Face Recognition System based on Deepface

Bin Huang 12012910 Qiang Hu 12111214 Qijia He 12111211

I Introduction

As the era advances, facial recognition is gradually becoming an increasingly popular field. Traditional facial recognition algorithms rely on a single statistical model, feature matching, and texture analysis, which pose certain problems and challenges. In recent years, with the widespread application of deep learning in the field of images, it has been discovered that utilizing deep learning algorithms often yields better performance in facial recognition. Prominent methods that have been proposed and widely acclaimed include VGG-Face and FaceNet. The modern process of facial recognition primarily consists of four stages: detection, alignment, representation, and verification. As is an open-source deep learning toolkit for facial recognition that integrates previous models. deepface provides a simple and user-friendly interface for facial recognition, detection, and identification of attributes such as emotions, gender, race, and age.

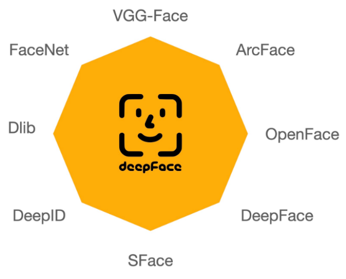


Fig. 1: framework: our project

In our project, we primarily rely on the DeepFace framework to build facial recognition and facial analysis systems (This can be worked in our GUI by running main.py). By leveraging online resources and our own curated datasets, we conduct experiments and comparisons to assess the detection and analysis performance of the models on different datasets (see /interesting/experimemt.ipynb).

For more details, you can directly visit our open-source project on GitHub. The specific link to the project Fae42/cv-final-project is as follows: https://github.com/Fae42/cv-final-project

II Related Works

DeepFace can be broadly divided into two parts: image recognition and image analysis. Its construction is heavily reliant on previous research in the field. Prior to the introduction of DeepFace, there was a significant amount of work done on face recognition and facial attribute analysis. Here we list several key contributions in this area:

- VGG-Face model: VGG-Faces is a deep convolutional neural network model developed by the Visual Geometry Group team at the University of Oxford in 2015. This model demonstrates high performance and accuracy in large-scale face recognition tasks.
  https://github.com/rcmalli/keras-vggface
- FaceNet: FaceNet is a model developed by Google in 2015, based on deep learning and neural network technologies. It introduces a triplet loss function to train neural network models. This further reduces intra-class variations and inter-class differences in face recognition, enhancing the discriminative capability of the face recognition model.
  https://github.com/davidsandberg/facenet.
- OpenFace: OpenFace implements automatic facial behavior analysis. As the first toolkit capable of facial landmark detection, head pose estimation, facial action unit recognition, and eye-gaze estimation with available source code for both running and training the models. OpenFace provides a valuable reference for the development of deepface affective attribute analysis.
  https://github.com/cmusatyalab/openface
- Dlib: Originally proposed in 2002 and continuously improved thereafter, the Dlib library is capable of achieving precise face detection and keypoint localization. By accurately detecting the position of faces and their key points, Dlib provides a foundation for subsequent facial attribute analysis.
  https://github.com/davisking/dlib
- Other related works: DeepID, DeepFace, LightFace...

III Framework

A. Framework of deepface

Realised in 2019, DeepFace is built on top of multiple models, integrating various state-of-the-art models, like VGG-face, Arcface, and so on. It has continuously evolved and improved over time. Here are the model frameworks that DeepFace offers on GitHub, which is an open-source platform: [[serengil/deepface: A Lightweight Face Recognition and Facial Attribute Analysis (Age, Gender, Emotion and Race) Library for Python (github.com)](https://github.com/serengil/deepface)]

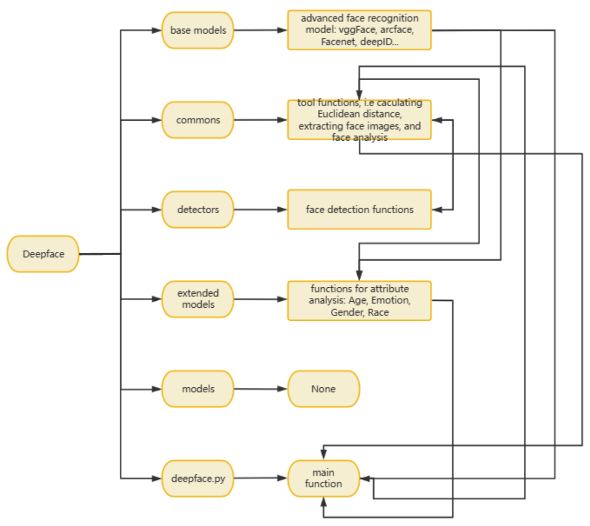

Deepface
base models
advanced face recognition model: vggFace, arcface, Facenet, deepID...
commons
tool functions, i.e caculating Euclidean distance, extracting face images, and face analysis
detectors
face detection functions
extended models
functions for attribute analysis: Age, Emotion, Gender, Race
models
None
deepface.py
main function

Fig. 2: framework: deepface

Seen from Fig.2, now we give a clear insight of the structure of DeepFace.

