ADVANCED VISUAL RECOGNITION

MINI PROJECT-1

Team Name

RAVEN CLAW

Team Members

Name	Roll Number
Prasad Magdum	MT2022078
Rittik Panda	MT2022090
Prithviraj Purushottam Naik	MT2022083

CONTENTS

Topics	Page Number
1. Task 1: Face Verification	03
a.Introduction	03
b.Face Verification Using PCA	03
c.Face Verification Using AutoEncoder	07
d.Face verification Using DNN & Pretrained Facenet	13
e.Comparison Of All The Approaches	18
f. Conclusion	18
2. Task 2: Face Jewellery	19
a.Introduction	19
b.Steps Performed	19
c.Haar feature-based cascade classifiers	20
d.Challenges	21
e.Observations & Discussion	21
f.Results	22

Task1: Face Verification

Introduction: -

Face verification is a fundamental problem in computer vision and biometrics. It involves confirming whether two facial images belong to the same individual or not. The primary objective of face verification is to determine if a given face matches a reference face, often stored in a database. Face verification algorithms typically analyze facial features, such as landmarks, textures, and patterns, to make a decision regarding the similarity or dissimilarity between the two faces. Some of the popular use cases of face verification systems are tagging people on social media platforms, face lock on smartphones, attendance tracking etc.

1.Face verification using PCA

In this section we have implemented a PCA based approach for face verification.

Methodology: -

Preprocessing: -

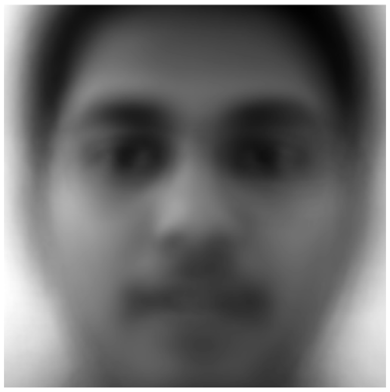
- **Face Detection: -** We first took the raw facial images and then applied Haar feature-based cascade classifier face detection algorithm to extract the face from given images. We used face-detected images, because
    1. It removed a significant portion of non-face background and noise from the input data.
    2. It improved feature extraction because facial landmarks (such as eyes, nose, and mouth) are often positioned consistently in face-detected images.
- **Converting into Grayscale: -** we converted the detected faces into grayscale. We converted the images into grayscale because -
    1. It reduces the dimensionality of the data.
    2. Grayscale images are less sensitive to variations in lighting conditions compared to color images.
    3. Grayscale images tend to have less noise.
- **Resizing: -** We resize all images into the same size to represent them in the same feature space.
- **Vectorization: -** we converted all images into 1-D vectors so that we can represent them into D dimensional space.

Principal Component Analysis (PCA) : -

- It is a dimensionality reduction technique.
- It projects the features into directions that have high variance. Higher variance implies more information.
- It reduces the number of features in a dataset while preserving as much of the original information as possible.
- So here in our task we will be using PCA to project the faces into lower dimensional feature space. And using this lower dimensional representation we can calculate the similarity between two faces.

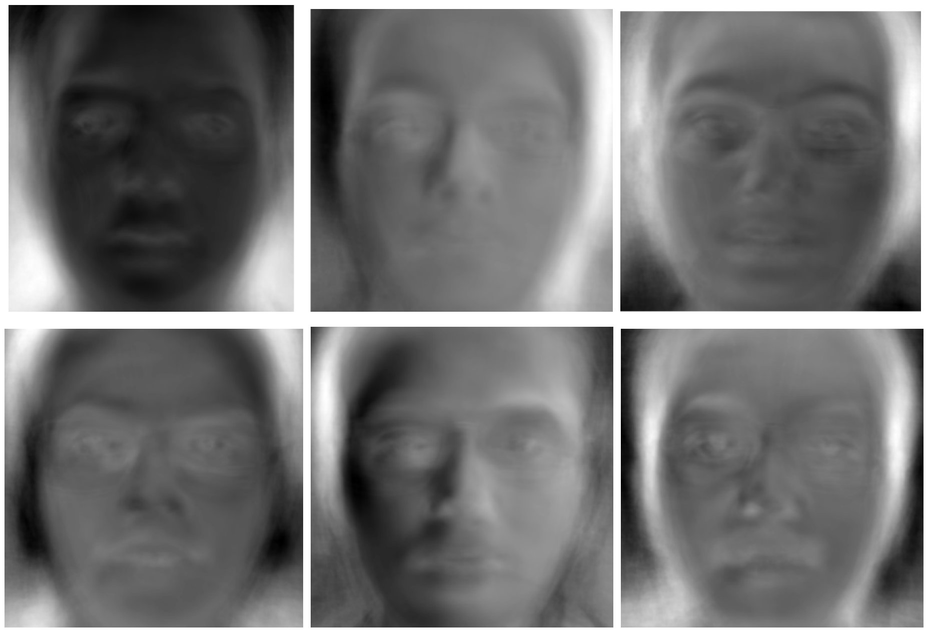
Steps Performed: -

- **Calculating Mean Face: -** First we calculated the mean face from the training dataset. Using the vectorized images we calculated the mean of all vectors.



