{17} \quad \text{and} \quad V_{Y3} = Y_3 – Y_{17}$$

.

..

$$V_{x33} = X_{33} – X_{17} \quad \text{and} \quad V_{Y33} = Y_{33} – Y_{17}$$
$$V_{x34} = X_{34} – X_{17} \quad \text{and} \quad V_{Y34} = Y_{34} – Y_{17}$$

Where ,

V_{xi} = *x*-component of vector V for point i,
V_{yi} = *y*-component of vector V for point i,
X_i = *x*-coordinate for point i,
Y_i = y- coordinate for point i.

Vector V_i (magnitude) = sqrt $(V_{xi}^2 + V_{yi}^2)$

Vector V_i (direction) = V_{Yi} / V_{xi}

C. Vector Selection

Vector selection is an important procedure before computation of similarity between template and sensor data[15]. In case if all vectors are considered it will increase the similarity score between the faces. It will happen as there are many points in face which are same for different facial expressions i.e. facial points corresponding to fixed facial parts and do not contribute in making different expression using our face, e.g. – Jawline points (point 26 to 34 for 35 point model) (not all points were excluded 29, 30 and 31 were included for similarity). There are the points which only vary in different person based on face shape and therefore may be different for different people and hence can also reduce the similarity expression for similar expressions. The face parts which involve movement and contribute for expressions are eyebrows, eyes, mouth, nose and chin. The points used for comparing similarity of template and bitmap from sensor for 35 point model are 0-13, 17-35 and 29-31.

D. Cosine Similarity

Cosine similarity measures the similarity between two vectors[16] of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a *term-frequency vector*.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let ***x*** and ***y*** be two vectors for comparison. Using the cosine measure as a similarity function, we have

$$\mathrm{Sim}(x, y) = x.\mathbf{y} / ||\mathbf{x}|| * ||\mathbf{y}||$$

where, $||\boldsymbol{x}||$ is the Euclidean normal of vector x=(x0, x1 ,…, xn), defined as sqrt($x0^2 + x1^2 + x2^2 + \ldots + xn^2$). Conceptually, it is the length of the vector. Similarly, $||\boldsymbol{y}||$ is the Euclidean norm of vector ***y***. The measure computes the cosine of the angle between vectors ***x*** and ***y***. A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

$$\text{Similarity} = \cos(\theta) = (A\mathbf{.}B) / ||A||\,||B||$$

$$= (\Sigma A_i * B_i) / \mathrm{sqrt}\, \Sigma (A_i)^2 * \mathrm{sqrt}\, \Sigma (B_i)^2$$

Fig 5: Representation of formula for cosine similarity

When attributes are binary-valued, the cosine similarity function can be interpreted in terms of shared features or attributes. Suppose an object ***x*** possesses the *i*th attribute if $x_i = 1$. Then $\boldsymbol{x}^t \cdot \boldsymbol{y}$ is the number of attributes possessed (i.e., Sim (x, y) = 0.94

```
int cosSimilarity (ArrayList<int> template,
ArrayList<int> sensor_data)
{
        double cosSimilarityScore = 0;
        cosSimilarityScore =
CalculateCosineSimilarity (template,
sensor_data);
        return cosSimilarityScore;
}

private static double
CalculateCosineSimilarity (ArrayList<int>
template, ArrayList<int> sensor_data)
{
        double dotProduct = DotProduct
(template, sensor_data);
        double magnitudeOfTemplate =
Magnitude (template);
        double magnitudeOfSensorData =
Magnitude (sensor_data);
        if (magnitudeOfTemplate != 0.0 &&
magnitudeOfSensorData != 0.0)
        {
                return dotProduct /
(magnitudeOfTemplate *
magnitudeOfSensorData);
        }
        else
        {
                return 0.0;
        }
}

private static double DotProduct
(ArrayList<int> template, ArrayList<int>
sensor_data)
{
        //Not validating inputs here for
simplicity.
        double dotProduct = 0;
        for (var i = 0; i <
template.length; i++)
        {
                dotProduct += (template[i]
*sensor_data [i]);
        }

        return dotProduct;
}

private static double Magnitude
(ArrayList<int> input)
{
        return Math.Sqrt (DotProduct
(input, input));
}
```

Fig 6: Code for cosine similarity.

shared) by both ***x*** and ***y***, and |***x***||***y***| is the *geometric mean* of the number of attributes possessed by ***x*** and the number possessed by ***y***. Thus, *sim*(***x***, ***y***) is a measure of relative possession of common attributes.

