# Face Recognition using OpenCV

Aparna

**Dublin Business School**

This dessertation is submitted for the degree of
M.Sc., in Data Analytics

January 2020

# Declaration

I thusly proclaim that aside from where explicit reference is made to craft by others, the substance of this thesis is unique and has not been submitted in entire or to some extent for thought for some other degree or capability in this, or some other college. This exposition is my very own work and contains nothing which is the result of work done in a joint effort with others, aside from as determined in the content and Acknowledgments.

Aparna
January 2020

# Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

# Abstract

Face recognition is one of the most useful contactless biometric identification tool, having several applications in security, surveillance and human-computer interactions etc. The concept of face recognition has been familiar in the Computer Vision community since long, but little development have been achieved due to challenges in feature engineering and achieving robustness (in terms of pose, illumination, occlusion etc.). However, with the inception of deep learning, most of these problems have been mitigated. But with all this improvents, deep learning models require a large amount of data and computational power to provide an accuracte model.

In this work, we propose a face recognition pipeline based on deep learning architecture which can be trained with minimum number of samples without hours of training and can be implemented to run on videos in real time. Our method uses a pretrained deep learning framework based on the Inception module for feature generation, i.e. this module provides the feature vector for each face image. These features are used to train a Support Vector Machine (SVM) for classification. This architecture is validated on two large datasets (LFW and IMFDB). We have tested our model on videos, specifically movie clips to access the model accuracy and speed. The achieved accuracy is above 91% with approximately 4-5 frames per second which proves the superority of our model. We have also provided a real life example of face recognition using our method with very few training images, and the achieved results are very promising. Finally, we have analysed the factors affecting the end result and investigated some of our failure cases for better understanding.

# Contents

# List of Figures

*Dedicated to my…*

# Chapter 1

# Introduction

Face recognition is one of the most helpful non-intrusive biometric person identification tool that can be used today. The idea of face recognition is widely popular because it can be applied in a mobile platform for not only identification or recognition, but other important aspects such as crowd control, robotics or even as a commercial identification or marketing tool. Compared to other biometric recognition procedures such as fingerprint or iris recognition, a facial recognition system is contactless. This has several applications in security and surveillance, including automatic attendance systems, smart cities, home automation and secutrity etc. The emerging technologies of VR and human-computer interactions that are being integrated in our everyday life, depend heavily on face detection and recognition systems.

The idea that face recognition can be applied as an identification tool in everyday life poses several challenges. A structured dataset of face-only (localised and cropped faces) is required for each identity in order to successfully learn discriminative features to recognise each person. The primary challenge arises in learning a viewpoint invariant complete representation of the human face. The recognition accuracy is severely affected with the alignment of the face to the camera, specifically when instead of the frontal view, only a part of the face is visible. Recent developments in this field try to overcome this problem by employing a larger dataset for each person, extracting features of the human face from different angles. However, the availability of such dataset is not always guaranteed. In addition to that, face recogntion models still suffer from occlusion, rotation, illumination and scale variations etc.

Deep Learning techniques, especially Convolutional Neural Networks (CNN) have shown significant advantage in the field of face recognition in recent years, due to its capacity to learn discriminative features. Most works that employ deep learning algorithms train the discriminative layer over a set of known face identity images and rely on an intermediate layer to generate reprentational features that is being used for face verification. The problem with this approach is that the choice of the bottleneck layer is qualitative, and there is no direct way of accessing the discriminative capability of the chosen layer. Also, the generalizability to new faces is poor.

In this thesis, we try to overcome the aforementioned challenges by an improved algorithm for face detection and recognition in real time. The algorithm cosists of two parts: (1) A robust Face detection model and (2) A Face recognition model. The architecture of our network is designed such that it does not necessarily need face-only images for training, rather it can extract the localised faces from the full image of the person and construct training examples. The facial blob detection algorithm is implemented in this phase. This idea is extremely useful when dealing with unstructured data where face-only images are not readily available. We make use of the CNN architecture to generate a 128-dimensional embedding in a Eucledian Space and then using the Triplet Loss function to minimize the intra-class distance while maximizing inter class variability. This architecture leverages the challenges of translation, rotation and scale. Our model is able to achieve promising results in Indian Movie Face database (IMFDB), when employed across various movie scenes (videos) with less computation time per frame i.e. achieving high frame rate.

We define the significant contributions of our work as,

- We employ a facial blob detection algorithm in both phases of trainig and recognition, and this leverages the dependency on structured face-only dataset.
- Our model generates 128-dimensional facial embeddings from images to a Eucledial space, which is further optimized using a triplet loss function and introduce a novel mining strategy. A discriminative learning algorithm is implemented on top of this learned embedding space for better classification and recognition accuracy.
- The architecture is modular, lightweight and can run in real time. The implementation is possible in both supervised and unsupervised setting.

# Chapter 2

# Literature Review

Face recognition has been an important computer vision problem for a long time. Earlier works used handcrafted methods for facial feature extraction and training a classification model using the feature vectors. However, deep learning methods have surpassed the handcrafted feature based recognition models and is currently the state-of-the-art in facial recognition systems. In this work, we take cues from the Inception model by Szegedy et al. [1] that implements a novel architecture using max pooling layers, non-linear activations and local response normalization (batch normalization) layers.

The problem of face recognition have been well known in the computer vision community and researches date back to as early as 1970s, when typical clasification methods used facial features based on the human face profile. In 1990s, this area had a significant boost due to the availability of real time hardware [2]. During this time, structured datasets started coming into existence, which again helped in uniform comparison methodology. Since its inception a number of techniques have been proposed, which can broadly be classifies as **Appearance based** and **Model based**. The different methodologies and their developments are demonstrated in Figure 2.1.

## 2.1 Appearance Based Methods

These methods are focused on constructing a global image representation of the complete human face in a vector space using some form of transformation, rather than using local facial features. Most of them are based on images only, and do not use any intermediate

