International Institute of Information Technology
Bangalore

Advanced Visual Recognition and Rendering
AI-836

Mini Project Report

Sarthak Harne
-IMT2020032

Monjoy Narayan Choudhury
-IMT2020502

Abhinav Mahajan
-IMT2020553

September 13, 2023

Task 1

Aim

The aim is to build a face verification system and compare the performance of different methodologies on in-sample and out-of-sample data. The objective is to also develop a meaningful latent space representation of faces specifically for the downstream task of face verification.

General Approach

1. We compare the following metholodgies to solve the Face verification problem:
   (a) PCA (Prinicpal Component Analysis)
   (b) AutoEncoder architectures
   (c) Deep Neural Networks and other pre-trained models
2. To compare the above, we build a general, re-usable framework and we generalise the Face Verification problem to a pairwise learning problem. Wherein, when given a pair of images of faces and we have to classify whether it belongs to the same person or not.
3. To perform the same, we take the old dataset of IIITB-Faces, and generate pairs of images, and generate training and testing pairs. Also, we reserve an arbitrarily chosen person's faces and not include them in any of their photos in the in-sample training and testing pairs, so that we can perform out-of-sample testing as well. This serves as metric of generalisation as well as scalability of models.

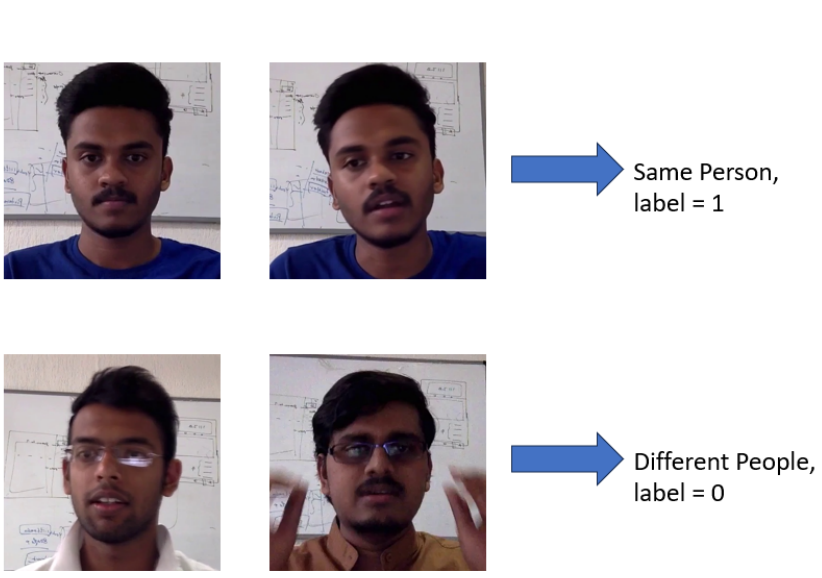

Figure 1: Dataset contruction: First we split training and testing images. Then among we the training images, we form pairs as shown above. For In-sample testing, we choose a picture from the testing set, and 1 from (testing set)U(training set). Now the training and testing images have at least 1 photo from every individual, but 1. That 1 person's photos have been reserved for out of sample testing. In out of sample testing, we form pairs of images, in which 1 comes from the reserved set, and 1 comes from anywhere in the whole dataset, including the reserved set.

4. The pairs of data will be used to train models in a Siamese setting.
5. In pairwise generation, there will be a huge bias of pairs in which the faces do not belong to the same person, and therefore models will be biased towards classifying the faces as dissimilar. We take care of this with a Weighted-Random-Sampler, so that during training, the models see an equal proportion of similar and dissimilar faces. This is applicable for the AE and DNN approaches (b.) and c.)), whereas for PCA we make use of normalised accuracy. The exact details will be elaborated later.

6. PCA (method a.)), serves as the de-facto data-driven heursitic/meta heuristic unsupervised method of performing the task, whereas methods b.) and c.) serve as data-driven self-supervised learning approaches, in which the AE model is completely off-the-shelf and the DNN method involves a pre-trained backbone. This project is a comprehensive case study of the methods.

7. Also, as mentioned before, our objective was to build and compare the representatives and meaningfulness of the latent spaces in each method. Therefore, in each method we ultimately project a final 128 dimensional latent space, and then use cosine similarity between the 128 dimensional vectors of the pairs of data to perform the classification task. We refrained from training a pure classifier on top of the latent space (for eg. Applying Logistic Regression on top of the 128 dimensional vector) because the weights of the classifier would off-load the meaningfulness of the latent space. In the method we have used, we show that using the latent space alone for direct classification, we are able to get great results.

A.) PCA (Principal Component Analysis)

Principal Component Analysis has been explored in the fields of Face Recognition, EigenFaces [1], is the method of dimensionalality reduction of the image space, so that face's can be generalised and represented in a de-noised fashion in the latent space and it can be used for various tasks such as verification, recognition etc.
Given a dataset/datamatrix, we can calculate the covariance matrix of the features of the same. Upon evaluating the eigenvalues and eigenvectors of it, by sorting the eigenvectors by the descending order of their eigen values, we end up getting a new coordinate space which minimizes the covariance of the projection of the original dataset onto the eigenvectors coordinate space. We perform dimensionality reduction by choosing the n most prominent vectors and discarding the rest. These prominent vectors are the most dominating vectors in minimizing the aforementioned covariance, therefore upon choosing the top-n eigenvectors, we can represent the data in a de-noised, generalised and meaningful way.

