Comparative Analysis on Face Identification-based Modeling

Kwanit Gupta
B19EE046
gupta.45@iitj.ac.in

***Abstract*— The study centered on investigating various face-recognition pipelines (whether driven by traditional ML techniques or with SOTA CNNs). The project examined existing well-known implementations, such as utilizing cosface with densenet, as well as the use of arcface with resnet, etc. Moreover, the ML-DL approaches with different face-recognition streamlined losses were investigated. Using the Multi-Class ROC, class-wise accuracies, and traditional evaluation measures, many possibilities and inferences were found and understood. This study provides significant insight into the history of modeling types, their advantages, and limits, as well as their applicability in diverse settings and datasets (LFW and IMFDB). The Github Codebase is available at https://github.com/kwanit1142/Comparative-Analysis-on-Face-Identification-based-Modeling and Colab Notebook is also attached hereby.**

***Index Terms*— Multi-Class ROC, LFW, IMFDB, CNNs, Resnet, Densenet, Arcface, Cosface**

I. INTRODUCTION

Face recognition is a job that has garnered a great amount of attention in the field of computer vision due to the breadth of its potential applications, which include surveillance and identification systems, personal identity, and online social networking. Recognising human faces in a variety of stances, with a variety of emotions, and under a variety of lighting conditions is a difficult undertaking.

Face recognition is an umbrella word that refers to numerous subtasks, such as face detection, face alignment, feature extraction, and face identification. These subtasks are all included in face recognition. Because the effectiveness of face recognition systems is highly dependent on the precision of these subtasks, finding a solution to this challenge can be difficult.

Convolutional neural networks (CNNs) have streamlined loss functions such as softmax, contrastive loss, triplet loss, and centre loss. Over the years, there has been a shift away from older techniques such as Eigenfaces, which use principal component analysis for feature extraction, and towards more modern and robust techniques such as CNNs. Eigenfaces use principal component analysis for feature extraction. CNNs are now regarded the state-of-the-art face recognition technology because to their impressive performance in a variety of face identification benchmarks.

Nevertheless, despite the substantial advancements that have been made in the field of facial recognition, a number of obstacles and limits still remain. One of these restrictions is the demand for a significant amount of training data, which is particularly problematic for deep learning models. It is difficult to generalise across diverse contexts because face recognition algorithms can be influenced by changes in look caused by factors such as ageing, cosmetics, and facial hair. This makes it difficult to identify individuals based just on their appearance.

As a result, it is vital to do a comparison study of various face identification-based modelling approaches in order to gain an understanding of their benefits, drawbacks, and constraints. An investigation of this kind can provide light on methods that are now considered to be state-of-the-art and direct further research towards the development of facial recognition systems that are more reliable and accurate.

Contributions:

- Kwanit: Basically everything, from writing codebases and experimentations to making report and presentation video (repository link too).
- ChatGPT: Debugging of Codes in between (in case of version control issues or absence of any functionality in a library)

II. RELATED WORK

Face recognition is the process of identifying or verifying the identity of a person by analyzing their facial features. It is a critical task in many applications, including security systems, access control, surveillance, and human-computer interaction. In recent years, the development of computer vision and deep learning techniques has revolutionized the field of face recognition.

A. Traditional methods of Face Recognition

The earliest approaches to face recognition were based on statistical models such as Eigenfaces, Fisherfaces, and Local Binary Patterns (LBP). Eigenfaces, introduced by Turk and Pentland in 1991, was one of the earliest successful methods for face recognition. The method uses Principal Component Analysis (PCA) to represent faces as linear combinations of eigenfaces, which are the eigenvectors of the covariance matrix of the face images. Similarly, Fisherfaces, proposed by Belhumeur et al. in 1997, uses Linear Discriminant Analysis (LDA) to maximize the ratio of between-class variance to within-class variance. Local Binary Patterns (LBP), proposed by Ojala et al. in 2002, is another popular method for face recognition that encodes the texture information of the face images.

B. Uprising of Deep Learning

In recent years, deep learning has emerged as a powerful tool for face recognition. Deep learning models are based on neural networks that can automatically learn complex representations of face images. The success of deep learning in face recognition can be attributed to the availability of large-scale datasets such as Labeled Faces in the Wild (LFW), YouTube Faces (YTF), and MegaFace.