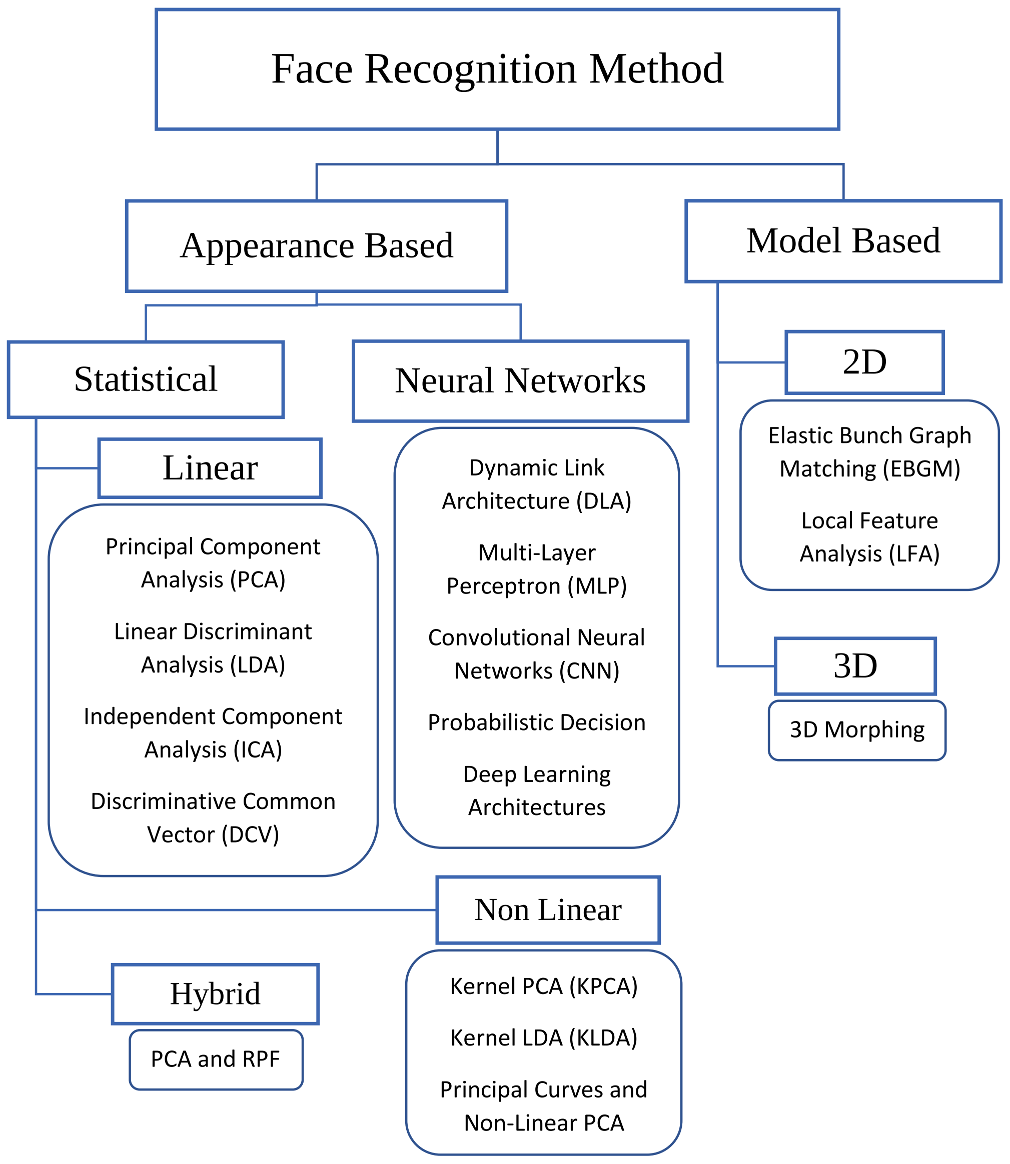
Face Recognition Method
Appearance Based
Model Based
Statistical
Neural Networks
2D
Elastic Bunch Graph Matching (EBGM)
Local Feature Analysis (LFA)
Linear
Principal Component Analysis (PCA)
Linear Discriminant Analysis (LDA)
Independent Component Analysis (ICA)
Discriminative Common Vector (DCV)
Dynamic Link Architecture (DLA)
Multi-Layer Perceptron (MLP)
Convolutional Neural Networks (CNN)
Probabilistic Decision
Deep Learning Architectures
3D
3D Morphing
Non Linear
Kernel PCA (KPCA)
Kernel LDA (KLDA)
Principal Curves and Non-Linear PCA
Hybrid
PCA and RPF

FIGURE 2.1: The literature flow and different methodologies used in face recognition methods.

modelling, i.e. features are directly derived from the image pixels. This captures the variance between faces of different identities thus identifying unique individuals. In apperance based methods, the whole 'face image' is considered as an input for face detection system to perform face recognition [3] [4]. Apperance based methods can be of different types based on the methodology used.

### 2.1.1 Statistical Methods

These methods use image transformation to generate representation to a vector space i.e. feature space from the high dimensional image space based on different statistical viewpoints. The projection coefficients are used as the feature representation of each face image through the projection of the face vector onto the basis vectors. These are further divided based on the type of projection space, i.e. **linear** or **non-linear**. The classic machine learning algorithms such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Discriminative Common Vector (DCV) are examples of linear subspace analysis. In non-linear analysis, the subspace is more complicated. The kernelised versions of PCA and LDA, as well as principal curves etc. are some non-linear analysis algorithms that are predominantly used in face recognition. The **hybrid** approach make use of both transformations at different stages of analysis. The algorithmic details are not elaborated here, however Barnouti et. al. [5] provided a detailed literature review in this regard and the mentioned algorithms can be found in his work.

The primary advantage of these methods are the lower computational cost and simplified architecture, but the proper selection of basis vectors i.e. vector space is extremely important. Also, these methods are not very efficient in dealing with large data, specifically because the dimensionality can be huge which might overshadow the computational ease.

### 2.1.2 Neural Networks

Neural Network based methods primarily make use of facial features that are extracted using statistical or model based approach. Based upon the structure i.e. layers and nodes of the network, it can have high discriminative ability to map features in a multi-dimensional manifold to capture intra-class clustering and inter-class variations. The earlier neural network architectures, such as Dynamic Link Architecture (DLA) and Multi Layer Perceptron (MLP) were able to provide high classification accuracy that leveraged the high dependence on feature engineering, although that is equally important as well.

Neural Networks grew rapidly in the last decade, and tremendous advancements are observed in the field of computer vision, specifically when dealing with large amount of image data. Convilutional Neural Networks (CNNs) gave the advantage of automatically extracting feature vectors that capture the variablity of data without any manual intervention. These methods operate using a optimization function such as Stochastic Gradient Descent (SGD) that aims to find the global minimum of the loss function manifold. The deep features can then be extracted from an intermediate layer, which can be used for matching i.e. recognition. They significantly outperform the traditional methods by a huge margin, these are discussed separately in section 2.3.

## 2.2 Model Based Methods

Model-based methods aim to reconstruct a model of the human face that capture facial variations. Prior knowledge and structure of the human face is highly utilized in designing the model. Model-based matching derives the distance and relative position features from the placement of facial elements (such as eyes, nose, lips etc.). Model-based methods can be made invariant to size, orientation, and lighting. The other benefits of these schemes are the compactness of the representation of face images and rapid matching [6] [7].

Three different extraction methods are distinguished (generic methods based on edges, lines, and curves; feature template-based methods, and structural matching methods that consider geometrical feature constraints) [4]. The major disadvantage of these methods is the difficulty of automatic feature detection. Implementation of any of these methods needs arbitary decisions on which features are important.

## 2.3 Deep Learning based Face Recognition

Deep learning methods, such as convolutional neural networks, use a cascade of multiple layers of processing units for feature extraction and transformation. They learn multiple levels of representations that correspond to different levels of abstraction. The levels form a hierarchy of concepts, showing strong invariance to the face pose, lighting, and expression changes. The initial layers automatically learn the deep features, which is equivalent to designing years of hand engineered features, such as Gabor, SIFT etc., and the later layers further learn higher level abstraction. Finally, the combination of these higher level abstraction represents facial identity with unprecedented stability.

In 2014, DeepFace [8] achieved the state-of-the-art accuracy on the famous LFW benchmark [cite], approaching human performance on the unconstrained condition for the first time by training a 9-layer model on 4 million facial images. Inspired by this work, research focus has shifted to deep-learning-based approaches, and the accuracy was dramatically boosted in just three years. Deep learning technique has reshaped the research landscape of face recognition in almost all aspects such as algorithm designs, training/testing data sets, application scenarios and even the evaluation protocols. Therefore, it is of great significance to review the breakthrough and rapid development process in recent years.