Within the base models, for each model like FaceNet, ArcFace, etc., pre-trained model parameters are directly downloaded from GitHub for facial detection. The **extended models** employ CNN classification and regression methods to predict facial age, race, emotions, and gender. The weights and parameters of each convolutional neural network are derived from pre-trained models available on GitHub. The **commons** folder primarily includes essential utility functions, such as calculating the Euclidean distance between vectors, cropping images after face recognition, and facial attribute analysis (which calls the main function of DeepFace). The **detector** folder implements positional face detection, supporting the implementation of functions.extract_faces in commons. Finally, **DeepFace.py** encapsulates all of these methods, enabling efficient, fast, and simple implementation of facial detection and related tasks.

DeepFace provides several commonly used functions, including **DeepFace.find()**: for finding the best matching image in a database for an input image, **DeepFace.verify()**: for verifying if two images belong to the same person. **DeepFace.stream()**: for detecting faces in a video stream while analysis the face's attributes. **DeepFace.detection()**: for locating faces in an image, and **DeepFace.analysis()**: for analyzing facial attributes, predicting the target face's age, gender, emotions, and race using regression and classification methods. In the next section, we will provide a detailed explanation of these functions.

B. *Framework of our project*

In our project, we mainly divided it into three folders: **GUI, testprecision, and interesting**. Within the GUI folder, there are five Python code files that are connected to the **main.py** file to implement the basic facial recognition system. The **test_precision** folder is dedicated to testing the accuracy of facial recognition on three different datasets. In the **interesting** folder, we built our own dataset called **ourfaces** and performed facial recognition within the database using the **ourfacedetection.py** file. We conducted model testing on our dataset and performed a simple analysis of the test results in the **experiment.ipynb** file. The results of the analysis are stored in the analysis folder. Fig.3 gives a clear outlook of the framework of our project

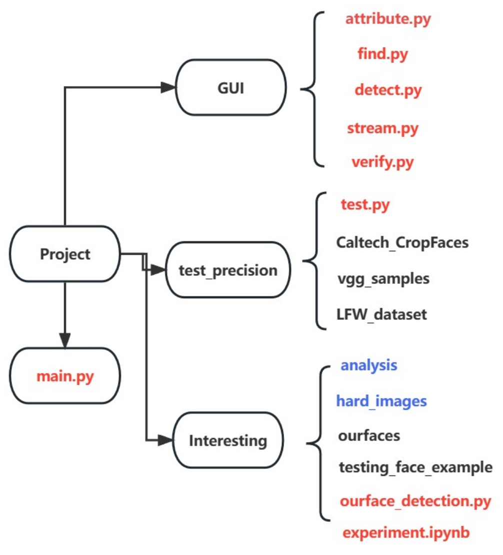


Fig. 3: framework: our project

IV Useful Functions (Method)

A. *Image Face Detection*

DeepFace.extract_faces()

- Input:
  - img_path: the path of one image that need to be detected faces from.
  - other parameters:target_size, detector_backend, grayscale, enforce_detection, align
    Note: the enforce_detection (T/F), if set to T, we will perform Rigorous face detection, otherwise the detection wil be rough, which means we might have some non-face objects regarded as faces.
- Output
  - a list of dictionaries, in each of which is a detected face including subimage of face, parameters foe bonding box and confidence.
- Realizations: This model relies on keypoint detection for face recognition, and keypoint detection depends on OpenCV. Deepface uses OpenCV's **cv2.dnn.readNetFromCaffe()** to load the network structure of a pre-trained Caffe model and build a DNN

network. It then utilizes **cv.dnn.blobFromImage()** to infer the Binary Large Object of the input image, aiding in keypoint detection.

Once keypoints (eyes) are detected, further face detection is performed through image alignment. The specific alignment method is as follows:

- 3 steps to align the face
  - Find the position of eyes for each face using algorithm in OpenCV.