Example of cosine similarity:-
Suppose that x and y are the first two term-frequency vectors. x = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0) and y = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1). We need to find how similar are x and y?

Using the cosine similarity between the two vectors, we get:

$\mathbf{x}^t\mathbf{y}$ = 5*3 + 0*0 + 3+2 + 0*0 + 2*1 + 0*1 + 0*1 + 2*1 + 0*0 + 0*1 = 25
$||\mathbf{x}|| = \text{sqrt}\ (5^2 + 0^2 + 3^2 + 0^2 + 2^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2) = 6.48$
$||\mathbf{y}|| = \text{sqrt}\ (3^2 + 0^2 + 2^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2) = 4.12$

E. Face Orientation Error

Face orientation[17] similarity is the check for comparing the similarity in orientation of both the template face and bitmap from sensor i.e. the direction in which both the faces are looking or in which direction are the faces tilted. There are two types are orientation which are computed. These are:-

- Left-Right**:** The left-right orientation is computed by calculating ratio of distance of x-coordinates between points 7 and 14 to 14 and 11 ((x_7 – $x1_4$) / (x_{14}-x_{11})). Ideally this value should be close to 1 when a person is looking straight into the camera. When the person's head tilts towards left the value of left-right orientation ratio becomes greater than 1 and keeps on increasing as the person keeps tilting its head towards left. Similarly when a person looks towards right the value of left-right orientation ratio becomes lesser than 1 and keeps on decreasing as the person keeps tilting its head towards right.

- Up-Down**:** Similar to the left-right orientation ratio, in up-down orientation ratio of distance of y-coordinates between points 14 and 17 to 17 and 30 ((y_{14} – y_{17}) / (y_{17}-y_{30})). Ideally this value should be close to 0.5 when a person is looking straight into the camera. When the person's looks up the value of up-down orientation ratio becomes less than 0.5 and keeps on decreasing as the person keeps tilting its head upwards. Similarly when a person looks down the value of up-down orientation ratio becomes greater than 0.5 and keeps on increasing as the person keeps tilting its head downwards.

The error for both the ratios are calculated by:-
$E_{lr} = (LR_{sensor} - LR_{template}) / LR_{template}$
$E_{ud} = (UD_{sensor} - UD_{emplate}) / LR_{template}$

Here,
E_{lr} = Error left-right orientation ratio
$LR_{template}$ = Left-right orientation ratio for template face
LR_{sensor} = Left-right orientation ratio for sensor face
E_{ud} = Error up-down orientation ratio
$UD_{emplate}$ = Up-down orientation ratio for template face

UD_{sensor} = Up-down orientation ratio for sensor face

The root mean squares of both the errors are compared using formula:-
RMSE = sqrt(E_{lr} * E_{lr} + 4 * E_{ud} * E_{ud})
Here a factor of 4 is multiplied to E_{ud} its value is normal case is close to 0.5 whereas value of E_{lr} in normal case is close to 1 and we are squaring both the errors before taking square root.

1. Compute left-right and up-down ratio for both template and sensor image:
   LR=x_7 -x_{14}/x_{14} + x_{11}
   UD=y_{14} -y_7/y_{17} - y_{30}

2. E_{lr} = (LR_{sensor} - $LR_{template}$) / $LR_{template}$
   E_{ud} = (UD_{sensor} - $UD_{emplate}$) / $LR_{template}$

3.Calculate root mean square error
RMSE = sqrt(E_{lr} * E_{lr} + 4 * E_{ud} * E_{ud})

Fig 7: Code for face orientation error.

