Facial Recognition

Real-time Model Deployment Guidelines

Dr. Monika Singh
Senior - Data Scientist

Pushing Facial Recognition technology beyond conventional applications for real-time deployment on a large-scale will require overcoming numerous challenges to achieve high accuracy rates at minimal processing time. Mentioned below are some of the best practices to follow while Revolutionizing the Technology.

Index

1.Introduction 3

2.Approaches - Then and Now. 3

3.Face Recognition:. 3

3.1 Face Detection

3.2 Face Recognition

4.Loss Functions and Architectural Details 5

5.Model Training and Results 6

6.Testing on Real-time: Challenges and Solutions 7

7.Guidelines. 8

8.Summary 8

References 8

1. Introduction

The technology that allows computers to identify and recognize a person from digital images or video sources is popularly known as “Face Recognition.”

Facial recognition as a biometric authentication technology is widely used as it is a contactless process and the newly developed deep learning approaches are highly accurate. There are massive use-cases where face recognition technology is paving its way. The technology is used not only for solving basic issues like recognizing shop-lifters in retail stores or access control and automatic log-in on personal devices like mobile phones, but it can also aid in solving more complex real-world issues like criminal identification, missing child recognition, forensic investigation, medical diagnosis and facilitating secure online transactions.

The paper summarizes the facial recognition process, challenges in real-time testing and guidelines for deploying the model in real-time.

2. Approaches: Then and Now

Earlier approaches for facial recognition include using shallow classifiers based on some local features, which are handcrafted and aggregated into a face descriptor using a pooling method such as a **Fisher Vector**. Low-level features are captured using Scale Invariant Feature Transformation (SIFT), Histograms of the Oriented Gradients (HOG) and Local Binary Patterns (LBP), Laplacian, Gabor and Sobel filters.

Recently, face recognition technology has drifted towards deep-learning based classifiers, primarily due to its high performance, availability of pre-trained models and high scalability. The state-of-the-art deep-learning classifiers are DeepFace [1] and FaceNet [2]. FaceNet and DeepFace architectures differ with regard to learning the embedding vectors.