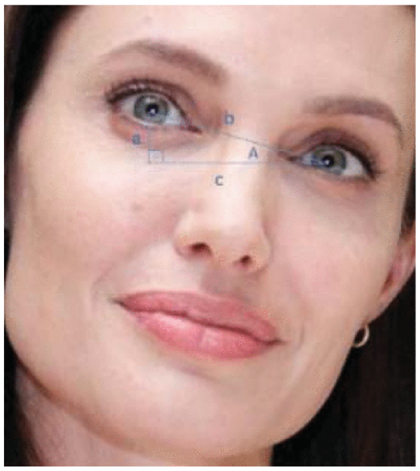


Fig. 4: Example: Image alignment

  - According to Fig.4, calculate the angle between the hypotenuse and the horizontal side by the following formula.

$$cosA = \frac{b^2 + c^2 - a^2}{2bc}$$

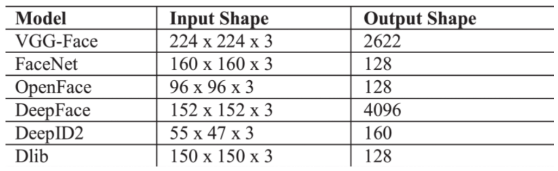
  - Rotate the face according to angle A. For Fig.4, we rotate the image counterclockwise.

B. Image feature extraction

DeepFace.represent()

- Input:
  - image_path: the path to the image
  - other parameters: model_name, enforce_detection, detector_backend, align, normalization
- Output:
  - a zip that contains the feature vector of the aligned image, bounding boxes of faces and its corresponding confidence score
- Realizations

  After we aligned the image(as shown in section A), we start to extract feature vectors. The process of extracting feature vectors mainly relies on different algorithmic models proposed by previous researchers. The core of these models involves using convolutional neural networks for feature extraction. However, different models have distinct network structures and feature vector dimensions. The specific parameters are shown in the table below. Once a model is chosen, the pre-trained model available on GitHub can be utilized to directly extract the corresponding feature attributes.



Model	Input Shape	Output Shape
VGG-Face	224 x 224 x 3	2622
FaceNet	160 x 160 x 3	128
OpenFace	96 x 96 x 3	128
DeepFace	152 x 152 x 3	4096
DeepID2	55 x 47 x 3	160
Dlib	150 x 150 x 3	128

Fig. 5: alignment

C. Face Verification

DeepFace.verify()

- Input:
  - img1_path: the path of one image that need to be tested
  - img2_path: the path of the other image that need to be tested
  - other parameters: model name, distance metric, align, normalization, enforce_detection, detector_-backend, silent
- Output:
  - a zip that contains verify result, distance, threshold, selected model and some other parameters that might be useful
- Realizations:

  Calculate the distance between the feature vectors extracted from two images. the process of feature extraction calls the function **DeepFace.represent()**. The larger the distance between the feature vectors, the greater the difference between the images. Use the decision tree algorithm to determine the optimal threshold value. When the threshold is higher than true value, we consider the two images to be the same image. Here we show two typical ways to calculate distance:
- 2 ways to calculate distance
  - euclidean distance:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i = y_i)^2}$$

  - cosine distance:

$$D_c(x,y) = 1 - cos\theta = 1 - \frac{\sum_{i=0}^{n} x_i y_i}{\sqrt{\sum_{i=0}^{n} x_i^2}\sqrt{\sum_{i=0}^{n} y_i^2}}$$

D. Face Matching

DeepFace.find()

- Input:
  - img_path: the path of the image that need to be tested
  - db_path: the path to the dataset
  - other parameters: model_name, distance_metric, align, normalization, enforce_detection, detector_-blacked, silent
- Output:

  - a list of images that have person which is similar to the input image. In each image, we have: image's path, bounding boxes of faces, and confidence score
- Realizations

  Similar to DeepFace.verify(), we calculate the similarity scores between the input image and each image in the database, sort them, and output the top few images with the highest similarity.

E. *Stream Face Recognition*

DeepFace.stream()

- Input:
  - db_path:facial database path used for verification
  - source:the path of exact video, access web camera if set this to zero.
  - other parameters: model_name, detecter_backend, distance_metric, enable_facial_analysis, time_threshold, frame_threshold.
- Output:
  - a window to show the results of recognition and analysis of every several(depends on frame_threshold) frames.
- Realizations:

  For each frame that to be detected, confirm if there's a face in it by extract_faces(). If so, use find() and analyze() to confirm its identity and analyze other information.

F. *Feature Extraction*

DeepFace.analyze()

- Input:
  - img_path: exact image path, numpy array (BGR) or base64 encoded image could be passed. If source image has more than one face, then result will be size of number of faces appearing in the image.
  - actions (tuple): The default is ('age', 'gender', 'emotion', 'race'). Can be used to dropsome of those attributes.
  - enforce_detection (bool): The function throws exception if no face detected by default.
  - detector_backend (string): set face detector backend to opencv, retinaface, mtcnn, ssd, dlib or mediapipe.
  - silent (boolean): disable (some) log messages
- Output:
  - a list of dictionaries for each face appearing in the image.
- Realizations:

  Build the required models by $actions$. The age, gender and race prediction model were built on the base VGG-Face model.