Deep Learning based methods primarily consists of two stages, (1) Deep Feature Extraction, and (2) Face Matching i.e. Recognition

- **Deep Feature Extraction:**
  Feature extraction methods generally use a backbone CNN architecture such as AlexNet [cite], VGGNet [cite], GoogleNet [1] etc. trained on a large scale image database such as ImageNet [9] as a baseline and finetune the model based on facial images. However, with increasing datasets on human faces, many novel architectures are being designed specific to face recognition. An ensemble of networks, each having specific tasks, are also used in face recognition tasks. Hu et al. [cite-86] shows that it provides an increase in performance after accumulating the results of assembled networks.

  The **Loss Function** employed in the optimization is equally important in achieving high accuracy. Traditionally, the Softmax loss is used for object classification. However, in face recognition, intra class variations might be higher than inter class differences, and softmax loss is not sufficient. Many variations of softmax is eveolved through time in face recognition, such as L2 normalization. In addition to that Eucledian distance based and Cosine distance based loss functions have also proved significant success. The novel Triplet Loss function is based on eucledian distance, which we have used in our work.

- **Face Matching:**
  After the deep networks are trained with massive data and an appropriate loss function, each of the test images is passed through the network to obtain a deep feature representation. Once the deep features are extracted, most methods directly calculate the similarity between two features using cosine distance or L2 distance;

> then, the nearest neighbor (NN) and threshold comparison are used for both identification and verification tasks. In addition to these, other methods are introduced to postprocess the deep features and perform the face matching efficiently and accurately, such as metric learning, sparse representation based classifier (SRC), and so forth.

Earlier deep learning based face recognition approaches used an SVM like classification model [10] [8] on top of the deep feature representations taken from an intermediate bottleneck layer as a measure of generalization to leverage the dependency on the limited training identities. However, this approach is indirect an inefficient in the sense that there is no proven way of knowing that the bottleneck representations are well generalized beyond the training identities and also the feature sizes from the bottleneck layers are usually very large. These works [10][8][11] employ a multi-stage approach that uses PCA for dimensionality reduction on top of the deep CNN output, and then training an SVM over the transformed feature set. The work of Taigman et al. [8] proposes a general 3D shape alignment method. This multi stage approach uses a multi-class deep neural network trained on an ensemble of multiple face dataset. The authors experimented with a Siamese Network trained using a L1-distance between two classes. In a similar approach Zhu et al. [11] uses a deep network to "warp" faces into a canonical frontal view and then use a CNN classification network for identification. For verification, authors use an ensemble of SVMs over the PCA transformed features from the network. Sun et al. [10] use an ensemble of small networks for face identification. 25 of these networks operate on different face patches. This idea gives significant improvement in terms of computational efficiency. The authors use both PCA and a Joint Bayesian Model over 50 regular and flipped responses and uses linear transformation in the embedding space. Wang et al. [12] uses a similar loss to the one used here for ranking images by semantic and visual similarity.

Szegedy et al. [1] proposed the GoogleNet Architecture based on the 'Inception' module, which won the ImageNet Challenge [9] in 2014. CNNs have shown accuracy improvements when using a deeper architecture, but simply stacking layers one after another gives rise to computational problems such as Overfitting and Vanishing Gradient during backpropagation. The Inception module makes use of $1 \times 1$ convolutional layers in front of the $3 \times 3$ and $5 \times 5$ convolutional layers in parallel to reduce the dimensionality and then use filter concatenation. This increases sparsity constraints that overcome the problems of traditional deep networks. This implementation is effective in reducing model parameters while giving benefits of the very deep architectures, and this is the main reason we have used the Inception module extensively in our model architecture.

# Chapter 3

# Methodology

The proposed face recognition pipeline runs through four primary steps:

- Detection of faces from images
- Generating face embeddings from the detected faces
- Training a classification model based on the face embeddings
- Prediction/Matching of any unknown face image

In this section, we will go through the basic building blocks of our model, an explain in detail the theory behind each component.

## 3.1 Face Detection

### 3.1.1 Image Preprocessing

The input image is processed using three methods: (1) Image resizing, (2) Mean substraction and (3) Scaling. This form of preprocessing provides additional advantage when detection mechanisms are applied.

- **Image Resizing:**
  The inout images are of varying dimensions. However, the face detection methodology expects a fixed domension input. The image size is decresed to a fixed lower

resolution for faster processing by the detection mechanism. Bilinear interpolation for image rezizing is used by default.

- **Mean subtraction and scaling:**
  The image is normalized using mean subtraction and scaling. We use the channel-wise mean ($\mu_R$, $\mu_G$, $\mu_B$) of the Imagenet [9] dataset and apply normalization in every channel using a scale factor $\sigma$.

  $$R' = (R - \mu_R)/\sigma_R, \qquad G' = (G - \mu_G)/\sigma_G, \qquad B' = (B - \mu_B)/\sigma_B \tag{3.1}$$

  where $R'$, $G'$, $B'$ are the processed RGB channels of the input image.

### 3.1.2 Detection methodology

A pretrained deep learning model is used off-the-shelf for face detection from the processed image. We are using a Caffe based object detector developed using the ResNet [cite] architecture. This model is finetuned using face images to localize faces in an image and is available open-source. We have not applied any additional training for face detection. The face detector model gives the confidence (probailities) and coordinates of each face detected. Detectiond above a certain confidence threshold is cropped as '*face images*' and approved as training samples for our deep learning architecture.

We assume that each image contains a single face, which is fairly practical as that kind of dataset can be readily constructed from any person, specifically using social media images. The details of the face detection model is not discussed here, which is attached in the appendix.

## 3.2 Model Architecture

### 3.2.1 Convolutional Neural Networks

Our face recognition model is developed on Convolutional Neural Networks (CNNs), that have revolutionised the world of computer vision. A CNN has four main components: (1)

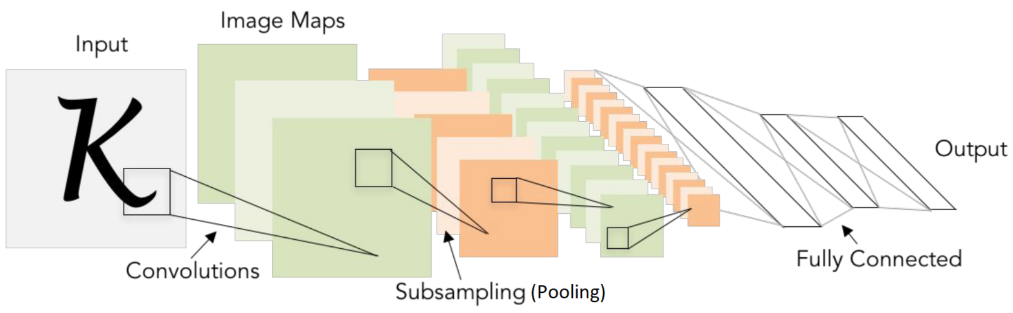

FIGURE 3.1: An example of convolutional neural network architecture

Convolution layer, (2) Activation function, (3) Pooling layer and (4) Classification layer (Fully connected network).