- **Shifted Faces: -** Then we subtracted all faces from mean faces. We performed this step to center all faces to origin. This step eliminates any variations in lighting conditions, pose, or facial expression.
- **Covariance Matrix: -** we calculated the covariance matrix of shifted faces. It describes the relationships between different facial images.
- **Eigen Faces: -** Then calculated the eigen values and vectors of the transposed covariance matrix. Eigenvectors represent the directions of maximum variance, and eigenvalues indicate how much variance is explained along those directions.
- **Choose Top K Eigen Faces: -** We will choose the K eigen vectors corresponding to top K eigen values. Top eigen vectors allow us to identify the principal components that capture the most variance in the data.

- **Top 6 EIgen Faces: -**



- **Normalizing the Eigen Faces: -** After choosing top K Eigen faces we normalized them. Because without normalization, the magnitude of the eigenvectors would affect their importance in explaining the variance, which can lead to misleading interpretations.
- **Projecting the faces onto Eigenspace: -** By using the normalized eigen faces we represented all faces in eigen space.
- **Calculating the threshold: -** Taking half of the maximum pairwise distance between the projected images.
- **Testing Phase: -** Shifted the test images using the mean image of the training dataset. And then projected the shifted test images onto the eigen space.
- **Face verification for test images: -** for each test image we calculated the distance between that projected image and all projected train images. The one which has distance less than threshold is returned from train images.

Challenges: -

Calculating the covariance matrix: -

- When we were having shifted faces in matrix A and while calculating the covariance matrix with AA^T we were getting a large dimensional matrix. And we were getting the memory limit exceeded error.
- To solve this MLE problem we calculated eigen vectors for the A^TA matrix. Because A^TA will be a small dimensional matrix.
- By solving equations we got to know that AA^T and A^TA have the same eigen values and related eigen vectors.

- So we got the eigen vectors of AA^T by multiplying the eigen vectors of A^TA with matrix A.

Choosing the Code size: -

- It is a crucial parameter as it determines the dimensionality of the feature space in which facial images are encoded.
- We started with a larger code size but it was not giving good results. Because projecting the facial images into larger dimensional space may contain a lot of noise.
- Instead of keeping it large we reduced it to a smaller value. And we got good results with it because smaller code sizes can capture only the most essential features.
- Code size of 100 gave us a good result.

Observations: -

Without face detected Images: -

- First we applied PCA without detecting the faces. But the result was not that good.
- Because applying PCA on raw facial images will contain some kind of noise.

With face detected Images: -

- When we applied PCA on face detected images we got good results.
- Because face detection removed some part of the non-face background and also it reduced noise from data.
- It improved feature extraction so that models can focus on key features for face verification tasks.

2.Face Verification Using AutoEncoder

In this section we have implemented an autoencoder based approach for face verification.

- **Autoencoders: -** Autoencoders are neural networks used for dimensionality reduction and feature learning.
- We used autoencoders to project the facial images into lower dimensional space while preserving the essential information.
- Autoencoder consist of two main components: an encoder and a decoder.
- **Encoder: -** The encoder extracts the essential features from input facial images.
- **Decoder: -** The decoder is responsible for reconstructing facial images from the representation generated by the encoder.
- Autoencoder projects the images to a lower dimension and then tries to reconstruct it back. while reconstructing the images we will get the representation of images into lower dimension while preserving the essential information.
- We use these representations to compare and verify the faces of individuals.

Architecture

Encoder: -

The encoder architecture consists of several convolutional layers followed by max-pooling and batch normalization layers. Here we are using convolution layers to extract the essential features from facial images.

- Input layer with the specified image size.
- A stack of convolutional layers with increasing filter counts (32, 64, 128, 256, 512, and 1024), each using a 3x3 kernel and 'relu' activation.
- Max-pooling layers with a 2x2 pool size and strides of 2x2 to downsample the feature maps.
- Batch normalization layers to stabilize training.
- A Flatten layer to convert the output to a vector.
- A Dropout layer with a 50% dropout rate to prevent overfitting.
- A fully connected Dense layer to generate the code of size code_size.