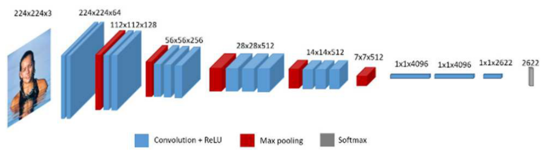
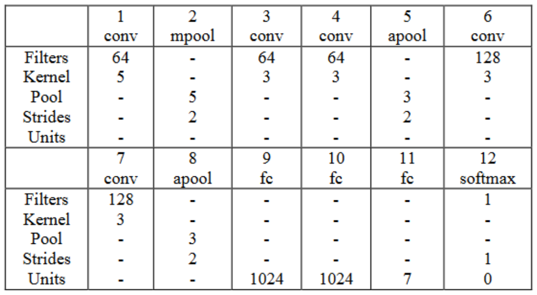


Fig. 6: VGG-Face structure

The structure of emotion model is shown in Fig. 7.



	1 conv	2 mpool	3 conv	4 conv	5 apool	6 conv
Filters	64	-	64	64	-	128
Kernel	5	-	3	3	-	3
Pool	-	5	-	-	3	-
Strides	-	2	-	-	2	-
Units	-	-	-	-	-	-
	7 conv	8 apool	9 fc	10 fc	11 fc	12 softmax
Filters	128	-	-	-	-	1
Kernel	3	-	-	-	-	-
Pool	-	3	-	-	-	-
Strides	-	2	-	-	-	1
Units	-	-	1024	1024	7	0

Fig. 7: emotion model structure

Calls extract_faces() function to detect and cut faces using a face detector(OpenCV by default). Using each model to predict corresponding attribute. These problems are defined as classification tasks. It should be noted that each output score of age model will be multiplied by the corresponding age. In this way, we can predict the apparent age.

V Experiment

A. *Performance Evaluation: A Literature review*

The development team of Deepface has conducted performance analysis during the research process of Deepface. Here, let's first review a series of results achieved by the Deepface team in performance analysis.

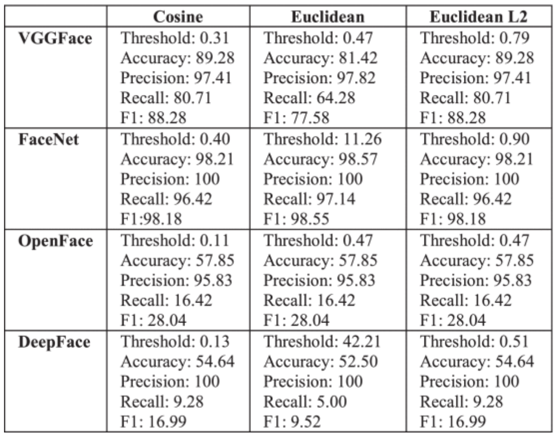
For most machine learning models, the evaluation of their performance relies primarily on metrics such as precision, recall, and the confusion matrix. The specific definitions of precision rate and recall rate are as follows:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

In their previous work, they primarily used recall rate, precision rate, confusion matrix, and distribution graphics as measurement evaluation tables. By testing these metrics on different models using their dataset, we can clearly observe the performance of Deepface in practical applications. For the rest of this section, we will introduce their test results one by one.

As mentioned in IV about DeepFace.verify(), decision tree algorithms are used to find the best threshold, the corresponding optimal performance for each model is shown below



	Cosine	Euclidean	Euclidean L2
VGGFace	Threshold: 0.31 Accuracy: 89.28 Precision: 97.41 Recall: 80.71 F1: 88.28	Threshold: 0.47 Accuracy: 81.42 Precision: 97.82 Recall: 64.28 F1: 77.58	Threshold: 0.79 Accuracy: 89.28 Precision: 97.41 Recall: 80.71 F1: 88.28
FaceNet	Threshold: 0.40 Accuracy: 98.21 Precision: 100 Recall: 96.42 F1:98.18	Threshold: 11.26 Accuracy: 98.57 Precision: 100 Recall: 97.14 F1: 98.55	Threshold: 0.90 Accuracy: 98.21 Precision: 100 Recall: 96.42 F1: 98.18
OpenFace	Threshold: 0.11 Accuracy: 57.85 Precision: 95.83 Recall: 16.42 F1: 28.04	Threshold: 0.47 Accuracy: 57.85 Precision: 95.83 Recall: 16.42 F1: 28.04	Threshold: 0.47 Accuracy: 57.85 Precision: 95.83 Recall: 16.42 F1: 28.04
DeepFace	Threshold: 0.13 Accuracy: 54.64 Precision: 100 Recall: 9.28 F1: 16.99	Threshold: 42.21 Accuracy: 52.50 Precision: 100 Recall: 5.00 F1: 9.52	Threshold: 0.51 Accuracy: 54.64 Precision: 100 Recall: 9.28 F1: 16.99

Fig. 8: performance

According to the outcomes, the FaceNet model is of the highest precision and recall among all these model, and the best case is **100 precision and 97.14 recall** under Euclidean metric.