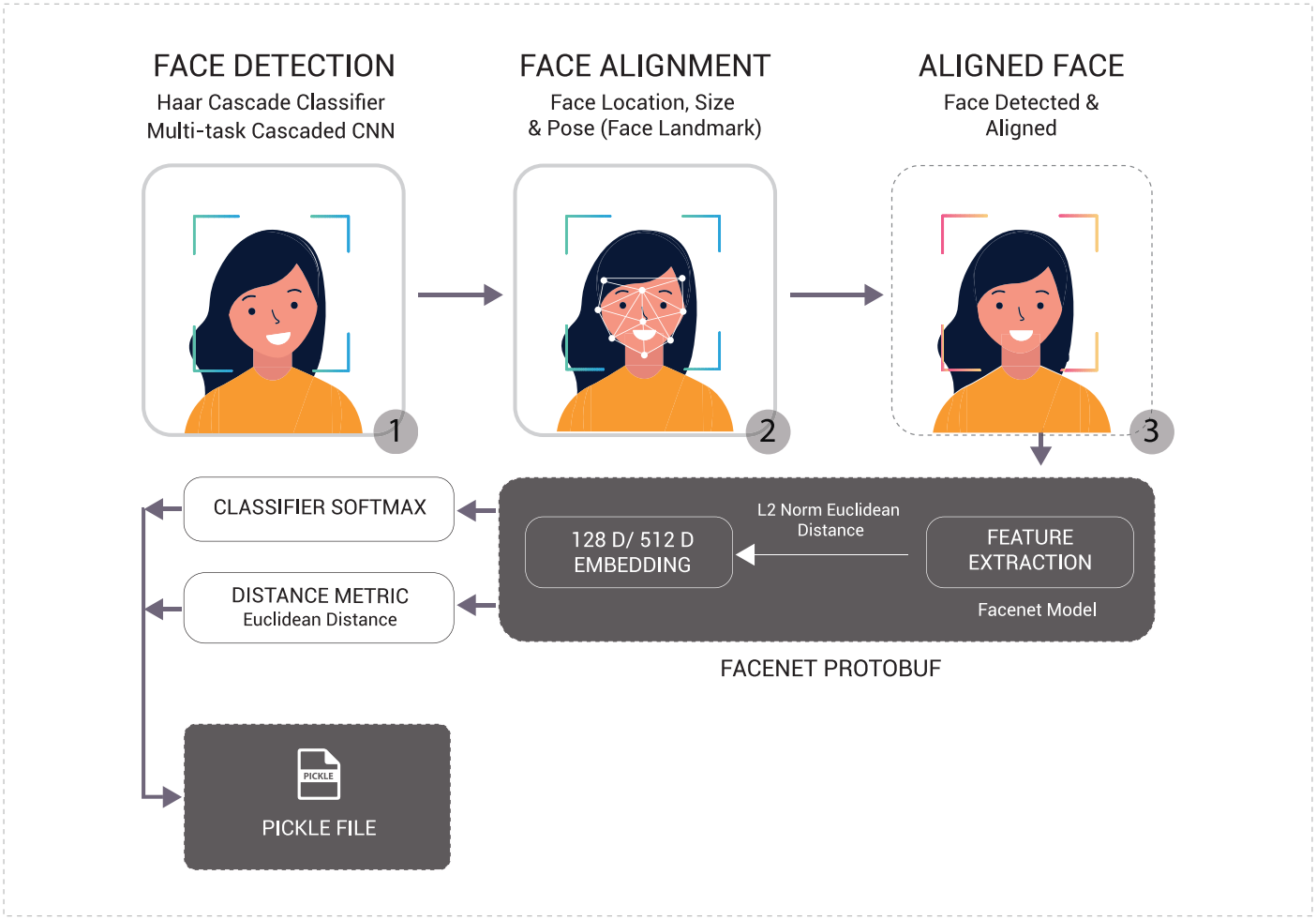
3. Face Recognition Using Facenet

There are two modules in face recognition technology, which are discussed below:

3.1 Face Detection: figures if the samples have a face in it. The detection part is done using HAAR cascade classifiers or Multi-task CNN (MTCNN), among others. Facial landmark information is used for face alignment. Facial landmarks are used to understand regions of the face such as eyes, nose, jawline, eyebrows, etc. This module enables the flexibility to detect and align faces using the correct image dimensions. While a larger crop imposes to learn unnecessary features like hair or background, a smaller crop cuts out important facial features.

3.2 Face Recognition: deals with verifying the face. Facenet can be based on some advanced deep neural network architectures, some of which are Inception-Resnet-V1, Inception-Resnet-V2, Zeiler & Fergus Neural network architectures, Mini-inception, Tiny-inception, etc.

The architecture is explained step by step in the section below. The outline of the face recognition module is presented in the figure below:



- An image is retrieved after "detecting and aligning" inputs post processing by HaarCascade classifier.
- It is then read as a 3D matrix, where the RGB channel describes each dimension. This is fed to the FaceNet model, for instance, Inception Resnet V1 architecture.
- The weights from the pre-trained models taught on VGG2Face2 and Casia-WebFace datasets are used to train images on FaceNet model.
- Inception Resnet architecture takes the input. It has 128/512 neurons (embedding layers) in its last layer which is connected to the classifier. The role of the Facenet architecture is to extract features. This will include learning facial features and assigning each feature with unique weights for corresponding images. Embeddings are nothing but features represented as numeric values in dimensional arrays.
- During real-time testing, we can use a distance metric or a classifier to classify the embeddings of the test image into a label. While classifying embeddings using a distance metric, we must find the distance between embeddings of the test image and training sets. The test image would be given a label corresponding to the most similar embedding. Similarly, if we use a classifier (**Support Vector Classifier** in this case), the goal of which is to predict the label of the given picture based on the Softmax or Triplet loss; thus, classifying similar embeddings. The loss functions are discussed in section 4.

4. Loss Functions and Architectural Details

Loss Functions

Softmax Loss:

When we provide our training datasets, the embeddings of images are obtained independently by optimizing the loss function of the classifier, which in this case will be Softmax Loss. Softmax Loss is briefly discussed below:

Minimize: $$-\frac{1}{M}\sum_{i=1}^{M}\log\frac{e^{W_{y_i}^T f(\mathbf{x}_i)+b_{y_i}}}{\sum_{j=1}^{C} e^{W_j^T f(\mathbf{x}_i)+b_j}}$$

Subject to: $$\|f(\mathbf{x}_i)\|_2 = \alpha,\ \ \forall i = 1, 2, ...M,$$

That's the overall objective of the L2-Softmax Loss, where:

- x_i is the input image within a mini-batch of size **M**
- y_i is the i^{th} target
- $f(x_i)$ is the **d** dimensional feature descriptor
- **C** is the number of classes
- **W** and **b** are the trainable weights and bias in the network

Triplet Loss

The goal of triplet loss is to minimize the distance between an anchor and a positive example meanwhile maximizing the distance between the anchor and the negative example. The positive example, in this case, will have the same label as that of the anchor. And the negative example will have a different label to that of the anchor. Learning the anchor, positive and negatives for training examples is also known as Siamese network. The distance between the anchor and positive or the anchor and negative can be minimized and maximized respectively using the squared L2 distance.