Decoder: -
It comprises fully connected and transposed convolutional layers in reverse order of the encoder:

- Input layer with the code size as input.
- A fully connected Dense layer to expand the code into a suitable shape (4x4x1024).
- Reshape layer to transform the output into a 4D tensor.
- A series of transposed convolutional layers (Conv2DTranspose) with decreasing filter counts (512, 256, 128, 64, 32, and 1) to gradually upsample the features.
- Each convolutional layer uses a 3x3 kernel and 'relu' activation.
- The final layer generates the reconstructed image.

We trained the autoencoder with MSE loss.

Image Preprocessing: -

- **Face Detection: -** We first took the raw facial images and then applied Haar feature-based cascade classifier face detection algorithm to extract the face from given images. We used face-detected images, because
    3. It removed a significant portion of non-face background and noise from the input data.
    4. It improved feature extraction because facial landmarks (such as eyes, nose, and mouth) are often positioned consistently in face-detected images.
- **Converting into Grayscale: -** we converted the detected faces into grayscale. We converted the images into grayscale because -
    4. It reduces the dimensionality of the data.
    5. Grayscale images are less sensitive to variations in lighting conditions compared to color images.
    6. Grayscale images tend to have less noise.
- **Resizing: -** we used CNN to encode the images. And we know that it takes a fixed size of images as input so we resized all input images to the same size.
- **Data augmentation & Normalization: -** we used keras “ImageDataGenerator” class to perform

Challenges: -

1. **Choosing the Architecture: -**

- **Autoencoder with Pretrained CNN: -**
    I. We tried using pre-trained CNN like VGG16 to extract the features and project the images into lower dimensions.

II. And then we constructed the decoder architecture that takes the input from pretrained CNN and tries to reconstruct the images back.
III. We added a linear layer after removing the last two layers of VGG16.
IV. And then we trained the model by freezing all the parameters of pre-trained VGG16 except the last layer that we added.
V. But it did not give us good results because our dataset is small. Model was overfitting because of the small training dataset. And also we can say that VGG16 is trained on diverse datasets that may differ significantly from our face verification dataset in terms of lighting conditions, pose, background, and facial expressions.

- **Autoencoder with our own CNN: -**
  I. After trying the pre-trained CNN for feature extraction we constructed our own architecture for autoencoder.
  II. We started with a small architecture of 3 convolution layers along with a Max pooling layer and 3 transposed convolution layers for reconstruction of images. But this model struggled to capture fine details and variations in the facial images. This resulted in lower reconstruction quality and a less expressive image representation.
  III. Then we started increasing the number of layers in both encoder and decoder. But while increasing the layers we observed that as we went deeper the model was overfitting because of the small dataset.
  IV. So after trying different numbers of layers and different kernel sizes we finalized the architecture.
- **Adding batch normalization and dropout: -**
  I. We trained our autoencoder by using Adam optimizer. But while training it we observed that our model was converging very slowly. At starting 2-3 epochs loss was decreasing but after that it was constant. We tried with different numbers of epochs and learning rate but still it was not converging.
  II. So we added batch normalization after each convolution layer. And then we observed the improvement in convergence.
  III. And one more observation was about the number of parameters that we had to train. We had more parameters in the dense layer so we added dropout for the dense layer to avoid the overfitting.
  IV. After adding batch normalization and dropout we got the best result and also we got the good reconstructed images.

2. Choosing the Code size: -

- It is a crucial parameter as it determines the dimensionality of the feature space in which facial images are encoded.
- We started with a larger code size but it was not giving good results. Because projecting the facial images into larger dimensional space may contain a lot of noise.
- Instead of keeping it large we reduced it to a smaller value. And we got good results with it because smaller code sizes can capture only the most essential features.

Observations: -

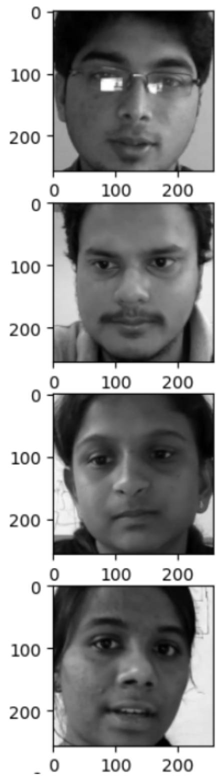
1. **Non-linearity: -**
    - **Autoencoders:** Autoencoders can capture complex and non-linear relationships in the data. This is particularly beneficial for face verification because faces are highly non-linear and vary in terms of lighting, pose, and expression.
    - **PCA:** PCA is a linear technique that works well when the data has linear structures. Faces, however, do not have simple linear relationships between their features. PCA does not capture the complex variations in facial images as effectively as autoencoders.
    - So autoencoders are giving better results as compared to PCA.