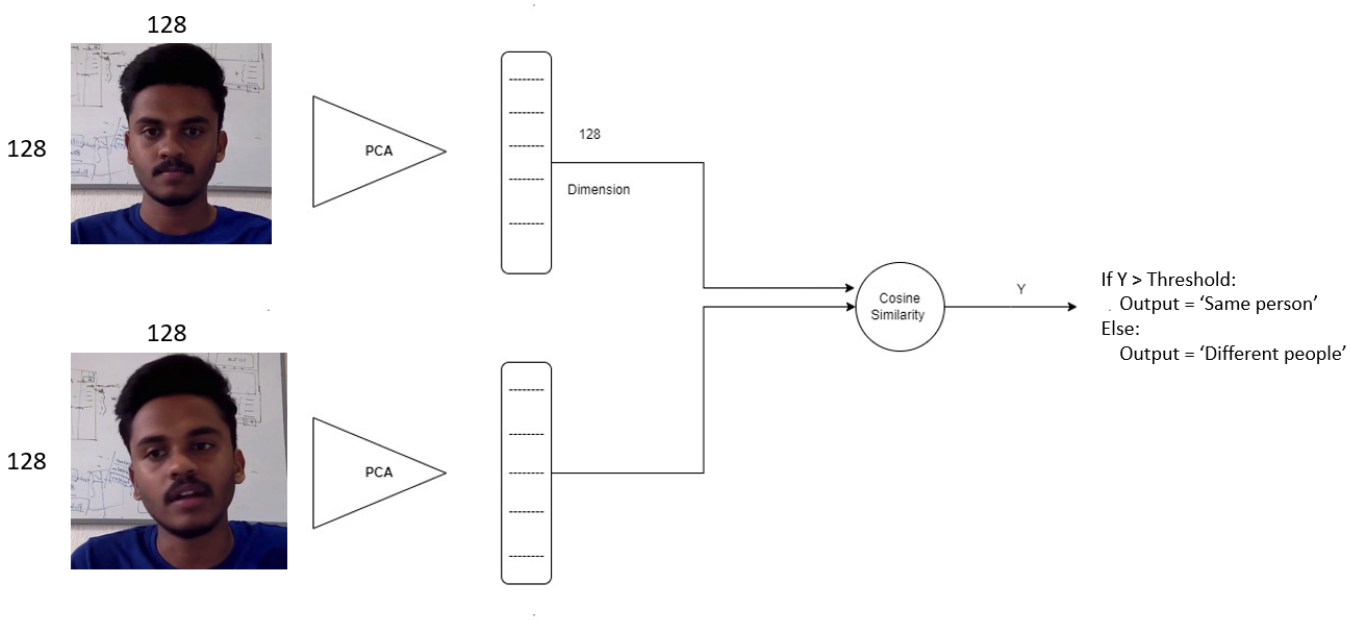

Figure 2: PCA Model Diagram

To perform the face verification task, we follow these steps: -

1. Firstly, we treat this as a meta-heuristic task, in which we are not learning anything iteratively.

2. In the next step we extract all the images in the dataset. We then divide them into training images and testing images. Also, as mentioned before, we arbitarily choose all photos of 1 person, and do not include any of them in the training set. So that this can be used as out-of-sample testing, since during training it hasn't seen even one photo of this individual.

3. By training images, we mean the data the PCA model is allowed to look at, or the data on which PCA has to be applied to find a generalised set of eigenvectors/reduced dimension space.

4. Also, we resize all images to (128, 128) (to have a standard coordinate size) and convert them greyscale images (PCA seems to work well on Grayscale images, compared to RGB images, and this is because the image space/pixel space reduces by a factor of 3, which in itself is method of denoising the data. [2])

5. We have chosen the value of n (n is the size of reduced feature space) as 128, for the sake of consistency among all the methods. Therefore, we can represent any face image with 128 features, irrespective of their height, width and channels

6. Once we have fit our PCA model on the training images, we transform all the images in the dataset (training, in-sample testing as well as out-of sample testing images) to its 128 dimension vector.

7. We then make use of pairs to evaluate its performance on Face Verification. As mentioned before, we treat this as a classification task, wherein we have to classify whether the 2 input images belong to the same individual or not.

8. After several experiments, the following method gave the best results for the classification task. In the set of training images, we make pairs of images. And since its labelled, we also mention a 'label' demarcating whether they belong to the same individual or not. We then find the cosine similarity of the 128 PCA dimensions/EigenFaces of the images in the pairs of training data, and all we have to do is find an appropriate threshold. The essence of the 'threshold' lies in the folllowing, if the cosine similarity of the reduced feature space of a pair of images is $>$ threshold, it is considered to be classified as 'Same Person', else, if it is $<$ threshold,m it is classified as 'Different People'. If it was another Machine Learning task, we could have hardcoded the threshold and during the learning phase, our backbone would have learnt how to marignalise the cosine similarity to fit the threshold (like in the case of Contrastive Loss, where the margin is hardcoded). However, in this case, where we are treating the whole process meta-heuristically, we find the threshold heursitically to maximise the number of correct observations. Since precision of the threshold's is not of the upmost importance to us, we discretise the range from 0.0.1 to 0.99 and we take steps of 0.1. The number of correct observations are defined as number of similar pairs $>$ threshold + number of dissimilar pairs $<$ threshold.

9. The problem with the above is that the number of dissimilar pairs in the training data is » number of similar pairs. This leads to a bias, in which our threshold will be skewed towards always observing two faces as dissimilar. Assuming this prior probability of sampling isn't important if the data is sampled independently, we normalise our number of correct observations formula. Our new formula is number(similar pairs $>$ threshold)/number(similar pairs) + number(dissimilar pairs $<$ threshold)/number(dissimilar pairs). This normalised value ensures equal weightage to similar and dissimilar pairs.

10. A perceptron, SVM could have been fit, or the threshold could'vbe been binary searched for, but since our time complexity is lesser than fitting the PCA model and transforming all the data, threshold finding is not the bottleneck and hence, we stick with the algorithm defined above.

Results

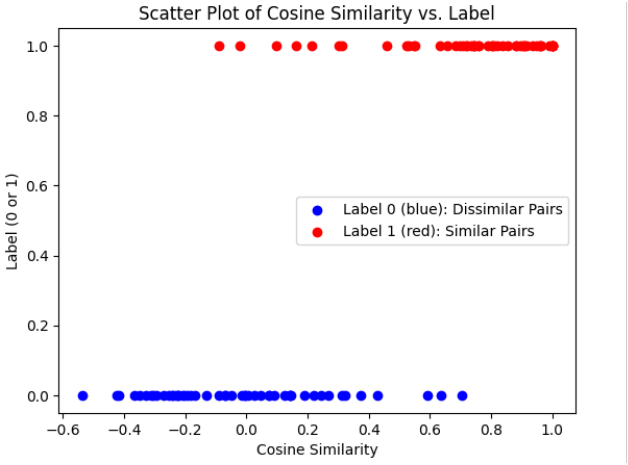

Figure 3: Taking 50 random Same person pairs and 50 random different person pairs, and upon plotting their cosine similarity, we can see there is a definite cluster of Similar Pairs around a higher cosine similarity, wheras a smaller cosine similarity for dissimilar pairs, which is a great hypothesis test of using cosine similarity as a distance metric. Now we are tasked with finding the ideal cosine similarity threshold such that maximal labels are seperated in the dataset and fairly.