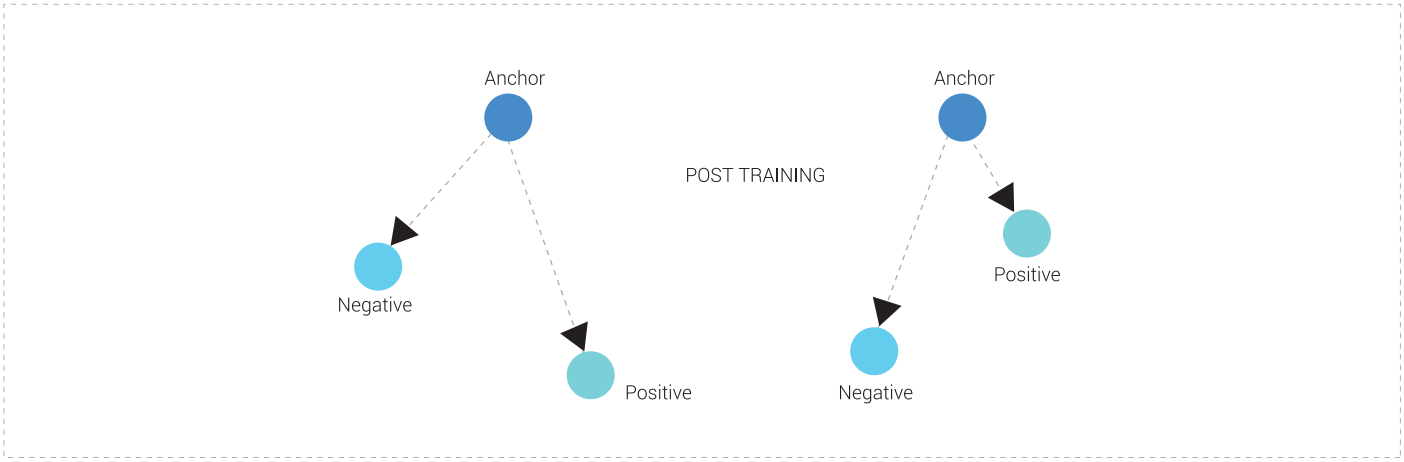
The pre-trained models are built using the **Inception Resnet** architecture, which can be optimized with triplet loss. And once the triplets are provided to the model, it can learn the optimal weights in the network, after which new images can be classified. The formula for triplet loss is as mentioned below:

Where: $$\left\|x_i^a - x_i^p\right\|_2^2 + \alpha < \left\|x_i^a - x_i^n\right\|_2^2, \forall\left(x_i^a, x_i^p, x_i^n\right) \in T$$

- x_i^a= anchor of a specific person is closer to all other images
- x_i^p= positive of the same person than it is to any image
- x_i^n= negative of any other person
- **α** is the margin that is enforced between positive and negative pairs
- **T** is the set of all possible triplets in the training set and has cardinality **N**

The loss that is being minimized is then:

$$L = \sum_{i}^{N} [\, \| f(x_i^p) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \,]$$



5. Model Training and Results

In this section, we discuss how to train with weights of a pre-trained model, validating on captured images and some results. These architectures can be trained with benchmark dataset faces, i.e. VGGFace2 and Casia-Webface. VGGFace2 comprises of approximately 9,000 unique identities for a dataset having approximately 3.3M images. And CASIA-webface comprises of 10, 575 unique identities for a dataset with a total of 4,53,453 images. Inception Resnet architecture was previously trained on VGG2Face and Casia-WebFace dataset.

We can use the weights from these pre-trained models to train our dataset, i.e. to learn the 512-byte embeddings for each picture in the training set. Once the embeddings are generated, the SVC classifier learns the label (name of the person) for the image input. The loss function used is L2-softmax loss or triplet loss. The model is trained till it provides good performance on the test set.

Once the loss function is best minimized and the model is deployed in real time, the embeddings of the test images (which in this case is a video input) are obtained. Following this, we can use distance metric/classifier (discussed in section 3) to classify. The label of the image with which the test image has the lowest squared error loss will be predicted as the label of the test image.