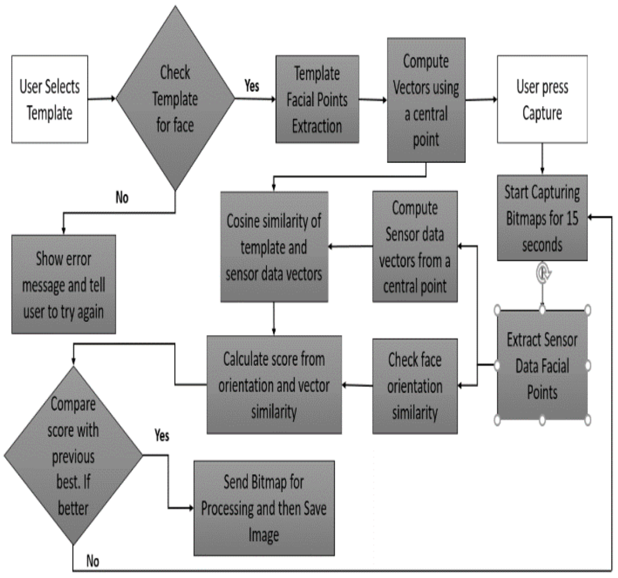
F. *Computing result by similarity and error score and capturing image*

Once we have got both the cosine similarity score and face orientation error score we compare it to the cosine similarity score and face orientation error score of previous captured image[18]. When no image has been saved, the initial value are:-
Cosine similarity score = 0
Face orientation error score = Integer.MAX_VALUE.
If both the score of bitmap are better than previous captured image we send that bitmap for processing and save that image into the memory and replace cosine similarity score and face orientation error score with the values of current bitmap which is saved.

```
1.  initial values:
Cosine similarity score = 0
Face orientation error score = Integer.MAX_VALUE.
2.  if($css_{bitmap}$>$css_{image}$ &&
    $foe_{bitmap}$<$foe_{image}$ )
        i.    bitmap processing
        ii.   save image
        iii.  replace css and foe values
              with the saved image
```

Fig 8: Code for result computation.

Architecture of Proposed Methodology


User Selects Template
Check Template for face
Yes
Template Facial Points Extraction
Compute Vectors using a central point
User press Capture
No
Show error message and tell user to try again
Cosine similarity of template and sensor data vectors
Compute Sensor data vectors from a central point
Start Capturing Bitmaps for 15 seconds
Compare score with previous best. If better
Calculate score from orientation and vector similarity
Check face orientation similarity
Extract Sensor Data Facial Points
Yes
Send Bitmap for Processing and then Save Image
No

IV. RESULTS

When comparing results of cosine similarity and face orientation ratio errors, there appeared certain cases where the final result was not close to the template image. This happens because of certain cases like:-

- If user does not try to match the expressions of template image but the first photo with very low similarity score will be clicked because initially cosine similarity score is zero and any image with face will have better similarity score.
- We also introduced max_error value for face orientation i.e. image will not be saved if error value for face orientation for bitmap exceeds a certain error value. The threshold for error values were 0.5

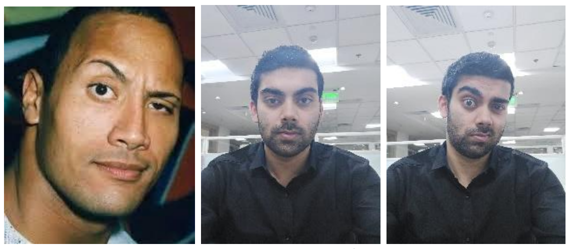


Fig 9: (A) Template image taken in which left eyebrow is raised. (B) First image captured after detecting face.
(C) Final image captured most accurate to the template image after trying different poses for 10-15 seconds..

A. *Sample UI*

The current implementation of this project is done on a sample app designed by us. Below is the screen shot of this sample app:-