2. **Dimensionality Reduction: -**
    - **Autoencoders:** Autoencoders can reduce the dimensionality of the data while preserving important features. The encoder part of the autoencoder effectively compresses the information into a lower-dimensional code, which can then be used for face verification. This representation contains more meaningful and expressive features.
    - **PCA:** PCA reduces dimensionality by linearly projecting the data onto a new basis. While it retains most of the variance in the data, the resulting principal components may not be as informative or expressive as the features learned by autoencoders.
    - So using higher dimensions to represent the faces gives good results in case of autoencoder because it captures more complex and expressive features than PCA.
    - But if we use higher dimension feature space for PCA it will not give good results because it captures a lot of noise if we increase the code size.
    - We used feature space of dimension 100 for PCA and 200 for autoencoder.

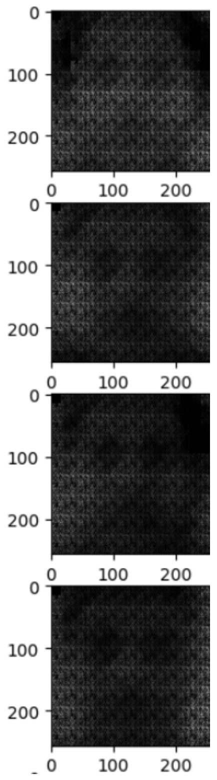
Results: -

- **Result of model without Batch normalization & Dropout: -**

Original Images

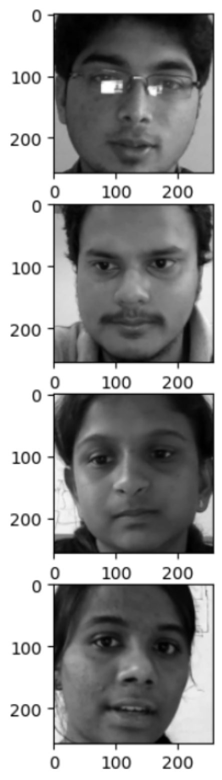


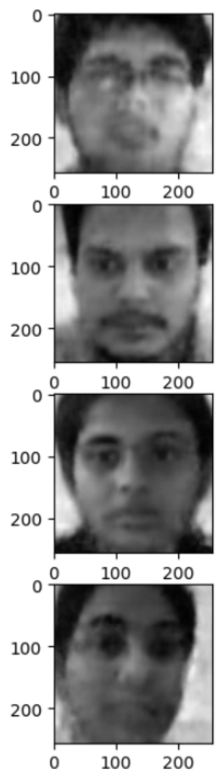
Reconstructed Images



- **Result of model with Batch normalization & Dropout: -**

Original Images **Reconstructed Images**





3.FACE VERIFICATION USING DNN AND PRETRAINED FACENET

In this part of the project we tried to implement face verification using a **DNN** and a pretrained **Facenet Network** by Google.

Methodology:

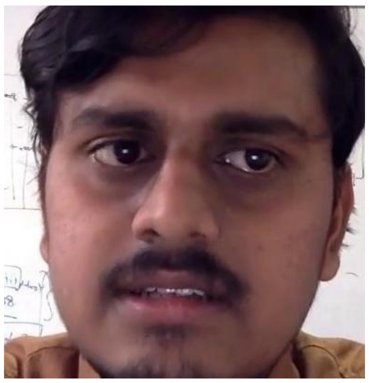
- **Preprocessing**

1.Face Detection & Cropping: One of the first steps in face verification is to isolate the actual face from the background of the image. This step chosen for various reasons:

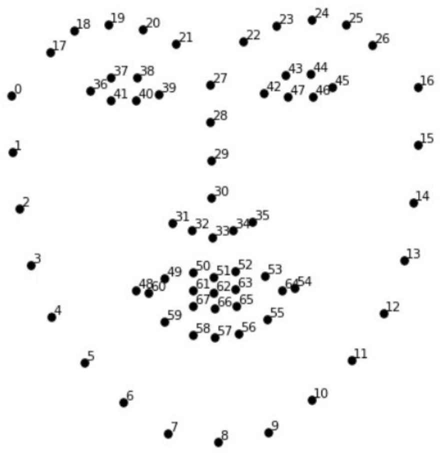
- **I.** It effectively eliminated a substantial amount of background and extraneous elements from the input data,
- **II.** It enhanced the extraction of facial features as face-detected images tend to have consistent positioning of facial landmarks like eyes, nose, and mouth.
- **III.** Face detection algorithms also are able to deal with bad and inconsistent lighting and various facial positions such as tilted or rotated faces.