This distribution graphic is estimated by kernel density estimation(KDE). It shows the distribution of each single model metric pair for yes and no classes and the robustness of each model as well.

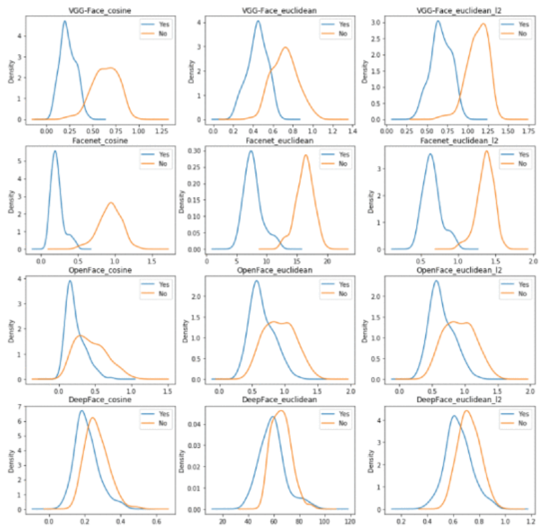
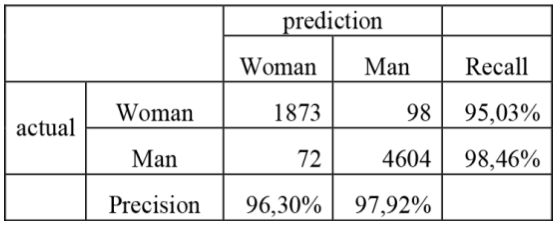


Fig. 9: distribution graphic

It can also be observed from the distribution graphic that the FaceNet model has the best robustness, since the two peaks are remarkably separated.

The confusion matrix for face analysis model shows the details of accuracy and error on each classes for the model. The confusion matrix for gender model(including 2 classes, man and woman) has been calculated as followed.

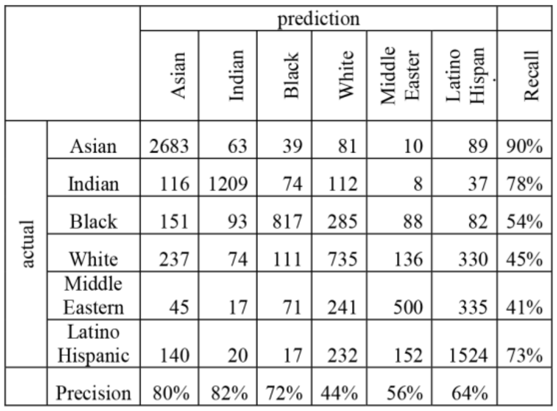


		prediction		
		Woman	Man	Recall
actual	Woman	1873	98	95,03%
	Man	72	4604	98,46%
	Precision	96,30%	97,92%	

Fig. 10: confusion matrix for gender model

Gender prediction model has **97.44% accuracy** on the ICCV'15 test set and relatively high precision and recall, which is shown in the table.

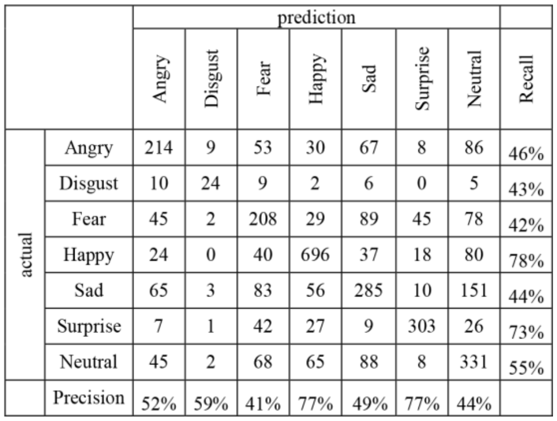
The confusion matrix for race/ethnicity model(6 classes including Asian, Indian, Black, White, Middle Eastern and Latino Hispanic) and emotion model(7 classes including Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral) has been also calculated as followed.



		prediction						
		Asian	Indian	Black	White	Middle Easter	Latino Hispan	Recall
actual	Asian	2683	63	39	81	10	89	90%
	Indian	116	1209	74	112	8	37	78%
	Black	151	93	817	285	88	82	54%
	White	237	74	111	735	136	330	45%
	Middle Eastern	45	17	71	241	500	335	41%
	Latino Hispanic	140	20	17	232	152	1524	73%
	Precision	80%	82%	72%	44%	56%	64%	

Fig. 11: confusion matrix for race/ethnicity model

The accuracy of race/ethnicity model has **68% accuracy**. The precision and recall for White and Middle Eastern is relatively low, while others have relatively high precision and recall.



		prediction							
		Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral	Recall
actual	Angry	214	9	53	30	67	8	86	46%
	Disgust	10	24	9	2	6	0	5	43%
	Fear	45	2	208	29	89	45	78	42%
	Happy	24	0	40	696	37	18	80	78%
	Sad	65	3	83	56	285	10	151	44%
	Surprise	7	1	42	27	9	303	26	73%
	Neutral	45	2	68	65	88	8	331	55%
	Precision	52%	59%	41%	77%	49%	77%	44%	

Fig. 12: confusion matrix for emotion model

The accuracy of emotion model is **57.42%**. The precision and recall of Happy and Surprise is the highest, while the performance of others are almost the same.