One of the most influential deep learning models for face recognition is the DeepFace model, proposed by Taigman et al. in 2014. DeepFace is a multi-layer convolutional neural network that directly maps face images to a high-dimensional feature space. The model achieves state-of-the-art performance on several benchmark datasets, including LFW and YTF.

C. Streamlined Loss Functions, especially for Face Recognition

Loss functions are a critical component of deep learning models, as they measure the difference between the predicted output and the ground truth. In recent years, several novel loss functions have been proposed specifically for face recognition. One of the most popular loss functions for face recognition is the triplet loss, introduced by Schroff et al. in 2015. The triplet loss minimizes the distance between the anchor image and the positive image, while maximizing the distance between the anchor image and the negative image in the feature space.

In addition to the triplet loss, several other loss functions have been proposed for face recognition, including center loss, sphere loss, and arcface loss. The center loss, proposed by Wen et al. in 2016, minimizes the distance between the deep features of the same class and learns a center for each class. The sphere loss, introduced by Liu et al. in 2017, adds a margin term to the softmax loss to increase the intra-class compactness and inter-class discrepancy. The arcface loss, proposed by Deng et al. in 2019, combines the softmax loss with an angular margin to improve the discriminative power of the deep features.

III. Methodologies

A. Traditional ML Techniques

Let's start with the Hello World problem of modeling i.e. applying traditional ML techniques on flattened samples of images. I applied Principal Component Analysis (PCA) to pick important contributions of pixels together.
Apart from that, I also tried to go with feature extractors like AKAZE, SIFT, and ORB for concising samples. Following Models are seen:-

- Decision Tree Classifier (Breiman et al. (1984)): Decision tree is a non-parametric supervised learning method used for classification and regression. It involves breaking down a dataset into smaller and smaller subsets by recursively applying a decision tree algorithm. The final result of this algorithm is a tree with decision nodes and leaf nodes. The decision nodes have two or more branches, and the leaf nodes represent a decision or a classification. The tree can be represented by if-else statements, where each internal node represents a test on an attribute and each leaf node represents a class label. One common metric used for selecting the best feature and threshold for each split is the Gini impurity, defined as:

$$I_G(p) = 1 - \sum_{i=1}^{c} p_i^2 \tag{1}$$

where p_i is the probability of an input belonging to class i, and c is the total number of classes. The Gini impurity measures the degree of impurity or uncertainty of a node, with a lower value indicating a more pure node.
The information gain for a given split can then be calculated as the difference between the Gini impurity of the parent node and the weighted sum of the impurities of the child nodes:

$$\Delta I_G = I_G(\text{parent}) - \sum_{i=1}^{2} \frac{N_i}{N} I_G(\text{child}_i) \tag{2}$$

where N_i is the number of inputs in the i-th child node, and N is the total number of inputs in the parent node.
- Random Forest Classifier (Breiman (2001)): Random Forest is a machine learning algorithm that is based on the idea of constructing multiple decision trees at training time and outputting the class that is the mode of the classes of the individual trees. It is an ensemble learning method that combines multiple decision trees to create a forest. Each decision tree is trained on a random subset of the training data, and the final classification is based on the majority vote of the individual trees.

$$\hat{y}i = modeT(x_i;\theta_t)t = 1^T, \tag{3}$$

where $\hat{y}_i$ is the predicted class label for the i-th input sample x_i, T is the total number of decision trees in the forest, θ_t is the set of parameters of the t-th decision tree, and $mode$ denotes the mode function that returns the most frequently occurring class label among the predictions of all decision trees.
- Bagging Classifier (Breiman (1996)): Bagging stands for Bootstrap Aggregating, which is an ensemble learning method that involves training multiple models on different subsets of the training data. The final output is the average of the outputs of the individual models. Bagging can be used with any type of classifier.The bagging algorithm can be summarized by the following equation:

$$\hat{f}bag(x) = \frac{1}{B}\sum b = 1^B \hat{f}^{*b}(x) \tag{4}$$

where $\hat{f}_{bag}(x)$ is the prediction of the bagging classifier, $\hat{f}^{*b}(x)$ is the prediction of the b^{th} model, and B

is the total number of models. In practice, the bagging algorithm is often implemented using decision trees as the base classifier.