1. When we find the threshold by maximising our un-normalised accuracy metric, the threshold we get is 0.74. Using that threshold:
   (a) Un-Normalised classification accuracy on training data = 98%
   (b) Un-Normalised classification accuracy on in sample testing data = 98%
   (c) Un-Normalised classification accuracy on out of sample testing data = 98%

   (a) normalised classification accuracy on training data = 73%
   (b) normalised classification accuracy on in sample testing data = 73%
   (c) normalised classification accuracy on out of sample testing data = 65%

2. When we maximise our normalised accuracy matrix, the threshold we get is 0.38. Using that threshold:
   (a) Un-Normalised classification accuracy on training data = 91%
   (b) Un-Normalised classification accuracy on in sample testing data = 91%
   (c) Un-Normalised classification accuracy on out of sample testing data = 90%

   (a) normalised classification accuracy on training data = 87%
   (b) normalised classification accuracy on in sample testing data = 87%
   (c) normalised classification accuracy on out of sample testing data = 78%

3. The problem with the first method is that it is heavily biased towards dissimilar pairs, and it has a normalised accuracy of 73-65%, due to the fact that it classifies similar faces as dissimilar because of its harsh threshold.

4. The problem with the second method is that it is lenient in face verification, and will seldom allow dissimilar faces also to pass as similar faces which may pose a security threat.

B.) AutoEncoder Approach

AutoEncoders is a famous data compression and reconstruction framework. Where the emphasis is to compress the input feature space to a smaller dimensional space, and then upsampling is performed to reconstruct the data. Autoencoders often have the simplest architectures in the family of reconstructive encoders (compared to VAE, beta-VAE etc.) and often do-not perform ideal re-construction and are very rarely used in practice as

generational models. But they are still used in combination with other networks for other non generative tasks, and reason for this is because the latent space of autoencoder models is more representative and meaningful than conventional convolutional encoders, because the reconstruction loss enforces more global features of the input feature space in the latent space, as the compressed data would be tasked to reconstruct the input image. This enrichment of latent space has been used and explored extensively for face verification and recognition tasks. [3] [4].

1. Abstract: Using training and testing pairs generated in the same way as mentioned above (A.) PCA), we build a dataloader with weighted random sampling to take care of class imbalance, and then use a AutoEncoder (AE) architecture with a small backbone, and project the Z/Latent space of the AE to a fully connected block which ultimately gives us a 128 dimensional vector. Contrastive loss along with reconstructive loss is applied and the weights of each loss is tuned and in one training loop, the entire architecture is trained. Finally for inference and evaluation/testing, we input pairs of faces, and if the Distance metric (Euclidean distance), same as the one used in Contrastive Loss, is $>$ margin, the pair is classified as 'Dissimilar', else they are classified as 'Similar'.