B. Experiment: Evaluations based on our data

After the literature review, we try to briefly test this model in different datasets, and even build our own dataset. The result of which are as follows. We first selected three datasets to reproduce the performance of Deepface in face detection. The three datasets include the positive face samples from Assignment 4, the sample data from VGG-Face, and the LFW dataset. By calculating the precision rate on each dataset, we obtained the following results:

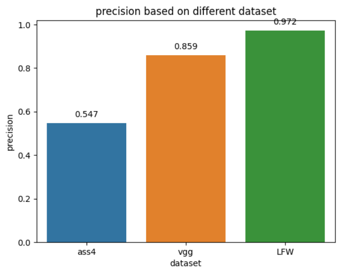


Fig. 13: precision in different datasets

It can be observed that Deepface achieves a higher accuracy rate on the LFW dataset, exceeding 97%, while it performs relatively poorly on the Assignment 4 dataset. **This is attributed to the low resolution of the data, where the face size is only 36x36 pixels.**

Then we started building our own dataset. Each team member contributed more than 20 personal face photos, and we added 12 noisy face images, resulting in a small face dataset of 71 photos (available at github/interesting/our_faces). On this dataset we performed further operations and processing on the model. The specific code implementation can be found in the **experiment.ipynb** file. First, we calculate the **precision rate** of the results, following the same procedure as mentioned above: we input each image into Deepface.extract_-faces(), and if no errors occur, we count the number of errors to calculate the specific value of precision. In our dataset, **the precision rate is 73%.** Next, we find out all the unrecognized faces, as shown in the image below.

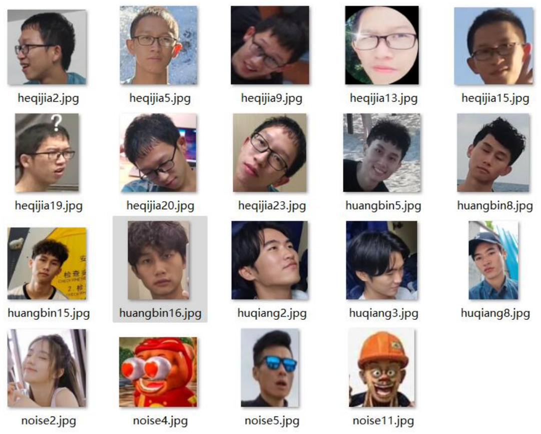
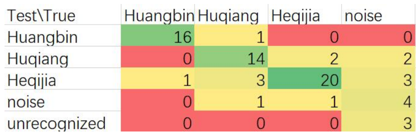


Fig. 14: unrecognized faces

From the image, we observe that the majority of unrecognized faces have noticeable head tilts, while a small portion only partially exposes the face. Hats (nosie11, huqiang8), sunglasses (nosie5), and lighting conditions (heqijia5, heqijia15) introduce certain interference to face recognition. Additionally, we noticed that a significant number of unrecognized faces have their eyes closed. **This can be attributed to Deepface relying on eyes as key points for alignment and detection.**

After completing the basic precision detection, we further generated a **confusion matrix** for the detection results. The specific form of the matrix is shown below. The additional dimension on the row of the 5x4 matrix represents the images where the faces were not recognized. From the image, we can conclude that the recall rate and precision rate for each face detection are approximately between 73% and 95%. The correct recognition rate for the noisy images is relatively low, with a higher occurrence of misclassifications. (Note: There are only three images that were not detected here, unlike before (detecting rate = 73%). This is because enforce_detection=False was set here to increase the probability of correct identification).



Test\True	Huangbin	Huqiang	Heqijia	noise
Huangbin	16	1	0	0
Huqiang	0	14	2	2
Heqijia	1	3	20	3
noise	0	1	1	4
unrecognized	0	0	0	3

Fig. 15: unrecognized faces

Meanwhile, we compared the confusion matrices of six models (VGG, FaceNet, DeepID2, Dlib, OpenFace, DeepFace) but found that their confusion matrices were completely identical (see BigMatrix in the code). **We attributed this to the limited amount of face data used in the dataset.**

Now we move a step forward to attribute analysis.