#### 3.2.1.1 Convolution layer

CNNs derive their name from the "convolution" operator. The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

A filter slides over the input image (convolution operation) to produce a feature map. The Convolution operation captures the local dependencies in the original image. Different filters generate different feature maps from the same original image. In practice, a CNN learns the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process). The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

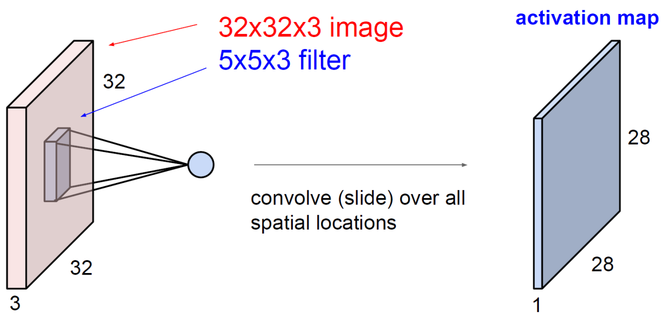

FIGURE 3.2: The convolution operation

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

**Depth**: Depth corresponds to the number of filters used for the convolution operation.

**Stride**: Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.

**Zero-padding**: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps.

### 3.2.1.2 Activation function

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not.

> "The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input."

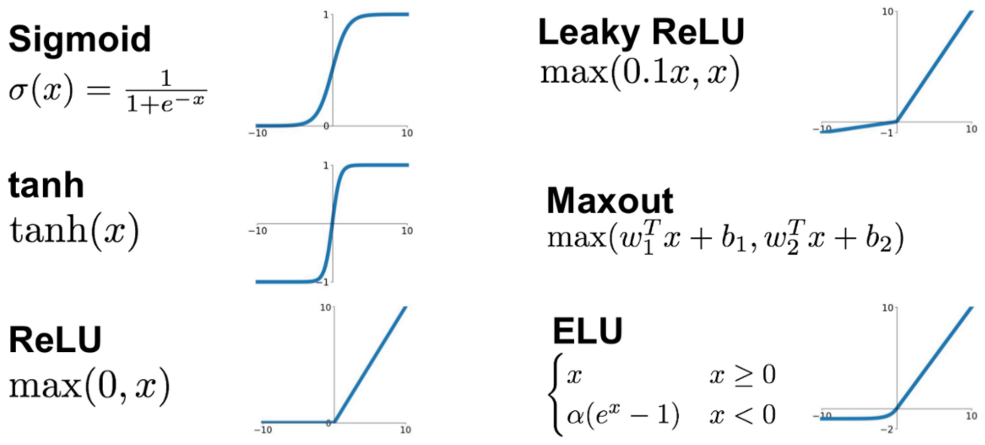

Figure 3.3: Different Activation Functions

We have different types of activation functions just as in Figure above, but for this post, my focus will be on Rectified Linear Unit (ReLU).

ReLU function is the most widely used activation function in neural networks today. One of the greatest advantage ReLU has over other activation functions is that it does not activate all neurons at the same time. From the image for ReLU function above, we'll notice that it converts all negative inputs to zero and the neuron does not get activated. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

#### 3.2.1.3 Pooling layer

Spatial Pooling, also called subsampling or downsampling, reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In case of Max Pooling, we define a spatial neighborhood (for example, a $2 \times 2$ window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

The function of Pooling is to progressively reduce the spatial size of the input representation.

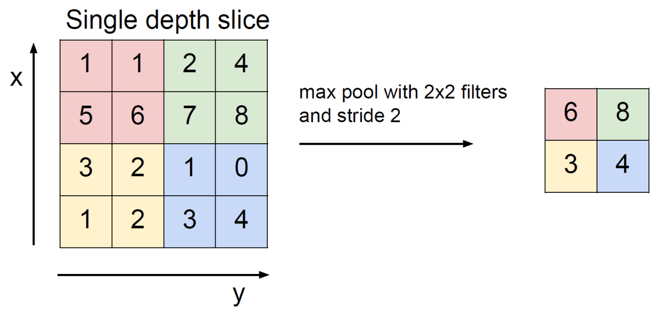

Figure 3.4: An example of pooling operation

- pooling makes the input representations (feature dimension) smaller and more manageable reduces the number of parameters and computations in the network, therefore, controlling overfitting.

- makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
- helps us arrive at an almost scale invariant representation of our image (the exact term is "equivariant").
- This is very powerful since we can detect objects in an image no matter where they are located.

#### 3.2.1.4 Classification layer (Fully Connected Network)

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a softmax activation function in the output layer. The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. For example, in image classification task it has four possible outputs, and the fully connected network should predict the highest probability for the correct class.

Apart from classification, a fully-connected layer also learns non-linear combinations of convolutional features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

#### 3.2.1.5 Training and Backpropagation

As discussed above, the Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier. The overall training process of the Convolution Network may be summarized as below:

- **Step 1**: We initialize all filters and parameters/weights with random values.

- **Step 2**: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class. Since weights are randomly assigned for the first training example, output probabilities are also random.

- **Step 3**: Calculate the total error at the output layer (summation over all classes)

$$Total\ Error = \sum_{i=1}^{c}(\ target\ probability - output\ probability\ )^2$$

- **Step 4**: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values/weights and parameter values to minimize the output error.
  - The weights are adjusted in proportion to their contribution to the total error.
  - When the same image is input again, output probabilities ahould be closer to the target vector.
  - This means that the network has learnt to classify this particular image correctly by adjusting its weights/filters such that the output error is reduced.
  - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

- **Step 5**: Repeat steps 2-4 with all images in the training set.

The above steps train the ConvNet – this essentially means that all the weights and parameters of the ConvNet have now been optimized to correctly classify images from the training set.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories. Note that the steps above have been oversimplified and mathematical details have been avoided to provide intuition into the training process.

### 3.2.2 Inception Module

Inception means going deeper. One can just make a deeper network, i.e. by increasing number of layers, or the number of neurons in each layer. But this often creates complications:

- Bigger the model, more prone it is to overfitting. This is particularly noticeable when the training data is small.
- Increasing the number of parameters means you need to increase your existing computational resources

A solution for this, as the paper suggests, is to move on to sparsely connected network architectures which will replace fully connected network architectures, especially inside convolutional layers. This can be done by parallel connection of multiple layers having different filters, and then finally concatenating all of those parallel paths to pass to next layers.

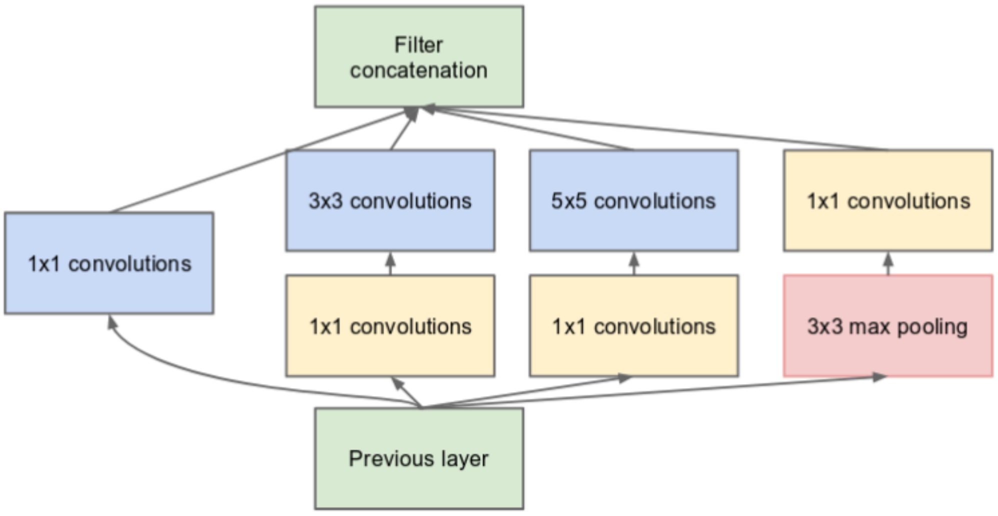

FIGURE 3.5: Inception module with dimension reductions

"(Inception Layer) is a combination of all those layers (namely, 1x1 Convolutional layer, 3x3 Convolutional layer, 5x5 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage."