2. Our model looks like this: -

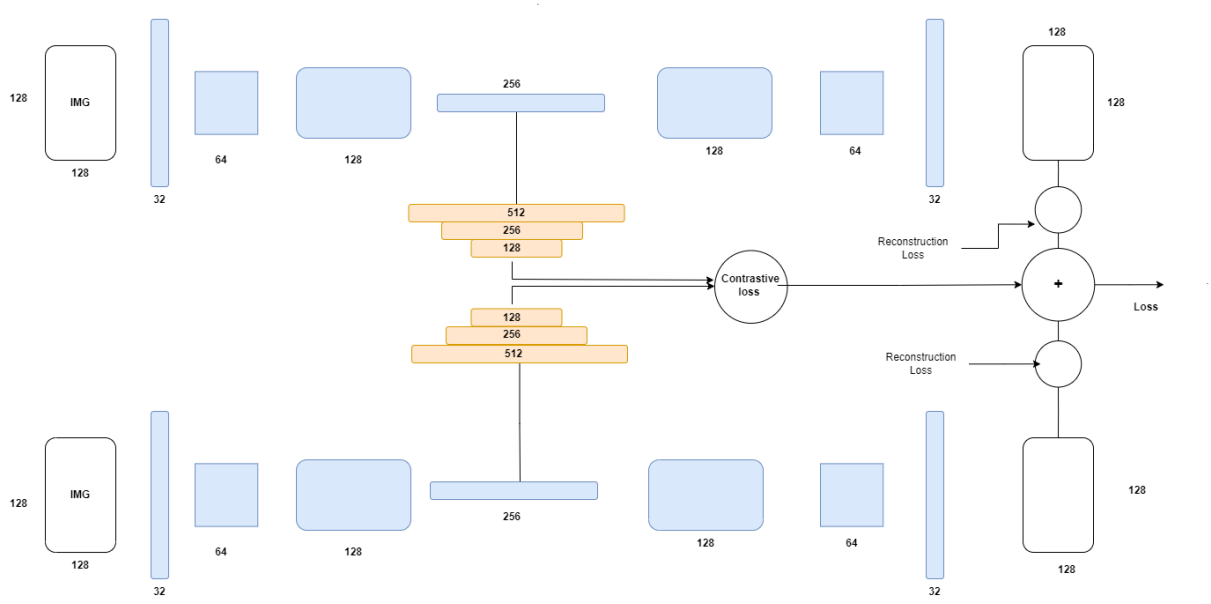

Figure 4: Model Architecture

3. To provide our model justifications, we would now provide a qualitative ablation study.

4. Firstly, we have huge dataset imbalance, which arises due to the way we have coded our dataloader. Due to the imbalance of data, if we simply sample data from all sets of possible pairs, we will end up biasing the model towards always guessing faces as dissimilar. Now if this same verification system was used in phones, or as security scans to enter a place, we would never be able to get passed. Although, we will be able to get a nice classification accuracy, if the probability of 2 faces being similar and dissimilar input to the model is the same, in that case our model would fail to generalise. Therefore, we introduce the metric of normalised classification accuracy, as seen in (A.) PCA). But that's just a emtric to evaluate our model after training, we need to enforce removing the bias at training time itself, and therefore, we make use of PyTorch's WeightedRandomSampler. Its functionality is super simple to use, After defining our custom dataloader, we simply define the weights/probabilites of each pair,and for the sake of simplicity, we assume the pairs follow a uniform distribution among themselves, therefore the probability/weight of any weight(sample) $=$ 1/number(samples $=$ label), where number(samples=label) is the number of samples in the dataset in which the label is 1 or 0 ('Similar' or 'Dissimilar'). PyTorch uses this weight and would sample during training pairs of data with equal probability of coming from a 'Similar' and 'Dissimilar' Distribution. Using this method, our normalised testing classification accuracy increases from 0.523 to 0.80!!

5. Due to small dataset, we restrict our model size. With a smaller backbone, we run the risk of underfitting and with a bigger backbone, we run the risk of overfitting. 4 ConV layers with 3 FC layers got the highest normalised classification accuracy out of our experiments with this architecture.

6. Our choice of loss this time out was Contrastive Loss. There are multiple alternatives, we can also find the cosine similarity and then treat it like a probability and then apply Binary Cross Entropy Loss on it. Contrastive Loss with margin 0.2, and distance metric as Euclidean Distance gave much better results, and infact, the model was unable to train when contrastive loss wasn't used, the Loss function stagnated and and the weights were unable to be adjusted, and this may be because we aren't using a pre-trained backbone and all the weights are Xavier Initlialised, which is the de-facto method of initalisation in PyTorch. [5] (Note: This citation is not the introduction of Xavier Initialisation, but merely talks about it and other techniques and their relevance to DNN's)

7. Also it's an AutoEncoder architecture, therefore we also provide reconstruction loss in the training loop, and the reconstruction loss was chosen as Mean Squared Error. Since our primary focus is Verification, our contrastive loss gets much higher weightage than reconstruction loss, as it will act asa tradeoff, and we want to maximise our training accuracy. Also its noteworthy, with the reconstruction loss, our normalisation accuracy increses from 0.78 to 0.80, so the AE architecture does provide some better results than just using a Convolutional Encoder.

8. Through manual tuning, our learning rate of 0.001 and Adam Optimiser without weight decay gave the best results. Number of epochs was chosen by Early Stopping method, the replace parameter in WeightedRandomSampler were chosen by experimentation and intuition and so were others such as margin=0.2 and distance parameter=Euclidean distance (in Contrastive Loss) and dropout rate (in fully connected layers, convolutional dropout is ill-adviced and generally doesn't perform well).

Results

1. These are our results on margin = 0.2 and distance metric = Euclidean Distance

   (a) Un-Normalised classification accuracy on training data = 98%

   (b) Un-Normalised classification accuracy on in sample testing data = 97%

   (c) Un-Normalised classification accuracy on out of sample testing data = 91%

   (a) normalised classification accuracy on training data = 92%

   (b) normalised classification accuracy on in sample testing data = 91%

   (c) normalised classification accuracy on out of sample testing data = 90%

C.) Pre-trained DNN methods

In the autoencoder method, we lacked a pre-trained backbone which made is significantly harder to train. Also our downstream task is critical to the performance and quality of our backbone. We needed to find the optimal set of parameters for B.) and then only were we able to get results, which required extensive efforts. Whereas in pre-trained models, since they have already been exposed to a a substantially bigger dataset, and they've generalised to a variety of downstream tasks, they very easily finetune to custom task specific datasets. Their networks are deeper and and they've been trained using state of the art techniques with industrial level computational strength and this is the reason way their latent spaces are so informative and meaningful. Infact, certain DNN's can directly be used without finetuning for our dataset, since our verification task is comparitively easier than recognition. However, we would finetune it and our results show that with minimal effort, one can get great results, and out of all the models and architectures explored so far, this method gave us the best results. And let's take a deep dive into that.

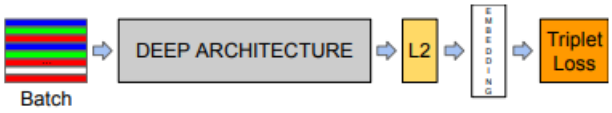

Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

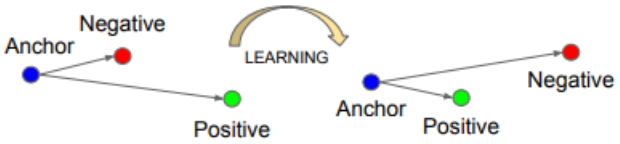

Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Figure 5: Highlevel Facenet Architecture

1. The Pre-Trained Deep Neural Network we have chosen is the FaceNet-Inception Resnet V1 architecture [6] and pre-trained weights.
2. From the name itself, we can have a flavour of the architecture, they've used Google's Inception-Net architecture [7], with the added residual connections which was the theme of the resnet paper [8]. Click here to see the code of the architecture.