The training dataset comprises of approximately 110 images (actual and augmented) each for 164 individuals. To check the robustness of the face recognition approach, we experimented test images captured from the top view. The images were blurred using **Gaussian Blur** method. The performance of face recognition has an accuracy rate of over **80%** from only 5 training images (eye-level face images) per person on a validation dataset. The validation accuracies on validation sets using Casia and VGG2Face models on different number of images are as given below:

# of trained images per person	CasiaWebFace	VGG2Face
1	0%	0%
5	82.5%	86.6%
35	~94%	~96.9%
45	~95%	~97.1%
100	~97.1	~98%

There are three inferences from the above table:

- Datasets trained using VGG2Face offer better results compared against Casia WebFace
- Results improved with increase in number of training images
- Training with only one image per person a.k.a. **Low-shot** learning fails for face recognition process primarily due to light variations, pose variations, illumination differences, etc.

In summary, to get predictions in any experimental setup, ensure to train using similar experimental conditions. Better learning requires diversified images. So try to capture images under different lighting conditions and capture multiple expressions and poses.

6. Testing in Real-time: Challenges & Solutions

- A clear photograph taken from a correct camera height is required for an accurate recognition. And a fast camera is required as individuals cannot be restrained until the desired click is obtained.
- There are many changes for poses, lights, controlled settings (for validation) and real-time settings (testing). So, training with varying conditions are also necessary.

  > Solution: Memory augmentation helps in better training. **Keras** 'Imgaug' augmentation library provides a good augmentation functionality. There are different types of augmentations which can be tried, for instance, Black and White, Brightness and Contrast, Random Flipping, Gamma Adjustments, Rotation, Shear, Zoom, etc.

- Face recognition technique must overcome some challenges such as: accurately recognizing the face in an unconstrained environment with varying head rotation and tilt poses, changes in make-up, hair, expression, pose, lighting or change due to aging or sickness.

  > Solution: To build an accurate recognition system, the best approach is to train a model using pictures of people taken on different days with different expressions, poses, lighting and props.

- FaceNet is learning facial features and giving each feature weightage in the form of embeddings. If the model is trained for English faces, then it has very little knowledge about Asian faces.

  > Solution: For best results, the pre-trained model used should be trained on individuals from the same ethnicity for testing in real-time.

7. Guidelines

Following are some suggestive guidelines which can be observed to achieve better model performance and response time:

- Response time: Once the model is deployed on the server, the pictures to be recognized in real-time are to be sent to the server and the response of the model must be presented on the **User Interface (UI)**. This can be done using the **Application Programming Interface (API)** calls. The process of sending the images to the server and retrieving results post model prediction on UI can be time consuming. To reduce this time, we can:
  - Load the model (the Pickle File and the Protobuf) once. Use it for predicting new images.
  - Use **Remote Procedure Call (RPC)** to reduce the recognition time. Refer **David Sandberg's GitHub page [3]** for details.
  - Instead of saving the test images, send the byte stream straight to the server where the model is deployed. The test images are captured in the form of **Numpy Array**. You can easily serialize them and send it to the model. Not saving the test images will save a lot of time. Refer [4] for details.
- Light weight model: To use Face Recognition model on an IOS/android device, first convert the model into its light-weight counterpart. To convert the model into the **TensorFlow Lite** variant, we can simply convert the Protobuf file to a TFLite version and get the embeddings for the SVC classifier.
- Augmentation: Different augmentation techniques (refer section 5) are useful to get better performance.
- Data Collection: Train the model with varying poses, emotions, illumination and take pictures on different days.

8. Summary

Introduction to Face Recognition technology, its use cases and algorithmic developments. Further discussions revolve around FaceNet algorithm for recognizing the faces. The two steps in face recognition Face detection and Face Recognition. Network weights are adjusted such that they uniquely represent individuals. Once the embeddings are generated, the SVC classifier learns the label of each image input. The model is tuned till it provides a good performance on validation sets.

Additional discussions include training, some results on test datasets, and some challenges in using face recognition technology along with their solutions.

Lastly, how to deploy this model on a server, sending the image to the server and getting the response on the User Interface along with some challenges and solutions to reduce the response time to use this technology on a large scale.

References

[1] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1701-1708).

[2] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).

[3] https://github.com/davidsandberg/facenet/pull/776/commits/a4e67a9ac541fda81d0070921caa2cfce707f696

[4] http://satoru.rocks/2018/08/fastest-way-to-serialize-array