Here for face detection we have used OpenCV's **haar cascade** classifier.

Original Image

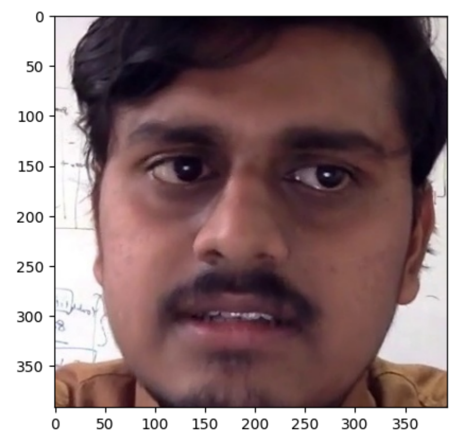

After Face Detection & Cropping

2.Face Alignment: When we isolate a face from the image, the face of a person can be projected in any way or the person may look in any direction. When we create an embedding of an isolated face, the embedding of the face of the same person projected in different ways can vary a lot.We will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces. To do this, we are going to use an algorithm called face landmark estimation.The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face. Then we will train a machine-learning algorithm to be able to find these 68 specific points on any face as seen in the figure:

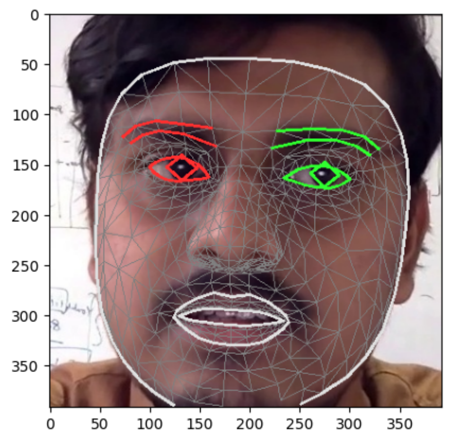


Facial Landmarks

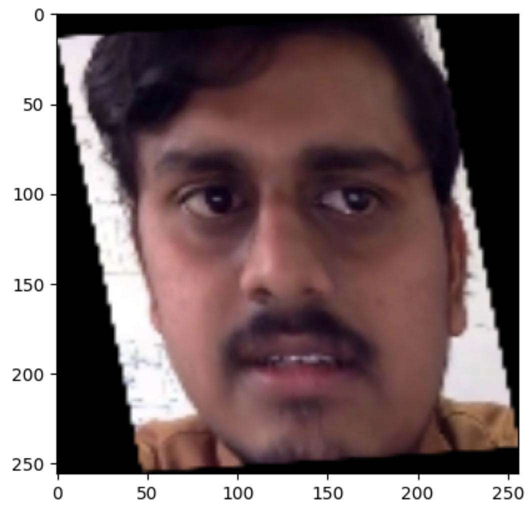
Now we can identify these landmarks in every face. We'll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible.



Cropped Face



Facial Landmark Estimation



Warped Image

Now no matter how the face is projected the output image having isolated face will always have outer eyes,nose & mouth positioned at the same position.This will make our next step to finding embedding of the face using Facenet a lot more accurate. We used **Google's Mediapipe** for facial landmark estimation.It employs a specialized deep neural network model for facial landmark estimation. This model is trained to predict the locations of key facial landmarks, such as the eyes, nose, mouth, and other facial features. The landmark estimation model takes the aligned face image as input and produces a set of 2D coordinates corresponding to these facial landmarks. Then we used OpenCv's **cv2.warpAffine()** function to warp the cropped images where the left eye will be aligned on coordinate (75,100), right eye on (125,100) and mouth center on (90,150).

- ## Model Architecture & Training Details:

Method 1: Training a DNN which has a softmax layer in the end containing 49 classes(as per our dataset) then evaluated our DNN model on our test dataset.

Architecture:

1. Model consists four layers of Neural Network
2. First layer is a dense layer giving 512 outputs with relu activation and dropout of 0.2.
3. Second layer is dense layer giving 512 outputs with relu activation and dropout of 0.2.
4. Third layer is a dense layer giving 256 outputs with relu activation and dropout of 0.2.
5. Fourth layer (output layer) is dense layer giving 49 outputs with softmax activation and dropout of 0.2.

We trained the model for 200 epochs first on cropped face images (first converted them into gray images) and got test accuracy of 85.06% and then on aligned images(first converted them into gray images) and got test accuracy of 93.98%.