First, we iterate through all the images and apply Deepface.analysis() to each image. We store all the extracted features in a list. Then, we extract the facial attributes from the list, sum up the corresponding elements, and output the normalized values using the **Min-Max Scaling method** (with gender normalized by dividing by the sum of all values, while age is not normalized). Below are the results of the facial attributes of our group member's faces

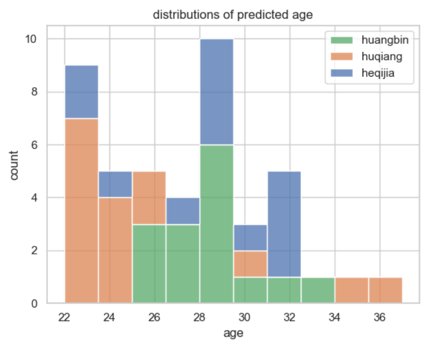
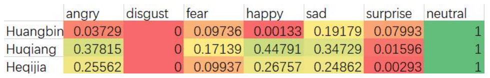
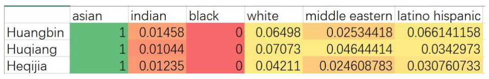


Fig. 16: distributions of predicted age



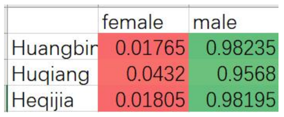
	angry	disgust	fear	happy	sad	surprise	neutral
Huangbin	0.03729	0	0.09736	0.00133	0.19179	0.07993	1
Huqiang	0.37815	0	0.17139	0.44791	0.34729	0.01596	1
Heqijia	0.25562	0	0.09937	0.26757	0.24862	0.00293	1

Fig. 17: emotion



	asian	indian	black	white	middle eastern	latino hispanic
Huangbin	1	0.01458	0	0.06498	0.02534418	0.066141158
Huqiang	1	0.01044	0	0.07073	0.04644414	0.0342973
Heqijia	1	0.01235	0	0.04211	0.024608783	0.030760733

Fig. 18: race



	female	male
Huangbin	0.01765	0.98235
Huqiang	0.0432	0.9568
Heqijia	0.01805	0.98195

Fig. 19: gender

From the chart above, it can be observed that DeepFace.analysis() has high accuracy in sentiment analysis, racial analysis, and gender analysis, with low variance in the data. This indicates that the distances between different categories of faces are relatively large, resulting in a lower probability of misclassification. However, in age recognition, it seems that each person in our group appears to be 5 to 10 years older. Additionally, there is a high variance in the data. This is likely due to the fact that facial changes are relatively small after adulthood. In terms of age prediction, there is still significant room for improvement in the current attribute analysis method.

Overall, whether it is the experimental data provided by the DeepFace team or our own experimental data, we have achieved relatively good results in terms of the test outcomes. However, our dataset is relatively small and limited in diversity, which prevents us from providing comprehensive and accurate model predictions. If we can further expand the dataset by including data from different races, genders, and age groups, we believe that the results will be more convincing

VI Graphic User Interface(GUI)

The GUI is divided into six parts, including main page, face attribute, video face detector, image face detector, face matching and face verification. All these six parts are built on the $PyQt5$ framework.

The main page contains five buttons.

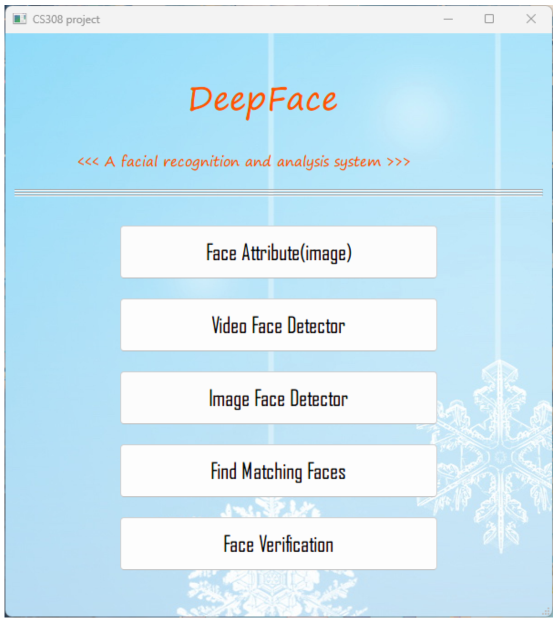


Fig. 20: Main page

Each button is connected to a corresponding function. When a button is clicked, the main page will awake a new window and close itself. We take face attribute as an example.