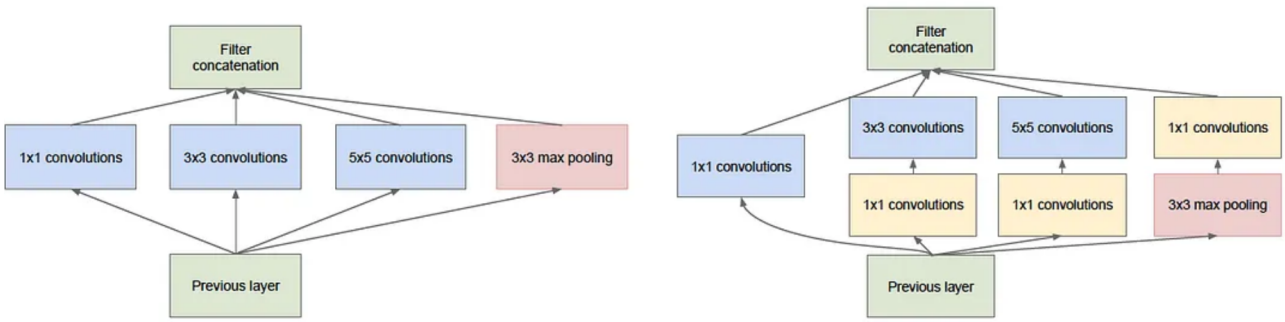

Figure 6: Inception architecture

3. The main idea of InceptionNet was to perform parallel convolutional filtering to reduce number of parameters. You can see from the image below how it looks like, this is used as the Vision/CNN backbone of the FaceNet architecture. Keep in mind, the FaceNet architecture comes in 2 variants, The one Inception-Resnet architecture as mentioned and also the Zeiler and Fergus architecture. We stick to the former as it performs better.
4. The architecture was pre-trained for a similar task such as ours, but it was in a triplet setting. In a triplet setting, you have an anchor, a positive and a negative example. The anchor and the positive examples are different images of the same person and the the negative example is an image of someone else. However, to make the model more robust, we may choose confusing pairs, so that our model learns better representativeness, in this case the anchor and the positive pair may suffer from bad lighting or orientation or occlusion or pose variance and the negative pair maybe a look-a-like or another confusing pair. This method of training leads to much better training and this explains our results. Also, using hard negatives and hard positives may also lead to overfitting and the model may fail to generalise, to overcome this, the authors came up with semi hard pairs, and they introduce it to the model, where in teh distance between the anchor and positive pair and anchor and negative pair is approximately same. Also, a novel [9] method of training was chosen by using cross entropy loss to train facenet, we take inspiration from the same and train.

5. It is also noteworthy that, an idea worth venturing in the future is adaptive margins in triplet loss. These adaptive margins give the pairs a relative softness or hardness, but also this is very dataset and task dependant job, but its an area worth venturing, and can give promising results.

6. We train it on the same datapairs as mentioned before, but this time, we use our 128 dimension vector, calculate the cosine similarity and if its > 0.75, they are considered similar, whereas they are considered dissimilar. We are allowed to hardcode this threshold, as the label predictions are then used for Binary Cross Entropy loss as mentioned above and this gives the best results.

Results

1. These are our results on threshold = 0.75 and distance metric = Cosine Similarity

   (a) Un-Normalised classification accuracy on training data = 97%

   (b) Un-Normalised classification accuracy on in sample testing data = 97%

   (c) Un-Normalised classification accuracy on out of sample testing data = 98.6%

   (a) normalised classification accuracy on training data = 95%

   (b) normalised classification accuracy on in sample testing data = 95%

   (c) normalised classification accuracy on out of sample testing data = 94%

Task 2

Aim

For this task, our objective is to inpaint/ put a jewel in the nose of a face from a video feed (which can be a live webcam or a pre-loaded video).

General Approach

To tackle this task we broke down the given task into 3 simple blocks:

1. Facial Landmark Detection: This involves running a face detector and on the detected face we need to get all the facial landmarks as a list.

2. Facial Landmark Extraction: Given the list of facial landmarks we must select the relevant landmarks for the region of the face where the jewel can be painted. In our setting and experiments, we have taken a fixed set of landmarks corresponding to where we want to put the jewel. However, this can easily be made user preference and lead to many creative applications in the commercial space.

3. Inpainting the jewel: Finally given the required "interesting" landmarks corresponding to the location where we want the jewel, we need to inpaint / overlay the jewel on the image frame. We discuss the various post-processing approaches we have tried to improve the realism of the solution in the upcoming sections.

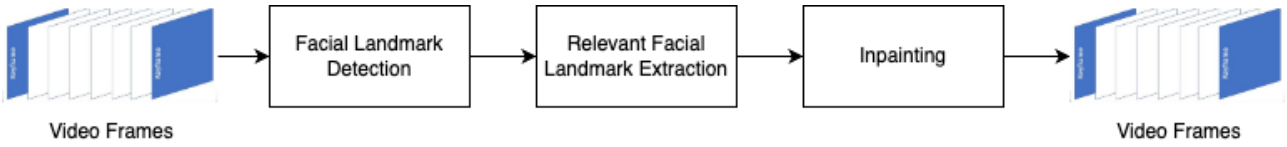

Figure 7: General Workflow

Approach 1: Mediapipe with Naive inpainting

For our experiment, we use the Mediapipe facial landmark detection pipeline for the Facial Landmark Detection Block. The above landmark detector is based on the paper "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs" by Bazarevsky et al. The authors intended this pipeline to be real-time in performance with fps guarantees between 200 to 1000 which is well above the frames we handled in our experiment (30 fps) on our local machine. Figure 8 shows the working of the pipeline. Following are the salient feature of the architecture:

1. A lightweight convolutional feature extractor that is very similar to mobilenet that tries to leverage depth convolutions with larger receptor fields as increasing size for depth convolution filter is cheap.

2. Highly parallelizable anchor box based on single-shot multi-box detectors. The authors observed that similar performance can be observed as heavier models that have more downsampling layers and fewer anchors per pixel over in the case of models with fewer downsampling layers and more anchors per pixel. The authors changed from 2 anchors per pixel to 6 anchors per pixel for an 8x8 filter-based downsampling layer.

3. Weighted sum post-processing over Non-Maximal Suppression. As seen above since the anchors per pixel have increased, the amount of overlap per pixel for a prediction also increases. In general schemes, this would increase the computation for a NMS scheme. To tackle this, the authors use the weighted sum of all the bounding boxes that the model predicts to predict the final box. The cost for the operation is the same as NMS in a scheme with fewer clashes. However, this has been observed to increase the accuracy by about 10%.