- Adaboost Classifier (Freund and Schapire (1997).): Adaboost stands for Adaptive Boosting, which is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. At each iteration, Adaboost assigns a weight to each training example based on how well it was classified by the previous weak classifier. The weight is then used to give more importance to the misclassified examples in the next iteration. Adaboost has been shown to be effective in improving the performance of many machine learning algorithms.Given a training set $D = (x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, where x_i is the i^{th} feature vector and $y_i \in -1, 1$ is the corresponding class label, Adaboost works as follows:
  Initialize the weights of each sample as $w_i = \frac{1}{N}$.
  For each iteration m:
  - Train a weak classifier $h_m(x)$ on the weighted training set $D_m = (x_i, y_i, w_i)_{i=1}^N$.
  - Compute the weighted classification error $\epsilon_m = \sum_{i=1}^N w_i I(h_m(x_i) \neq y_i)$, where I is the indicator function.
  - Compute the weight of the classifier $\alpha_m = \frac{1}{2} \ln \frac{1-\epsilon_m}{\epsilon_m}$.
  - Update the weights of each sample as $w_i \leftarrow w_i \exp(-\alpha_m y_i h_m(x_i))$.

  The final classification is given by $H(x) = \text{sign}(\sum_{m=1}^M \alpha_m h_m(x))$.
  Here, M is the number of weak classifiers, and sign is the sign function. Adaboost gives more weight to the samples that are difficult to classify correctly, and less weight to the samples that are easy to classify correctly, which leads to a better overall performance of the classifier.
- Support Vector Classifier (Cortes and Vapnik (1995)): Support Vector Machine (SVM) is a supervised learning method used for classification and regression. SVM finds the hyperplane that maximizes the margin between the two classes. The margin is the distance between the hyperplane and the closest data points from each class. SVM can be linear or non-linear, depending on the kernel function used. Support Vector Classifier is a kernelized SVM classifier. The equation of the hyperplane is given by:

$$w^T x - b = 0 \tag{5}$$

  where w is the weight vector, x is the input vector, and b is the bias term.
  The distance between the hyperplane and the closest data points from each class is called the margin. The objective of the Support Vector Classifier is to maximize the margin while minimizing the classification error. This can be formulated as the following optimization problem:

$$\min_{w,b} \frac{1}{2}|w|^2 \quad \text{subject to} \quad y_i(w^T x_i - b) \geq 1 \quad \forall i \tag{6}$$

  where y_i is the label of the i-th training example, and $|w|^2$ is the squared Euclidean norm of the weight vector. The inequality constraint enforces that the hyperplane correctly separates the data points into their respective classes.

B. Coming to Deep Learning Techniques

This is the required extension of Traditional Approaches, where instead of flattening the vectors, they will be taken intact via Convolutions, Poolings, and other components of DL Models. Since the spatial information was heavily lost due to flattening, it is beneficial to consider images as 2D-3D Signals and inspect the pixel fluctuations as the representation of some class label.

I applied various DL Architectures with their own set of uniqueness in their components level:-

- MobileNet-v3 Large: MobileNet-v3 is a lightweight neural network architecture designed for mobile and embedded devices. It was introduced by Howard et al. in 2019. It is an extension of the MobileNet-v2 architecture, which uses depthwise separable convolutions to reduce the computational cost of the model. MobileNet-v3 introduces several improvements over its predecessor, such as hard sigmoid activation, squeeze-and-excitation blocks, and a combination of inverted residuals and linear bottlenecks.One important equation used in MobileNet-v3 is the depthwise separable convolution, which is defined as follows:

$$y = PW(BN(\text{ReLU}(DW(x)))) \tag{7}$$

  where x is the input tensor, PW is the pointwise convolution with 1x1 filters, DW is the depthwise convolution with a kernel size of 3x3, BN is the batch normalization operation, $ReLU$ is the rectified linear unit activation function, and B is a set of learnable bias terms.
  The MobileNet-v3 architecture also includes a squeeze-and-excite block, which uses the following equations:

$$z = \text{GlobalAvgPool}(x) \tag{8}$$

$$s = \text{FC}(\text{ReLU}(\text{FC}(z))) \tag{9}$$

$$y = x \cdot \sigma(s) \tag{10}$$

  where x is the input tensor, GlobalAvgPool is the global average pooling operation, FC is a fully connected layer, ReLU is the rectified linear unit activation function, σ is the sigmoid activation function, and s and y are intermediate and output tensors, respectively.