Along with the above-mentioned layers, there are two major add-ons in the original inception layer:

- $1 \times 1$ Convolutional layer before applying another layer, which is mainly used for dimensionality reduction
- A parallel Max Pooling layer, which provides another option to the inception layer.

InceptionNets are preferable as they are not just deeper, but also wider and we can stack many such layers and still the output params to be trained are less when compared to all other architectures.

### 3.2.3 Triplet Loss

Triplet loss is extensively used for learning discriminative features by decreasing the feature distance of intra-class samples and pushing (increasing) the distance of the inter-class samples.

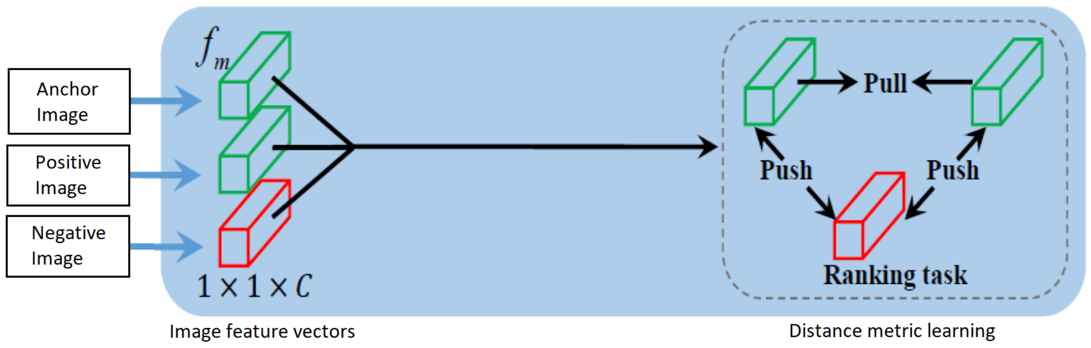

FIGURE 3.6: Triplet learning

The loss is defined over triplets of image embeddings:

- an anchor or base image
- a positive of the same class as the anchor
- a negative of a different class

For some distance on the embedding space $d$ the loss of a triplet $(a, p, n)$ is:

$$\mathcal{L} = max(d(a,p) - d(a,n) + m, 0) \tag{3.2}$$

where $d(x, y)$ denotes the feature distance (Eucledian) between two samples $x$ and $y$. We minimize this loss, which pushes $d(a, p)$ to 0 and $d(a, n)$ to be greater than $d(a, p) + m$, where $m$ is the margin.

#### 3.2.3.1 Triplet Mining Strategy

Based on the definition of the loss, we want to extract **hard triplets:** triplets where the negative is closer to the anchor than the positive, i.e. $d(a, n) < d(a, p)$

Typically, a batch of size $B = PK$ is chosen, composed of $P$ different persons with $K$ images each. A typical value is $K = 4$. The two strategies are:

- **Batch All:** select all the valid triplets, and average the loss on the hard and semi-hard triplets. We discard the easy triplets (those with loss 0), producing a total of $PK(K - 1)(PK - K)$ triplets ($PK$ anchors, $K - 1$ possible positives per anchor, $PK - K$ possible negatives).
- **Batch Hard:** for each anchor, select the hardest positive (biggest distance $d(a, p)$) and the hardest negative among the batch. This produces $PK$ triplets, and the selected triplets are the hardest among the batch.

The batch-hard mining strategy provides better discriminative ability since it selects the hardest samples from the batch. In our implementation the batch-hard mining strategy is used for finetuning the CNN architecture in a transfer learning setting.

### 3.2.4 Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in an $N$-dimensional space that distinctly classifies the data points. More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high or infinite-dimensional space, which can be used for classification, regression, or outliers detection etc. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

We will demonstrate the theory of Support Vector Machines (SVM) using the concept of Linear SVM using two class classification problem, which can be extended to multi-class classification. In our implementation, the scikit-learn model of SVM is used which can be modified for the number of classes and the number of data points.

#### 3.2.4.1 Linear SVM

Given a set of $n$ data points of the form $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$, where $\vec{x}_i$ is a $p$-dimensional vector and $y_i$ denotes its class label, either $+1$ or $-1$. We want to find a "maximum margin hyperplane" that divides the group of $\vec{x}_i$ with class $y_i = +1$ from the group of $y_i = +1$ so that the distance between the hyperplane and the nearest point $\vec{x}_i$ from either group is maximized.

Any hyperplane can be written as the set of points $\vec{x}$ satisfying

$$\vec{w} \cdot \vec{x} - b = 0$$

where $\vec{w}$ is the (not necessarily normalized) normal vector to the hyperplane. This is much like Hesse normal form, except that $\vec{w}$ is not necessarily a unit vector. The parameter $\frac{b}{\|\vec{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{w}$.

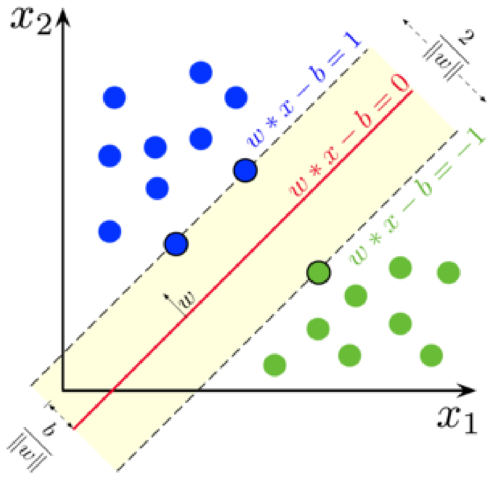

FIGURE 3.7: Maximum margin hyperplane and the margins for a 2-class classification problem

To extend SVM to cases in which the data are not linearly separable, the hinge loss function is introduced

$$\max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)$$

This function is zero if if $\vec{x}_i$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.