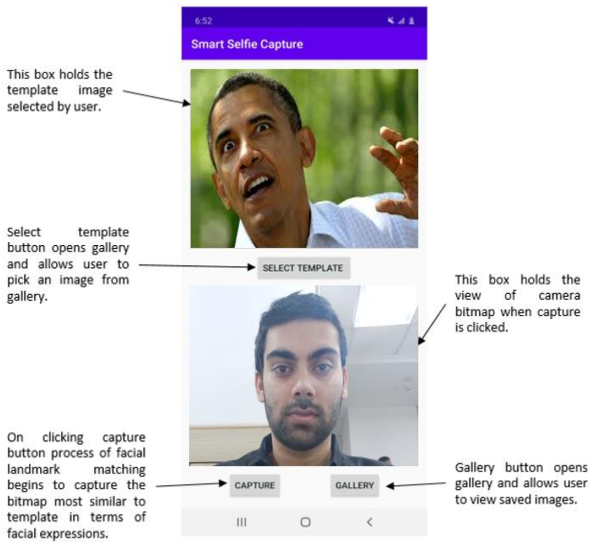

6:52
Smart Selfie Capture
This box holds the template image selected by user.
Select template button opens gallery and allows user to pick an image from gallery.
SELECT TEMPLATE
This box holds the view of camera bitmap when capture is clicked.
On clicking capture button process of facial landmark matching begins to capture the bitmap most similar to template in terms of facial expressions.
CAPTURE
GALLERY
Gallery button opens gallery and allows user to view saved images.

Fig 10: UI of sample app.

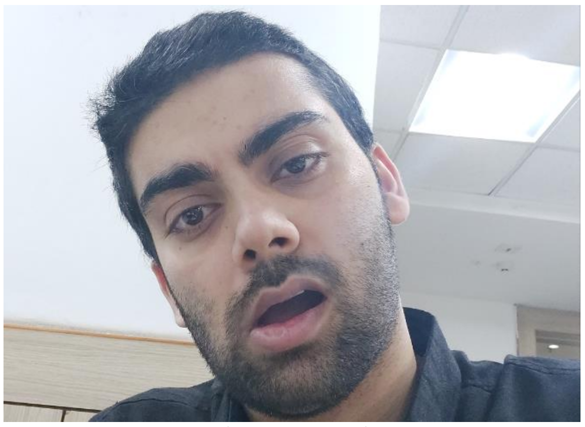


Fig 11: Final image captured for expression of template image in sample UI.

B. *Future Work*

The future work regarding this project is to incorporate smart -selfie as a separate mode for camera and include this mode in camera app.

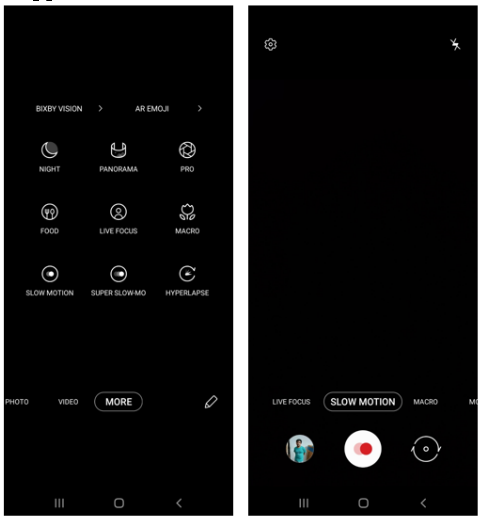


Fig 12: UI of camera app and how smart selfie can be incorporated as one of the modes of camera like slow motion, live focus, hyperlapses, etc.

V. CONCLUSION

We proposed an approach to use a facial feature recognition model to obtain the facial landmarks of a face selected by user to be called as template image. After extracting facial landmarks when user press capture button process begins running for 15 seconds in which bitmaps are send for processing where facial feature points of bitmap are extracted and compared to the facial feature points of template image using cosine similarity and facial orientation ratios are compared for error to now the deviation of captured bitmap from template image. These results are compared separately with the previous best image captured in the process (for 1st image we have used a minimum threshold value after comparing so that all corner cases are covered). If scores are better than previous stored image we send bitmap for processing and save that image and replace cosine similarity score and face orientation error score with scores of this image as these are the new best scores we have for the most similar pose and facial expressions as compared to the template image.

REFERENCES

[1] F. Dornaika, K. Wang, I. Arganda-Carreras, A. Elorza, and A. Moujahid, "Toward graph-based semi-supervised face beauty prediction," *Expert Syst. Appl.*, vol. 142, Mar. 2020, doi: 10.1016/j.eswa.2019.112990.

[2] K. Sun, H. Kang, and H. H. Park, "Tagging and classifying facial images in cloud environments based on KNN using MapReduce," *Optik (Stuttg).*, vol. 126, no. 21, pp. 3227–3233, Nov. 2015, doi: 10.1016/j.ijleo.2015.07.080.