- SqueezeNet 1.0: SqueezeNet is a lightweight neural network architecture designed for mobile and embedded devices. It was introduced by Iandola et al. in 2016. It is based on the idea of using 1x1 convolutions to reduce the number of parameters in the model, which leads to a smaller and faster model. SqueezeNet 1.0 achieves state-of-the-art performance on the ImageNet dataset with only 0.75 million parameters, which is 50x fewer than AlexNet.The architecture of SqueezeNet 1.0 can be represented using the following equations:

$$F_{out} = \text{ReLU}(W_{3x3} * \text{concat}(W_{1x1} * F_{in})) \tag{11}$$

where F_{in} is the input feature map, F_{out} is the output feature map, W_{1x1} and W_{3x3} are the weights of the 1x1 and 3x3 convolutional filters, respectively, and $*$ denotes the convolution operation.
The squeeze layer can be represented as:

$$F_{sq} = \text{ReLU}(W_{1x1} * F_{in}) \tag{12}$$

where F_{sq} is the output of the squeeze layer.
The expand layer can be represented as:

$$F_{ex} = \text{ReLU}(W_{1x1} * F_{sq}) + \text{ReLU}(W_{3x3} * F_{sq}) \tag{13}$$

where F_{ex} is the output of the expand layer.
- ShuffleNet v2 x0.5: ShuffleNet is a neural network architecture designed for mobile and embedded devices. It was introduced by Zhang et al. in 2018. It uses channel shuffling and group convolutions to reduce the computational cost of the model. ShuffleNet v2 x0.5 is a variant of ShuffleNet v2 that uses fewer channels in each layer, which leads to a smaller and faster model. It achieves state-of-the-art performance on the ImageNet dataset with only 1.4 million parameters.The output of a depthwise convolution layer can be calculated as follows:

$$Y_{i,j,k} = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} X_{i,j\cdot\text{stride}+r,k\cdot\text{stride}+s} \cdot K_{r,s,k} \tag{14}$$

where X is the input feature map, Y is the output feature map, K is the kernel weights, i indexes the batch, j indexes the spatial height, and k indexes the depth. R and S denote the height and width of the kernel, and stride is the stride length.
- ResNet-34: ResNet-34 is a deep neural network architecture introduced by He et al. in 2016. It is a variant of the ResNet architecture, which uses residual connections to avoid the vanishing gradient problem in deep networks. ResNet-34 has 34 layers and achieves state-of-the-art performance on the ImageNet dataset. It has become a popular architecture for transfer learning and is often used as a backbone in many computer vision tasks.

$$y = F(x) + x \tag{15}$$

where x and y are the input and output tensors, respectively, and $F(x)$ is the residual function. The idea behind this equation is to add the original input tensor x to the output of the residual function $F(x)$, allowing the gradient to flow directly from the output to the input without getting attenuated.
- DenseNet-121: DenseNet is a deep neural network architecture introduced by Huang et al. in 2017. It is based on the idea of densely connecting each layer to every other layer in a feed-forward fashion. DenseNet-121 is a variant of DenseNet that has 121 layers. It achieves state-of-the-art performance on the ImageNet dataset and has become a popular architecture for transfer learning in computer vision tasks.

$$x_l = H_l([x_0, x_1, ..., x_{l-1}]) \tag{16}$$

where x_l is the output of the l^{th} dense block, H_l is a set of convolutional layers, and $[x_0, x_1, ..., x_{l-1}]$ represents the concatenation of the feature maps from all preceding blocks.

C. Face-Recognition streamlined loss functions

It is a very important inclusion within Deep Learning, because of which the weights get converged and their path of optimization is decided. For facial recognition, it is important to account for several facial attributes associated with the personality and each class representation holds its unique set of attributive combinations. So, unbiased representation and expression is necessary hereby.