We then wish to minimize

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$$

where the parameter $\lambda$ determines the trade-off between increasing the margin size and ensuring that the $\vec{x}_i$ lie on the correct side of the margin. Thus, for sufficiently small values of $\lambda$, the second term in the loss function will become negligible, hence, it will behave if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

# Chapter 4

# Implementation

## 4.1 Architecture of the Deep Learning Model

The detailed architecture is depicted in the Figure 4.1. The two major differences are the use of $L_2$ pooling instead of max pooling $(m)$, where specified. Instead of taking the spatial max, the $L_2$ norm is computed in this case. The pooling is always $3 \times 3$ (aside from the final average pooling) and in parallel to the convolutional modules inside each Inception module. If there is a dimensionality reduction after the pooling it is denoted with $p$. $1 \times 1$, $3 \times 3$, and $5 \times 5$ pooling are then concatenated to get the final output. Our

| **type** | **output size** | **depth** | #1×1 | #3×3 **reduce** | #3×3 | #5×5 **reduce** | #5×5 | **pool proj (p)** | **params** |
|---|---|---|---|---|---|---|---|---|---|
| conv1 (7×7×3, 2) | 112×112×64 | 1 | | | | | | | 9K |
| max pool + norm | 56×56×64 | 0 | | | | | | m 3×3, 2 | |
| inception (2) | 56×56×192 | 2 | | 64 | 192 | | | | 115K |
| norm + max pool | 28×28×192 | 0 | | | | | | m 3×3, 2 | |
| inception (3a) | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | m, 32p | 164K |
| inception (3b) | 28×28×320 | 2 | 64 | 96 | 128 | 32 | 64 | $L_2$, 64p | 228K |
| inception (3c) | 14×14×640 | 2 | 0 | 128 | 256,2 | 32 | 64,2 | m 3×3,2 | 398K |
| inception (4a) | 14×14×640 | 2 | 256 | 96 | 192 | 32 | 64 | $L_2$, 128p | 545K |
| inception (4b) | 14×14×640 | 2 | 224 | 112 | 224 | 32 | 64 | $L_2$, 128p | 595K |
| inception (4c) | 14×14×640 | 2 | 192 | 128 | 256 | 32 | 64 | $L_2$, 128p | 654K |
| inception (4d) | 14×14×640 | 2 | 160 | 144 | 288 | 32 | 64 | $L_2$, 128p | 722K |
| inception (4e) | 7×7×1024 | 2 | 0 | 160 | 256,2 | 64 | 128,2 | m 3×3,2 | 717K |
| inception (5a) | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | $L_2$, 128p | 1.6M |
| inception (5b) | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | m, 128p | 1.6M |
| avg pool | 1×1×1024 | 0 | | | | | | | |
| fully conn | 1×1×128 | 1 | | | | | | | 131K |
| L2 normalization | 1×1×128 | 0 | | | | | | | |
| total | | | | | | | | | 7.5M |

FIGURE 4.1: Implementation details of the Deep Learning model

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, as depicted in Figure 4.2.

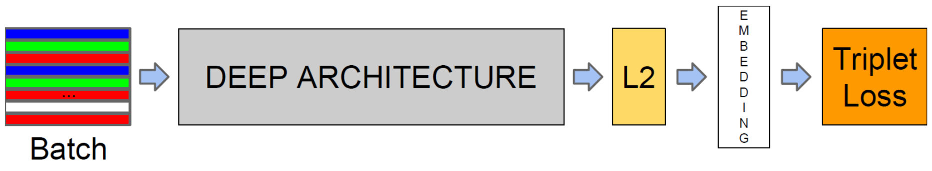

FIGURE 4.2: Model structure

## 4.2 Training Scheme

We have used OpenCV as the platform for the deep learning architecture to generate face embeddings. The OpenCV implementation does not require further training on faces, rather the face embeddings can be readily generated using the model. After the face embeddings are generated for each sample, a Support Vector machine is trained to further imporve the classification accuracy. Since the evaluation is done on known faces of the dataset, this classification model can be used. The training philosophy is demonstrated in Figure 4.3.

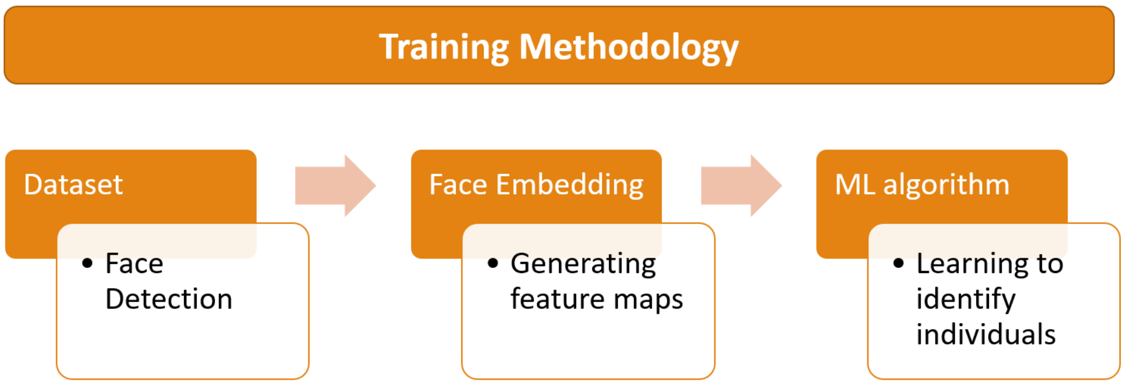

FIGURE 4.3: The training philosophy of our model

### 4.2.1 Verification

During verification, any random image or image frame from the video (image sequence) can be given to the model. The verification procedure is as follows:

- The face detection framework localises the available faces in the given input image/frame
- The localised faces are cropped and resized to input dimension of the CNN
- The face embedding is produced uding the deep learning model
- The SVM model takes this face embedding and provides the classification score (probability) for each class. This hight probability class is the predicted label of the face.

In addition to this, we draw a bounding box around the face and write the class label with confidence score in the output image.

## 4.3 System Requirements

DUring training and testing, we have used **Python 3.7** as the programming language in the **Anaconda** environment. We have also used **OpenCV** (cv2) as the major computer vision library for this project. The other packages that are being used in this work are, • numpy, • argparse, • imutils, • pickle, • os, and the machine learning library • scikit-learn (sklearn).

Since the model is lightweight, it can be run without GPU. Our system consists of Intel core i5 processor with 8 GB of RAM and the face embedding generation took approximately 0.2-0.3 seconds per image. The SVM model training is based on the number of samples, but for 4 classes with 100 images per class it took around 3-4 seconds.

# Chapter 5

# Results and Discussion

The proposed face recognition model is verified in two large datasets, and our model has given significant results. It is tested on both images and videos, with real life examples. We have also analyzed the shortcomings of our model and the effect of various parameters affecting the final accuracy.

## 5.1 Datasets

We have primarily used the Indian Movie Face database (IMFDB) for training our network, and then evaluation the performance (recognition accuracy) on different movie clips. However, for the sake of completeness, we have also evaluated our model in Labelled Faces in the Wild (LFW) dataset.

### 5.1.1 Indian Movie Face database (IMFDB)

We use the Indian Movie Face database (IMFDB) which is a large unconstrained face database consisting of 34512 images of 100 Indian actors collected from more than 100 videos. All the images are manually selected and cropped from the video frames resulting in a high degree of variability and diversity. This is the major testing platform for the validation and testing of our model.

FIGURE 5.1: An instance from the IMFDB dataset

### 5.1.2 Labelled Faces in the Wild (LFW)

Labeled Faces in the Wild (LFW) is a database of face photographs designed for studying the problem of unconstrained face recognition. A total of 13,233 images of 5,749 people were detected and centered by the Viola Jones face detector and collected from the web. 1,680 of the people pictured have two or more distinct photos in the dataset. This split of 1,680 person's images are used in training the deep learning model.