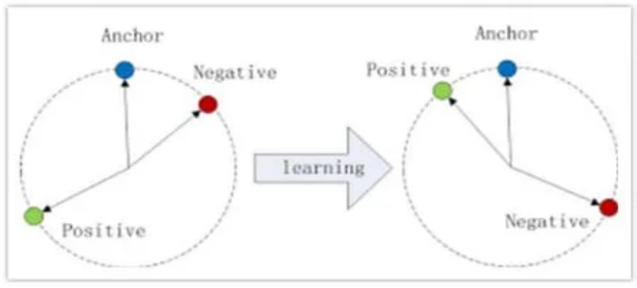
Method 2: Extraction of 512 dimensional vector from each image by passing the image through pretrained Facenet by Google. Each of this 512 dimensional vector represents facial features of each face which helps to uniquely identify that person. Using these feature vectors we trained a linear SVM on our dataset.

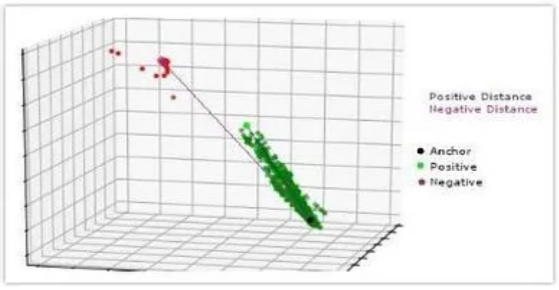
Facenet Architecture & Pretraining Strategy:

Now we need to find a way to represent each aligned face in numerical embedding. We can represent it using a pre-trained deep neural network OpenFace.During the pre-training portion of the OpenFace pipeline, 500,000 images are passed through the neural net. OpenFace trains these images to produce 128 facial embeddings that represent a generic face. OpenFace uses Google's FaceNet architecture for feature extraction and uses a triplet loss function to test how accurately the neural net classifies a face.

Triplet Loss Function:

To learn the parameters of neural network we need to train on every three different sets of images one is known to face image called anchor image, another image of the same person as anchor image(A) called a positive image(P), and 3rd image of a different person than anchor image called negative image(N).The network need to be trained in such a way that embedding of anchor image and positive image should be similar (i.e. closer in distance) and embedding of anchor image and negative image should be much farther apart or dissimilar.





Equation, $||F(A)-F(P)||^2 — ||F(A)-F(N)||^2 \leq 0$

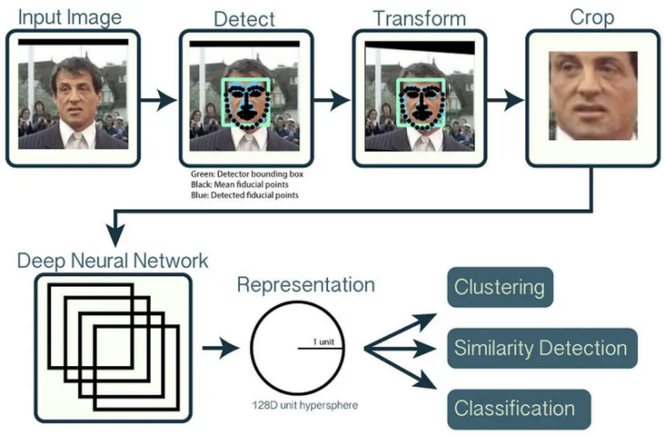
We want our dist(A, P) to be less and dist(A, N) be higher. To make sure NN does not just make dist(A, P) & dist(A, N) equate to zero to satisfy the equation so we decrease RHS from 0 to lesser margin value (α).Now equation boils down to , $||F(A)-F(P)||^2 — ||F(A)-F(N)||^2 \leq -\alpha$. Now for N triplets,

$$Loss = \sum_{i=1}^{N} \left[||f_i^a - f_i^p||_2^2 - ||f_i^a - f_i^n||_2^2 + \alpha \right]$$

So all we need to do ourselves is run our face images through their pre-trained open face network to get the 128 measurements for each face. The embedding is a generic representation of anybody’s face. **Unlike other face representations, this embedding has the nice property that a larger distance between two face embeddings means that the faces are likely not of the same person.**

Although the original FaceNet model extracts 128-dimensional vectors. However, there is also a 512-dimensional version of FaceNet that was developed by David Sandberg. The 512-dimensional version has been shown to achieve slightly better accuracy than the 128-dimensional version, but it is also more computationally expensive.Here we used 512-dimensional version.