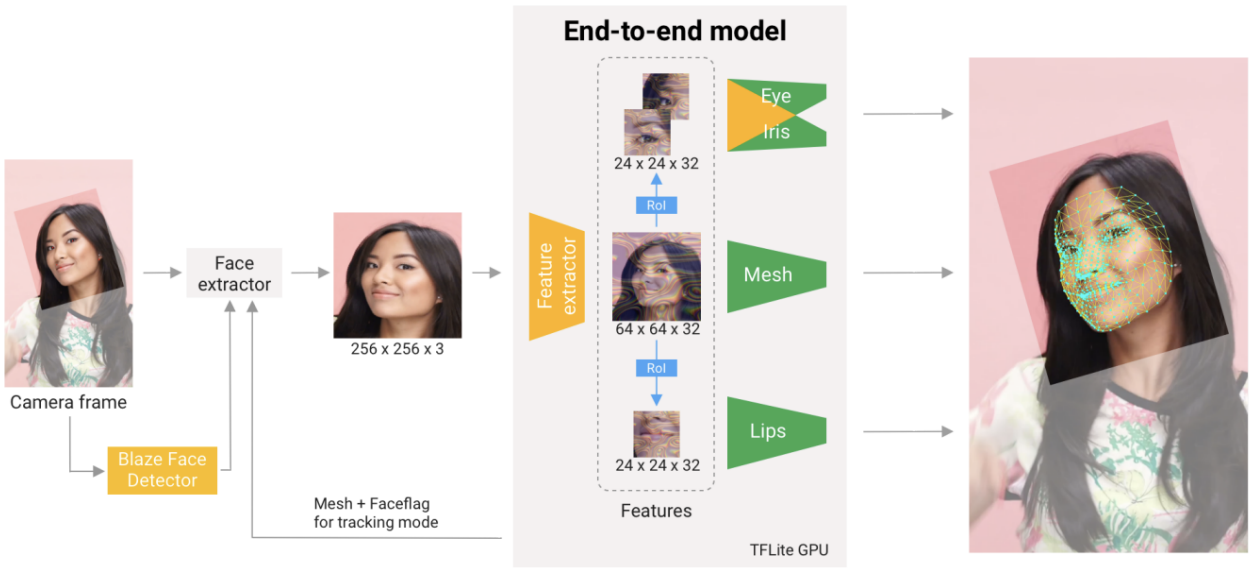

Figure 8: Courtesy of [10] Mediapipe pipeline

Once we get the relevant landmark points. We need to decide which points to use for the inpainting. This step is relatively easy due to the comprehensive mediapipe documentation which clearly cites which entry of a FacialLandmark Object lies on which part of the face as shown in Figure 9.

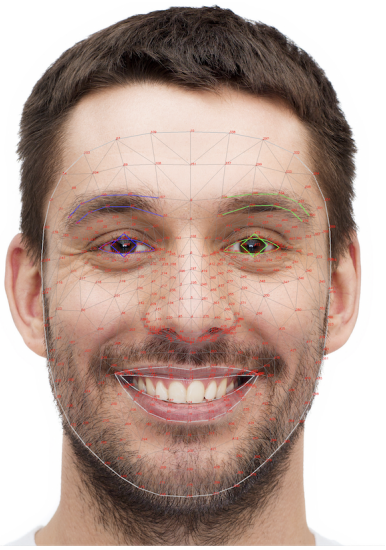

Figure 9: Facial Landmarks with their corresponding ids

In our naive implementation, we then simply use the OpenCV2 library to inpaint a circle on the nose of the subject and output the frame.

Experiments

We tried our pipeline for various settings possible. The settings consisted of lighting variation, camera distance, and motion of the head, and have noted our observations that will be discussed in the next section.

Observations

Following are the observations for the experiments performed:

1. For a more or less static face in the video the jewel looked good enough to seem to be embedded in the case of a symmetric jewel (like a dot). This can be seen in figure 10. When the jewel was replaced by a complicated texture, based on the shape of the texture it may or may not look realistic. This experiment can be seen in figure 11.

Figure 10: Simple solid fill jewel

Figure 11: Annular ring

Figure 12: Complex texture

2. For rapid movement of the head the model is unable to keep up with the fast-changing of frames leading to outputs with artifacts.

3. For medium light setting, mediapipe was still able to detect the face and consequently get the landmarks, and hence a respectable output was still produced. However, when we reduced the lighting conditions to an almost dark mediapipe still got some landmarks out, however, the jewel rendering becomes jittery due to difference in the result of the landmark detection changing in consecutive frames rather than in a range of motion. This is due to the fact that the detector is giving an approximate result rather than a result with high confidence due to the poor lighting condition. Both the situations can be seen below in figure 13 and figure 14

4. In the case of occlusion there is no provision to prevent the rendering of the jewel which leads to unrealistic frames. We aim to handle this better in the next section.