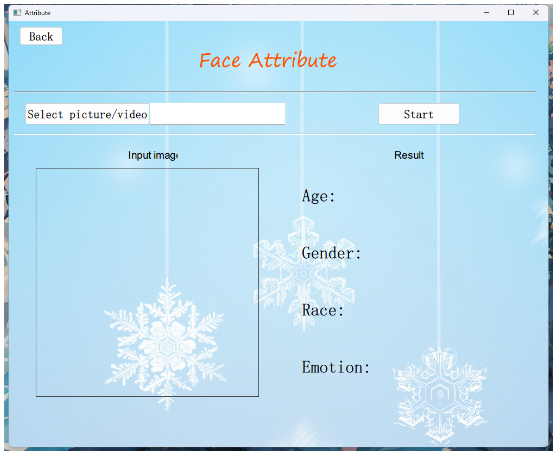
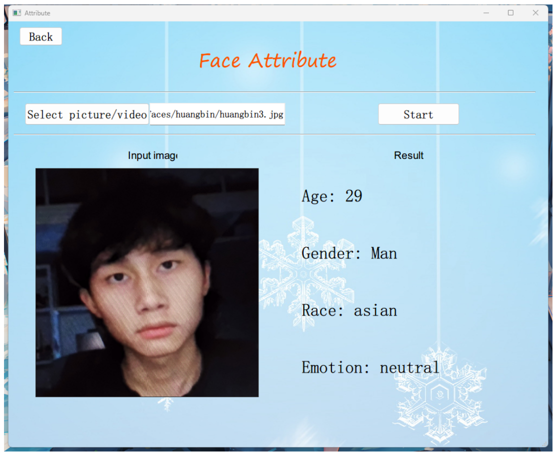


Fig. 21: Face attribute function

Selected image or video will be shown in the preview box. They will be send to DeepFace.analyze() after clicking the Start button. Once it finish working, the result will update in the GUI window.



Age: 29

Gender: Man

Race: asian

Emotion: neutral

Fig. 22: Face attribute result

The back button on the upper left can be used to go back to main page. To be specific, the face attribute window will awake main page window and close itself after the button is clicked. Then user can enter other functions without re-running the .py file.

VII Conclusion

A. Advantages:

Our facial detection system, based on the deepface framework model, performs well in most cases. As long as the face is properly aligned, it can detect and identify individuals accurately. After setting up the model, facial recognition typically takes only one second, and feature analysis is completed quickly. It is highly practical. Additionally, the model parameters are pre-trained online, eliminating the need for local training. Implementing face detection requires just a few lines of code, making it simple and easy to use.

B. Disadvantages:

Although the model exhibits high precision rates, we observed a significant decline in facial recognition performance when encountering various pose variations, as shown in Fig.14. Additionally, the deepface.analysis() results indicate relatively poor performance in emotion and race prediction, and our findings from Fig.16 demonstrate subpar performance in age prediction, with the model only able to predict rough age ranges. These areas require improvement and further development.

C. Future work:

Our team suggests two potential improvements to enhance facial prediction accuracy:

- Incorporating additional keypoint detection: As depicted in Fig.14, numerous faces with closed eyes were not successfully detected, primarily because the deepface.face_-extraction relies on eyes as keypoints for detection and alignment. Occlusion caused by closed eyes or sunglasses makes it challenging to detect faces. By incorporating more keypoints such as the nose, mouth, and ears, the detection rate is expected to improve.
- Conducting category-based training on the test data: Divide the head images into several categories, such as tilted head, wearing a hat, or wearing sunglasses, and train the model accordingly. This approach will make the model more sensitive to unique facial situations.

References

[1] S. I. Serengil and A. Ozpinar, "LightFace: A Hybrid Deep Face Recognition Framework," 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), Istanbul, Turkey, 2020, pp. 1-5, doi: 10.1109/ASYU50717.2020.9259802.

[2] S. I. Serengil and A. Ozpinar, "HyperExtended LightFace: A Facial Attribute Analysis Framework," 2021 International Conference on Engineering and Emerging Technologies (ICEET), Istanbul, Turkey, 2021, pp. 1-4, doi: 10.1109/ICEET53442.2021.9659697.

[3] Serengil, S., & Ozpinar, A. (2023). An Evaluation of SQL and NoSQL Databases for Facial Recognition Pipelines. Cambridge Open Engage. doi:10.33774/coe-2023-18rcn This content is a preprint and has not been peer-reviewed.

[4] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.

VIII Contributions

- Bin Huang (12012910):
  - code: main.py & attribute.py, GUI modification and integration
  - report: Method F & GUI & Conclusion & readme.md (github)
  - presentation: Method
- Qiang Hu (12111214):
  - code: stream.py & detect.py
  - report: Introduction & Method A, E & Experiment A
  - presentation: Experiment
- Qijia He (12111211):
  - code: verify.py & find.py & ourface_detection.py & experiment.ipynb
  - report: Related Work & Framework & Method B, C, D & Experiment B, report modification and integration
  - presentation: Introduction, framework and conclusion
- Dataset: All the team members