## 5.2 Evaluation Metric

we evaluate our method on the face verification task, i.e. given a pair of two face images a squared L2 distance threshold $D(x_i, x_j)$ is used to determine the classification of same and different. All faces pairs $(i, j)$ of the same identity are denoted with $\mathcal{P}_{same}$, whereas all pairs of different identities are denoted with $\mathcal{P}_{diff}$.

We define the set of all true accepts as

$$TA(d) = \{f(i, j) \in \mathcal{P}_{same}, \text{ with } D(x_i, x_j) \leq d\} : \tag{5.1}$$

These are the face pairs $(i, j)$ that were correctly classified as same at threshold $d$. Similarly, the set of all pairs that was incorrectly classified (false accept) is defined as,

$$FA(d) = \{f(i, j) \in P_{diff}, \text{ with } D(x_i, x_j) \leq d\} \tag{5.2}$$

The validation rate $VAL(d)$ and the false accept rate $FAR(d)$ for a given face distance $d$ are then defined as,

$$VAL(d) = \frac{|TA(d)|}{|\mathcal{P}_{same}|}, \qquad FAR(d) = \frac{|FA(d)|}{|\mathcal{P}_{diff}|} \tag{5.3}$$

## 5.3 Performance in Different Datasets

### 5.3.1 Perfomance in LFW

In the Labelled Faces in the Wild (LFW), our dataset has provided a classification accuracy of 93.59%. This is a significant improvement considering a simple and lightweight model. In general, a CNN architecture takes huge resource and computational cost, but our model can provide that level of accuracy with very less data.

### 5.3.2 Perfomance in IMFDB

The IMFDB dataset is the more significant dataset where we have given our conribution. A real life implementation very likely resembles the video clip of the movies. Hence the dataset extracted from these movies will definitely provide a better ground for testing. Our model provides an accuracy of 91.87% on the IMFDB dataset, which essentially proves that in real life scenario also this model will provide very good results.

## 5.4 Results on video clips

We have tested our results on different movie clips during the testing phase. The testing methodology on videos is stated as folows:

- The video is analyzed frame-by-frame. Each frame is extracted and then subsequent algorithms are run.
- The face detection algorithm is applied in each frame. The face detection algorithm localizes and extracts the number of faces ("face images") present in that frame.

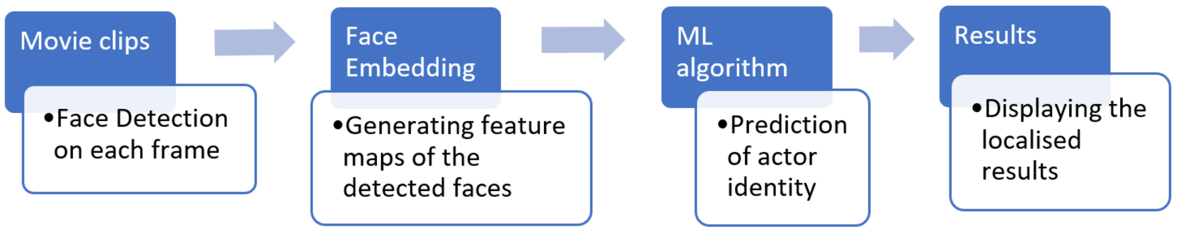
Movie clips
•Face Detection on each frame
Face Embedding
•Generating feature maps of the detected faces
ML algorithm
•Prediction of actor identity
Results
•Displaying the localised results

FIGURE 5.2: The testing methodology on videos

- The face embedding is generated for each face image. The resultant embedding is a 128-d vector i.e. a vector of length 128. This embedding is the face feature extracted using the deep learning model.
- These features are forwarded through the classification algorithm (ML algorithm), in this case it is a Support Vector Machine (SVM) that predicts the class label for each feature vector, i.e. probability of that facial feature belonging to which actor. The class (actor) with the maximum assigned probability is taken as the final prediction for that face.
- Once the prediction is complete, we superimpose the detected bounding box along with the actor name on the image frame and display it using the OpenCV framework. The confidence (probability) score is also displayed over the image.

An example where our algorithm is tested on a video clip is also provided. As a demonstration here, we have run our algorithm as described on the infamous *'3-idiots'* movie. The screenshots are provided in Figure 5.3. As we can see, the actor Aamir Khan is correctly identified with a probability of 94.53%. In the same clip, the actress Kareena Kapoor is also identified correctly with a probability of 93.53%. The resultant video clip has the speed of approximately 3-4 frames per second (fps) which will increase based on the system performance. From these results, the superiority of our algorithm is proved.

## 5.5 Real life example

put the real life example here

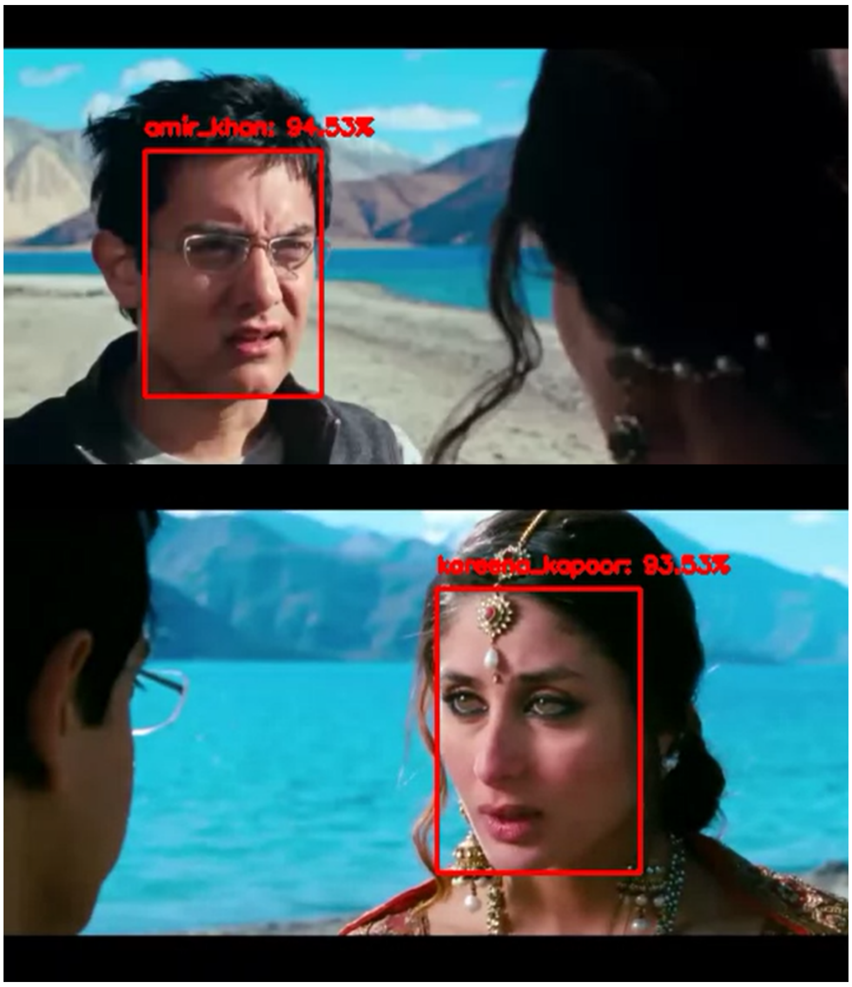

FIGURE 5.3: The results of our Face Recognition Algorithm on a movie scene.