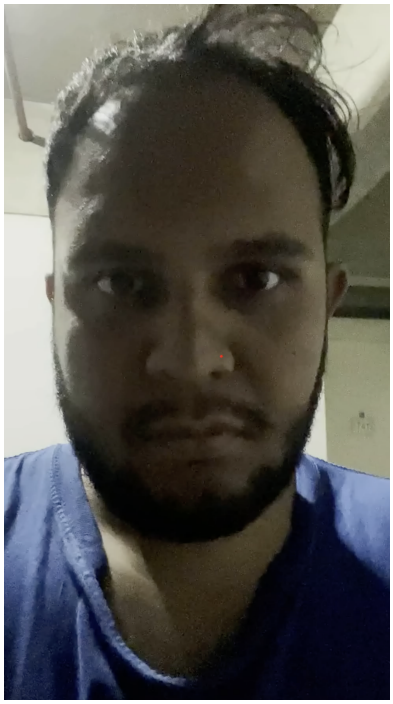

Figure 13: Medium lighting

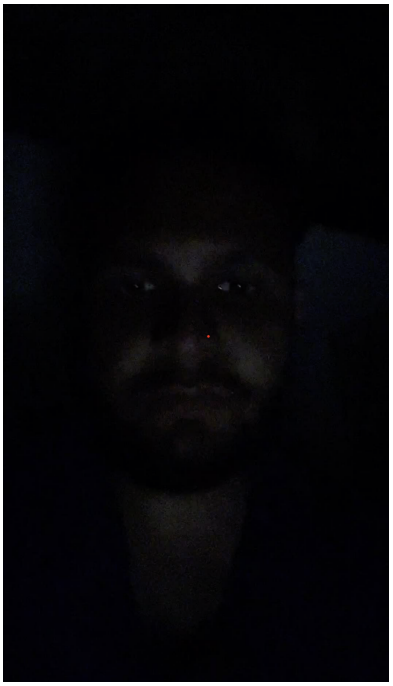

Figure 14: Almost Dark

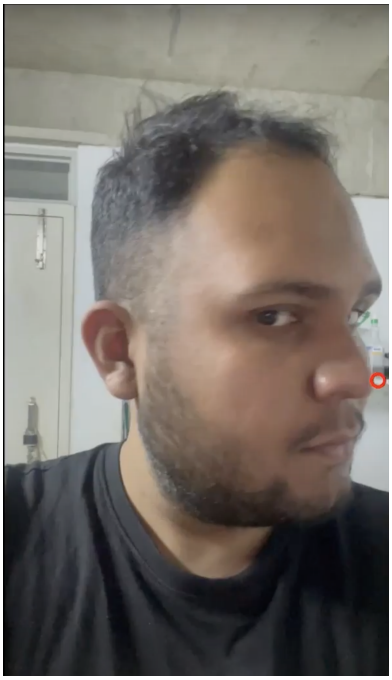

Figure 15: Occlusion case

Approach 2: Handling Occlusions

Approach 2.1: Naive Normal Dot Product

Although the first approach is able to put the jewel at the correct position, it is not able to handle occlusion, the part where the part of the nose itself is not visible because the head is turning.

The good part of mediapipe is that it provides the z coordinates, which provide the depth information. Using this, we can use the normals to the triangles in the mesh, and the image plane to decide if the jewel should be visible. For this we can have the vectors like this:

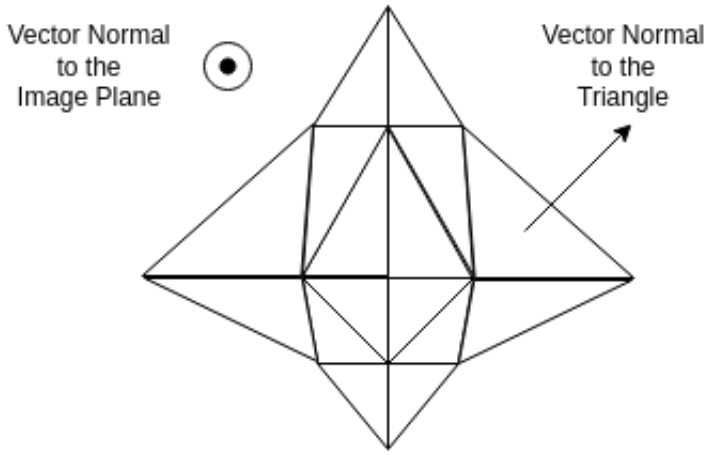

Figure 16: The two vectors being considered

Using these two vectors we can determine if the region we are interested is visible or not by taking a simple dot product. If this dot product is positive, then the angle between the vectors will be less than 90° and the region will be visible. Conversely, if this dot product is negative, then the angle between the vectors will be greater than 90° and the region will not be visible.

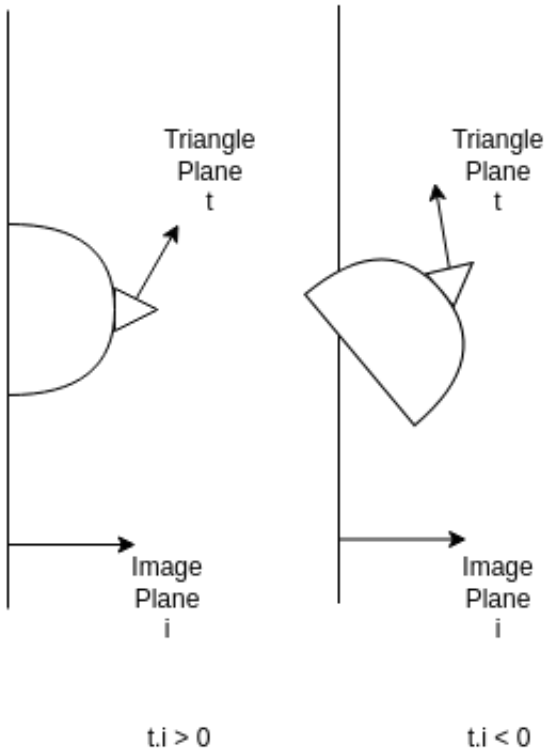

Figure 17: Top View: Dot Product between the triangle plane and the image plane

Procedure

For performing this task, we considered one triangle on the side of the nose of the person (the chosen triangle is 440, 363, 360). The jewel is placed at the centroid of this triangle. Let the vectors to the vertices be denoted by $\vec{v_1}$, $\vec{v_2}$ and $\vec{v_3}$. The normal to this triangle is $\vec{n_t} = (\vec{v_1} - \vec{v_2}) \times (\vec{v_2} - \vec{v_3})$. The image plane $\vec{n_i} = [0, 0, 1]$.

Occlusion is said to occur if $\vec{n_t}.\vec{n_i} < 0$ i.e. the dot product between the normal to the triangle plane and the image plane is negative.

Observations and Results

The following are the observations for the experiment:

- On the first look, it seems this method itself solves all the problems. And it does work well in normal lighting conditions for an average nose.
- This approach lacks a bit when the nose of the person is especially rounded. When the nose is rounded, the jewel is visible even when it shouldn't be.

Figure 18: The jewel is visible even when it shouldn't be

Approach 2.2: Considering the Face Horizontal Orientation

Approach 2.1.1 doesn't work with rounded noses well because rounded noses have a large curvature. Due to this, it might happen that before reaching the previous condition, a different part of nose itself might cover the part we are willing to put the jewel on.