I went with following variations in loss functions:-

- Multi-Class Cross Entropy (Authors: Goodfellow et al., 2016): Cross-entropy loss is a popular loss function used in classification problems. It measures the difference between the predicted probabilities of the model and the actual target probabilities. The loss function is minimized when the predicted probabilities are close to the actual target probabilities. In multi-class cross-entropy, the predicted probabilities of each class are compared against the actual target probabilities for each class. This loss function is widely used in deep learning models for multi-class classification problems.

$$L_{ce} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} log(\hat{y}_{ij}) \tag{17}$$

Here, L_{ce} represents the multiclass cross entropy loss, N represents the number of samples in the dataset, C represents the number of classes, y_{ij} represents the ground truth label for the i^{th} sample and the j^{th} class, and $\hat{y}_{ij}$ represents the predicted probability of the i^{th} sample belonging to the j^{th} class. The loss is calculated by taking the negative average of the sum of the logarithm of the predicted probabilities of the correct class for each sample.

- CosFace (Authors: Wang et al., 2018): CosFace is a loss function designed specifically for face recognition problems. It is based on the idea of using the cosine similarity between the feature embeddings of the input image and the weight vector of the true class label as the basis for the loss. The loss function aims to maximize the cosine similarity between the feature embeddings and the weight vector of the true class label while maintaining a margin between the cosine similarity of the true class label and the other classes. This helps to improve the discrimination between classes and produces more accurate face recognition results.

$$L_{cosface} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot(\cos(\theta_{y_i})-m)}}{e^{s\cdot(\cos(\theta_{y_i})-m)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cdot\cos(\theta_j)}} \tag{18}$$

where N is the batch size, θ_{y_i} is the angle between the features of the i^{th} input and the weight vector of the ground truth class y_i, θ_j is the angle between the features of the i^{th} input and the weight vector of the j^{th} class, m is the additive margin, and s is the scaling factor. The goal of the CosFace loss is to increase the angular distance between the features and the weight vectors of different classes while keeping the angle between the features and the weight vector of the ground truth class greater than m.

- ArcFace (Authors: Deng et al., 2018): ArcFace is another loss function designed specifically for face recognition problems. It is based on the idea of using the arc-cosine function of the cosine similarity between the feature embeddings of the input image and the weight vector of the true class label as the basis for the loss. The loss function aims to maximize the arc-cosine similarity between the feature embeddings and the weight vector of the true class label while maintaining a margin between the arc-cosine similarity of the true class label and the other classes. This helps to improve the discrimination between classes and produces more accurate face recognition results.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot\cos\left(m_{y_i}+\theta_{y_i}\right)}}{e^{s\cdot\cos\left(m_{y_i}+\theta_{y_i}\right)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cdot\cos(\theta_j)}} \tag{19}$$

where N is the batch size, m_{y_i} is the margin for the ground truth class of sample i, s is the scaling parameter, θ_{y_i} is the angle between the feature embedding vector and the weight vector of the ground truth class of sample i, θ_j is the angle between the feature embedding vector and the weight vector of the j-th class, and n is the total number of classes. The loss function encourages the feature embedding vector to have a larger cosine similarity with the weight vector of its ground truth class compared to other classes, with a margin m and scaling factor s.

IV. Experimental Details

We conducted our experiments on Google Colaboratory, which provided us with a GPU runtime environment. Specifically, we used the "Python 3" runtime environment with an NVIDIA Tesla K80 GPU. The GPU has 2496 CUDA cores, a base clock of 562 MHz, and 12GB of GDDR5 VRAM. We also utilized Pytorch Lightning, a library that enables faster and optimized deep learning-based training and inference.

In addition to Pytorch Lightning, we used Scikit-Learn and OpenCV libraries for traditional ML techniques implementations and feature extractors API. Scikit-Learn provided us with an array of machine learning algorithms to perform various classification tasks, while OpenCV offered a range of tools for image preprocessing, feature extraction, and object detection.

We evaluated our models on two datasets, the subset of IMFDB (Indian Movie Face database) and LFW (Labelled Faces in the Wild). The IMFDB is a large-scale face database comprising 34512 images of 1006 Indian actors collected from 517 Indian movies. Each image is labeled with the corresponding actor's name, gender, and movie name. The dataset is challenging due to variations in pose, expression, sizes, and lighting conditions.

On the other hand, LFW is a popular face recognition benchmark dataset comprising over 13,000 images of 5,749 individuals collected from the internet. Each image is labeled with the individual's name, and the dataset is challenging due to different distributions of training and inferring portions of dataset (that's why irrespective of any pipeline incorporated here, it resulted to 0%).