## 5.6 Detailed Analysis

We investigate the parameters that have an effect in the final recognition accuracy. The most important factor has come out to be the quality of the dataset. Here, we analysed two factors: the number of images per person, and the resolution of images. Having more number of good resolution images per person indeed increases the accuracy figure, but that also increses computational cost and the avaiability is not always guaranteed. This variability is captured in our anaysis.

### 5.6.1 Effect of increasing training data

The recognition accuracy increases with an increase of training samples per person. With more data, different viewpoints of the human face is explored for feature extraction which lead to a more robust face embedding generation. For a training sample of 20 per person, accuracy of 87% can be achieved. However, this increase saturates with very high number of samples (Figure 5.4). Images used in this alalysis are of the fixed resolution $(200 \times 200)$.

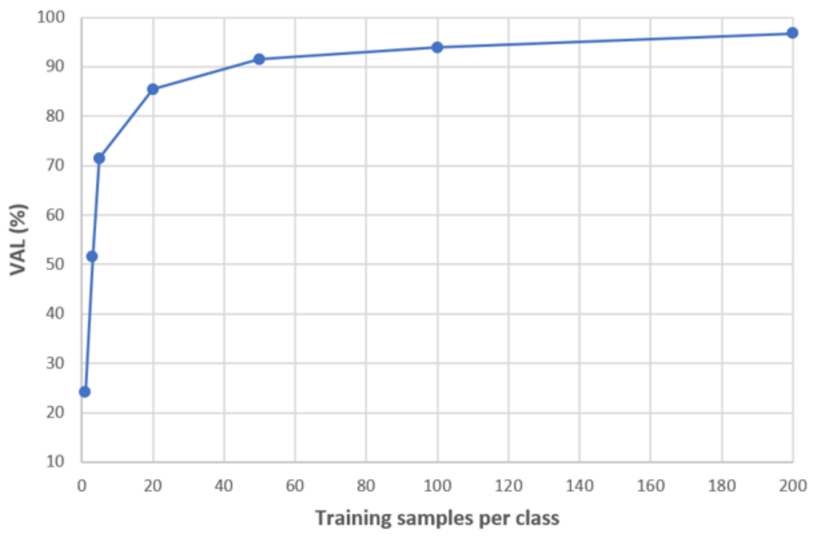

FIGURE 5.4: Effect of increasing training data on recognition accuracy.

### 5.6.2 Effect of image quality

In order to access how the image quality affects the recognition accuracy, a subest of the dataset is taken (with fixed number of images, 20 per person) and the accuracy of the trained model is evaluated on these images. As seen in Figure 5.5, as the the resolution of the face image incresases, the recognition accuracy also gets boosted significantly. This behaviour is mostly due to the fact that with increasing resolution, more pixel information is available and thereafter more detailed feature can be extracted, which contributed to better performance. However, this improvement also saturates at the end.

## 5.7 Failure cases

We have also analysed the failure cases of our model in two specific categories, (1) mis-detection of faces and (2) recognition errors.

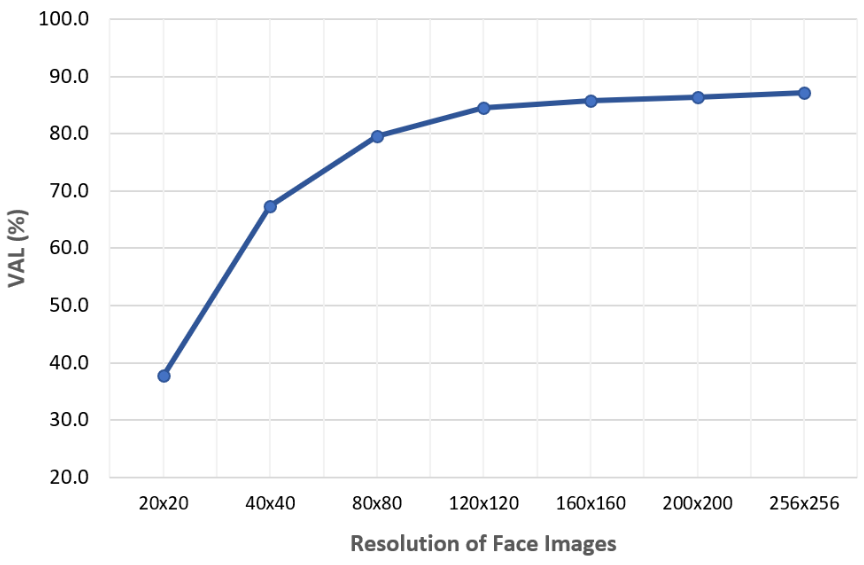

FIGURE 5.5: Effect of incresing image resolution on recognition accuracy

### 5.7.1 Misdetection of faces

Our face recognition model applies a face detector to extract the 'face images' from the given image. This detection is also applied on each image of the dataset, even though the dataset claims to have extracted the localised faces. Through this procedure, we ensure that the bad detections of the dataset does not affect our model. For the givan example of the movie '3 idiots', Figure ... gives a set of undetected faces that are present in the dataset but rejected by our face detection algorithm.

### 5.7.2 Recognition error cases

Although our model performs extremely good in most cases where relevant training data is available, there are some pitfalls. Figure ... describes some of the failure cases where the model is failed to identify the face of the intended person.

One main reason for this behavious is in built in the core machine learning ideology. Any new datapoint is classified in the given (fixed) number of training classes in any machine learning algorithm. In our case also, any person outside the training class was identified as one of the actors in the dataset. This is the major reasons for recognition errors.

One possible solution of this problem is to assign a confidence threshold. Any prediction with probability less than the threshold $T_p$ can be identified as an unknown identity.

However, the choice of threshold is extremely crucial as the training identities are in hundreds. The feature matching method can come in handy in this case.

# Chapter 6

# Conclusion

write the conclusion and future work here

## 6.1 Future Work

# Bibliography

[1] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[2] Rama Chellappa, Charles L Wilson, Saad Sirohey, et al. Human and machine recognition of faces: A survey. *Proceedings of the IEEE*, 83(5):705–740, 1995.

[3] Deepali H Shah, Dr JS Shah, and Dr Tejas V Shah. The exploration of face recognition techniques. *International Journal of Application or Innovation in Engg. And Management (IJAIEM) Web Site: www. ijaiem. org Email: editor@ ijaiem. org, editorijaiem@ gmail. com*, 3(2), 2014.

[4] Divyarajsinh N Parmar and Brijesh B Mehta. Face recognition methods & applications. *arXiv preprint arXiv:1403.0485*, 2014.

[5] Nawaf Hazim Barnouti, Sinan Sameer Mahmood Al-Dabbagh, and Wael Esam Matti. Face recognition: a literature review. *International Journal of Applied Information Systems (IJAIS)–ISSN*, pages 2249–0868, 2016.

[6] Uma Shankar Kurmi, Dheeraj Agrawal, and RK Baghel. Study of different face recognition algorithms and challenges. *IJER, Volume*, 3:112–115, 2014.

[7] Rabia Jafri and Hamid R Arabnia. A survey of face recognition techniques. *Jips*, 5 (2):41–68, 2009.

[8] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115 (3):211–252, 2015.

[10] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2892–2900, 2015.

[11] Zhenyao Zhu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Recover canonical-view faces in the wild with deep neural networks. *arXiv preprint arXiv:1404.3543*, 2014.

[12] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.