Here during training at first we resized our images into (160,160,3) shape, then passed it through Facenet and got 512 dimensional representation of each image and then trained a linear SVM classifier for predicting the person. At first we tried this on cropped faces and got test accuracy of 97.95% , then we tried this on aligned faces and got test accuracy of 100%.



Entire Training Pipeline

Observations: -

- We clearly saw that in both the methods aligned faces worked better than the cropped faces. As in case of aligned faces no matter how original face is projected the output image having isolated face will always have outer eyes,nose & mouth positioned at the same position.This will make our next step to finding embedding of the face using Facenet a lot more accurate.
- We also saw that a pre-trained Facenet model performed much better than our DNN. The two main reasons are pre-training the network using 500k images and the innovative pre-training strategy using triplet loss.
- The 512-dimensional version has been shown to achieve slightly better accuracy than the 128-dimensional version, but it is also more computationally expensive.But the KNN classifier is heavily affected by the curse of dimensionality in case of 512 dimensional vectors. If we have only a few training samples, 128-d embedding or even 64-d embedding are a better choice or we can use a linear SVM classifier.

Comparison Of All The Approaches: -

Approach	**Raw Images**	**Face Detected Images**	**Face Aligned Images**	**Accuracy**
PCA + KNN	YES	-	-	0.87
PCA + KNN	-	YES	-	0.89
Autoencoder without Batch Normalization	-	YES	-	0.83
Autoencoder with Batch Normalization	-	YES	-	0.92
DNN	-	YES	-	0.85
DNN	-	-	YES	0.93
FACENET + SVM	-	YES	-	0.97
FACENET + SVM	-	-	YES	1.0

Conclusion: -

In this part of the project we employed three key techniques for face verification: PCA, autoencoders, and FaceNet.

- PCA reduced dimensionality efficiently but struggled with complex variations.
- Autoencoders improved adaptability and resilience to variations.
- FaceNet delivered unmatched accuracy.

PCA provided a foundational step, autoencoders enhanced adaptability, and FaceNet delivered exceptional accuracy.

Task 2: Face Jewellery

- **Augment a jewel on the nose of a user real-time**
- **Document your approach and performance observations**

I. Introduction: -

In the world of augmented reality (AR) and computer vision, real-time facial augmentation has become increasingly popular and sophisticated. One exciting and creative application of this technology is the ability to add virtual jewelry, such as nose rings, to a user's face in real-time. This task challenges us to develop a system capable of augmenting a piece of jewelry, in this case, a nose jewel, onto a user's face in real-time.

We will outline the approach taken to accomplish this task, including the underlying technologies and algorithms used. We will also provide observations, highlight any challenges and potential improvements.

II. Steps Performed: -

1. Load the pre-trained face and nose detection models (Haar Cascade)
2. Load the nose ring image
3. Define the resizing factor
4. Capture Video from WebCam
5. Get the video's frames per second (fps) and frame dimensions
6. Define the codec and create a VideoWriter object to save the output
7. while True:
    a. Read a frame from the video
    b. Convert the frame to grayscale for face detection
    c. Detect faces in the frame
    d. for coordinates in face
        i. Calculate the position(Region of interest) for placing the nose ring
        ii. If at least one nose is detected, take the first one and break the loop
            - ❖ Resize the nose ring image to fit the detected nose
            - ❖ Create a mask for the white pixels in the nose ring image
            - ❖ Invert the white mask to make white pixels transparent
            - ❖ Calculate the position for placing the nose ring within the face ROI
            - ❖ Combine the nose ring with the frame, making white pixels transparent.
    e. Write the frame with the detected faces and noses to the output video.

III. Haar feature-based cascade classifiers

It is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "***Rapid Object Detection using a Boosted Cascade of Simple Features***" in 2001.

Algorithm Steps:

1. **Haar-like Features Definition:**
   Haar-like features are simple rectangular patterns that are used to describe an object's characteristics.

2. **Integral Image Calculation:**
   To speed up the computation of Haar-like features, an integral image is calculated from the original image.

3. **Adaboost Training:**
   Adaboost (Adaptive Boosting) is a machine learning algorithm used to select a subset of the most discriminative Haar-like features.

4. **Cascade of Classifiers:**
   The cascade is a series of stages, each containing a set of Haar-like features.

5. **Object Detection:**
   During detection, the cascade of classifiers is applied to the integral image of the input image.

6. **Non-maximum Suppression:**
   Non-maximum suppression is applied to remove redundant detections by keeping the most confident one and discarding others.

7. **Post-Processing and Visualization:**
   Once the object is detected and localized, post-processing steps can be applied, such as drawing bounding boxes around the detected objects.