A comparison of this can be seen below:

Figure 19: Correct Result for Normal Nose

Figure 20: Correct Result for Rounded Nose

To prevent this, we can also consider the horizontal orientation of the face while deciding if the jewel is occluded or not. The horizontal orientation can be considered by taking any two vectors which have the same z and x coordinate in the neutral face position. For this we can have the vectors like this:

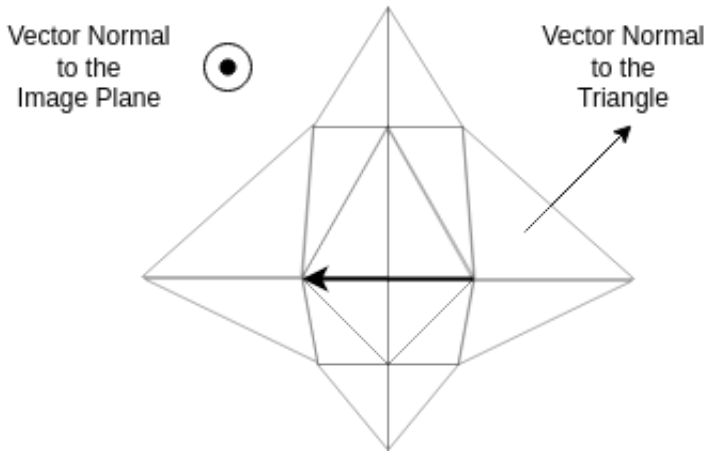

Figure 21: The horizontal vector along with the other vectors

The idea with this approach is that, once the face turns by a sufficient amount, the jewel is considered to be occluded. This method is applied along with the first method, where if either method says that the jewel is occluded, it is considered to be occluded.
Using these two vectors we can determine if the region we are interested is visible or not by taking a simple dot product. If this dot product is positive, then the angle between the vectors will be less than 90° and the region will be visible. Conversely, if this dot product is negative, then the angle between the vectors will be greater than 90° and the region will not be visible.
This setup will look like the following:

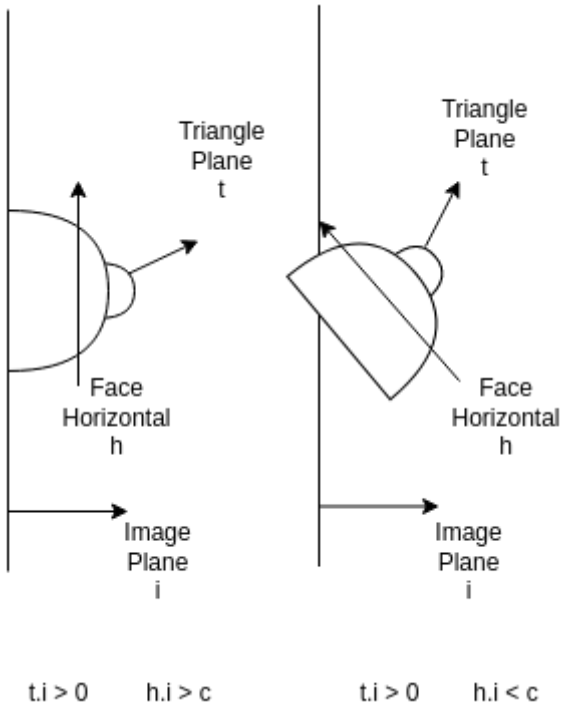

Figure 22: Top View: The Dot Product between the Triangle Plane and Image Plane is greater than 0, but Horizontal Vector makes a big angle with the image plane, which triggers occlusion

This requires finding out an angle beyond which the jewel is not visible.

Procedure and Results

For performing this task, we considered two points placed symmetrically with respect to the vertical line of the nose (220, 440). Let the vectors to this be denoted by $\vec{v_4}$ and $\vec{v_5}$. The horizontal vector to be considered then becomes $\vec{v_h} = \vec{v_4} - \vec{v_5}$.
Occlusion is said to occur if ($\vec{n_h}.\vec{n_i} < \text{c}$) or ($\vec{n_t}.\vec{n_i} < 0$). The value of c is fixed to be -0.3 in our implementation, corresponding to an angle of 107° or 17° from the neutral position (as the resting angle is 90°).

Observations

The following are the observations for the experiment:

- For all the faces we tried this method on, it seems to work well. There are some instances when the jewel appears and disappears in quick succession.
- This approach still does not consider the case when something is covering the nose of the person (like a mask or hand), but even then the landmarks are predicted and the jewel is placed.

Relevant Link

The code for the experiments can be found on GitHub.

References

[1] Matthew Turk, Alex Pentland; Eigenfaces for Recognition. J Cogn Neurosci 1991; 3 (1): 71–86. doi: https://doi.org/10.1162/jocn.1991.3.1.71

[2] Clifford Clausen, Harry Wechsler, Color image compression using PCA and backpropagation learning, Pattern Recognition, Volume 33, Issue 9, 2000, Pages 1555-1560, ISSN 0031-3203

[3] S. Gao, Y. Zhang, K. Jia, J. Lu and Y. Zhang, "Single Sample Face Recognition via Learning Deep Supervised Autoencoders," in IEEE Transactions on Information Forensics and Security, vol. 10, no. 10, pp. 2108-2118, Oct. 2015, doi: 10.1109/TIFS.2015.2446438.

[4] Stephen Balaban "Deep learning and face recognition: the state of the art", Proc. SPIE 9457, Biometric and Surveillance Technology for Human and Activity Identification XII, 94570B (15 May 2015); https://doi.org/10.1117/12.2181526

[5] Ghatak, A. (2019). Initialization of Network Parameters. In: Deep Learning with R. Springer, Singapore.

[6] Florian Schroff, Dmitry Kalenichenko, James Philbin; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815-823

[7] Going Deeper with Convolutions, Christian Szegedy and Wei Liu and Yangqing Jia and Pierre Sermanet and Scott Reed and Dragomir Anguelov and Dumitru Erhan and Vincent Vanhoucke and Andrew Rabinovich

[8] Deep Residual Learning for Image Recognition, Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, 2015

[9] FaceNet implementation using Softmax with cross entropy loss

[10] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., and Grundmann, M. (2019). BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. ArXiv./abs/1907.05047

[11] Mediapipe Documentation