[3] P. Wei, Z. Zhou, L. Li, and J. Jiang, "Research on face feature extraction based on K-mean algorithm," *Eurasip J. Image Video Process.*, vol. 2018, no. 1, pp. 1–9, Dec. 2018, doi: 10.1186/s13640-018-0313-7.

[4] B. Gökberk, M. O. Irfanoğlu, and L. Akarun, "3D shape-based face representation and feature extraction for face recognition," *Image Vis. Comput.*, vol. 24, no. 8, pp. 857–869, Aug. 2006, doi: 10.1016/j.imavis.2006.02.009.

[5] Y. M. Wu, H. W. Wang, Y. L. Lu, S. Yen, and Y. T. Hsiao, "Facial feature extraction and applications: A review," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7196 LNAI, no. PART 1, pp. 228–238, doi: 10.1007/978-3-642-28487-8_23.

[6] N. Wang, X. Gao, D. Tao, H. Yang, and X. Li, "Facial feature point detection: A comprehensive survey," *Neurocomputing*, vol. 275, pp. 50–65, Jan. 2018, doi: 10.1016/j.neucom.2017.05.013.

[7] E. Saber and A. M. Tekalp, "Frontal-view face detection and facial feature extraction using color, shape and symmetry based cost functions," *Pattern Recognit. Lett.*, vol. 19, no. 8, pp. 669–680, Jun. 1998, doi: 10.1016/S0167-8655(98)00044-0.

[8] D. E. King, "Dlib-ml: A Machine Learning Toolkit," 2009.

[9] S. Sharma, K. Shanmugasundaram, and S. K. Ramasamy, "FAREC - CNN based efficient face recognition technique using Dlib," in *Proceedings of 2016 International Conference on Advanced Communication Control and Computing Technologies, ICACCCT 2016*, Jan. 2017, pp. 192–195, doi: 10.1109/ICACCCT.2016.7831628.

[10] T. Kielmann *et al.*, "BLAS (Basic Linear Algebra Subprograms)," in *Encyclopedia of Parallel Computing*, Springer US, 2011, pp. 157–164.

[11] X. Sun, P. Wu, and S. C. H. Hoi, "Face detection using deep learning: An improved faster RCNN approach," *Neurocomputing*, vol. 299, pp. 42–50, Jul. 2018, doi: 10.1016/j.neucom.2018.03.030.

[12] Z. H. Feng, J. Kittler, M. Awais, P. Huber, and X. J. Wu, "Face Detection, Bounding Box Aggregation and Pose Estimation for Robust Facial Landmark Localisation in the Wild," in *IEEE Computer Society Conference on Computer Vision and Pattern*

Recognition Workshops, Aug. 2017, vol. 2017-July, pp. 2106–2111, doi: 10.1109/CVPRW.2017.262.

[13] V. P. Vishwakarma, “Deterministic learning machine for face recognition with multi-model feature extraction,” Mar. 2017, doi: 10.1109/IC3.2016.7880264.

[14] H. Otroshi-Shahreza, “Frame-based face emotion recognition using linear discriminant analysis,” in *Proceedings - 3rd Iranian Conference on Signal Processing and Intelligent Systems, ICSPIS 2017*, Mar. 2018, vol. 2017-December, pp. 141–146, doi: 10.1109/ICSPIS.2017.8311605.

[15] P. Forczmański and G. Kukharev, “Comparative analysis of simple facial features extractors,” *J. Real-Time Image Process.*, vol. 1, no. 4, pp. 239–255, Jul. 2007, doi: 10.1007/s11554-007-0030-4.

[16] P. Xia, L. Zhang, and F. Li, “Learning similarity with cosine similarity ensemble,” *Inf. Sci. (Ny).*, vol. 307, pp. 39–52, Jun. 2015, doi: 10.1016/j.ins.2015.02.024.

[17] “Digital image processing using face detection and skin tone information,” Dec. 2017.

[18] A. W. Young, K. H. McWeeny, D. C. Hay, and A. W. Ellis, “Matching familiar and unfamiliar faces on identity and expression,” *Psychol. Res.*, vol. 48, no. 2, pp. 63–68, Aug. 1986, doi: 10.1007/BF00309318.