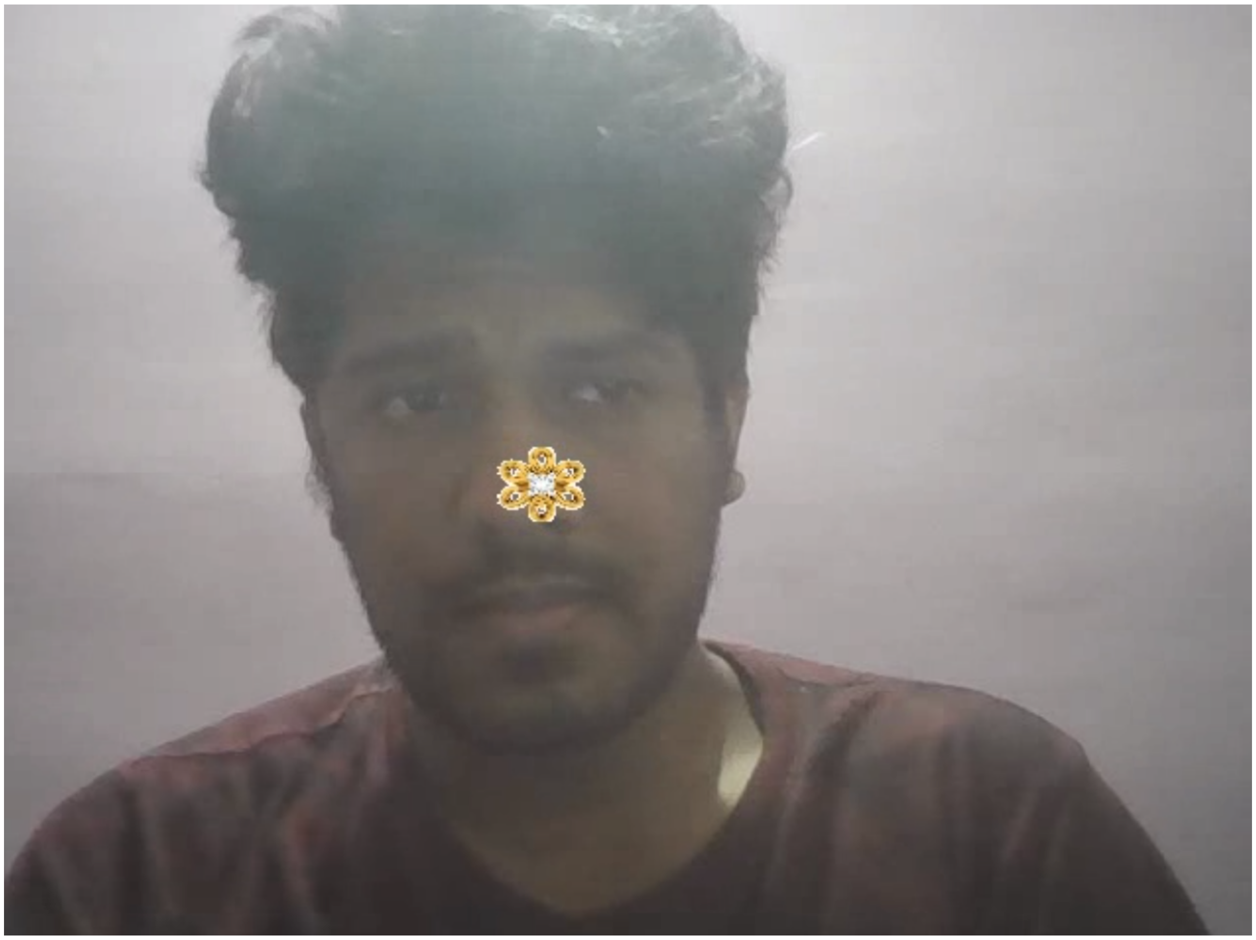
IV. Challenges

- When we used just a nose detector by Haar feature-based cascade classifiers instead of using face detection, many objects were misunderstood to be detected as nose when compared to face detection strategy.
- So we tried face detection and placing the nose ring at a specific position where the nose lies, but with different face sizes the results were not good.
- Nose ring image had a white background which had to be made transparent.

V. Observations and Discussion

- Multiface augmentation of the nose ring in user real time was possible.
- Since we faced the challenge using only face detection and placing the nose ring at a specific position, we used face detection as well as a nose detector in the frame where each face region was detected which gave good results.
- For making the background of the nose ring image transparent, we created a white mask for the background pixels in the nose ring image and performed inversion(logical not) with the white mask to make it transparent.

VI. Results



Video Output :Link