V. Evaluation Scheme

- Multi-Class ROC-AUC Curve: The receiver operating characteristic curve (ROC curve) is a widely used evaluation metric in machine learning. It provides a graphical representation of the performance of a classifier by plotting the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. The area under the ROC curve (AUC) is a scalar metric that measures the overall performance of the classifier. In our study, we used the multi-class ROC-AUC curve to evaluate the performance of our models. Let $\hat{y}_i$ be the predicted probability of sample i belonging to the positive class (i.e., class 1), and let y_i be the true class label of sample i, where $y_i \in 1,2,\ldots,K$ for K classes. The OvA ROC-AUC curve can be calculated as follows:
  For each class k:
  - Combine the predicted probabilities of all other classes $j \neq k$ into a single "negative" class probability $\hat{y}i, j^{neg} = \sum j \neq k \hat{y}_{i,j}$.
  - Compute the ROC curve using the true positive rate (TPR) and false positive rate (FPR) across different threshold values for $\hat{y}i, k$ and $\hat{y}i, j^{neg}$.
  - Calculate the Area Under the ROC Curve (AUC) for the ROC curve.

The final Multi-Class ROC-AUC score can be calculated as the average AUC across all classes. The mathematical equation for AUC can be expressed as:

$$\text{AUC} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(w_i w_j - w_i w_j[\hat{y}_i > \hat{y}_j]\right)[y_i = 1][y_j \neq 1] \tag{20}$$

where N is the total number of samples, w_i and w_j are the weights of samples i and j, respectively, and $[\cdot]$ is the Iverson bracket that evaluates to 1 if the condition inside is true and 0 otherwise. The term $[\hat{y}_i > \hat{y}_j]$ is an indicator function that evaluates to 1 if sample i has a higher predicted probability than sample j, and 0 otherwise. The weights w_i and w_j are used to handle class imbalance and can be set to 1 for all samples if the classes are balanced.

- Class-wise Accuracies, Precision, Recall, and Support: We also evaluated the performance of our models on a per-class basis. For each class, we computed the accuracy, precision, recall, and support. Accuracy is defined as the percentage of correctly classified samples in a given class. Precision measures the fraction of true positive predictions among all positive predictions. Recall measures the fraction of true positive predictions among all actual positive samples. Support is the number of samples in the given class.

$$Classwise\ Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{21}$$

$$Precision = \frac{TP}{TP + FP} \tag{22}$$

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

$$Support = TP + FN \tag{24}$$

where True Positives are the number of instances that are correctly classified as positive, False Positives are the number of instances that are incorrectly classified as positive, True Negatives are the number of instances that are correctly classified as negative, and False Negatives are the number of instances that are incorrectly classified as negative.

- Overall Accuracy: In addition to per-class evaluation metrics, we also computed the overall accuracy of our models. The overall accuracy is defined as the percentage of correctly classified samples over all classes. This metric provides a summary of the overall performance of the model.

VI. Results

For the case of every experiment run on LFW (Labelled Faces in the Wild), every supervised algorithm (irrespective of being ML or DL) resulted in 0% Metrics, since the distributions of identities are completely different and independent of each other (thereby nullifying the knowledge of trained models). For DL models, it became a problem due to inactive weight-based connections (tending to 0), set for non-training classes/labels.

While exploring the IMFDB's subset, metrics for the models seemed to be interesting. The following variety of results came for DL algorithms:-

- With Multi-Class Cross Entropy

Models	Acc.	Macro Acc.	Weighted Acc.
MobileNet	20.20	15	24
SqueezeNet	5.82	0	11
ShuffleNet	60.34	58	60
ResNet	<u>70.97</u>	<u>69</u>	<u>71</u>
DenseNet	**74.6**	**74**	**74**

ResNets and DenseNets are performing better and SqueezeNet performed worst for several reasons, some of which are:

  - ResNets and DenseNets have much deeper architectures than SqueezeNet, which allows them to learn more complex representations of the data. ResNets have a residual connection in each block which helps to alleviate the vanishing gradient problem and make training deeper networks easier.
  - DenseNets have dense connections between layers, which allows them to make better use of feature information and reduce the number of parameters. SqueezeNet, on the other hand, uses a "fire module" that tries to squeeze down the number of channels before expanding them again, which can result in a loss of information.
  - ResNets and DenseNets have been trained on much larger datasets than SqueezeNet, which allows them to learn more generalizable features. This is particularly important in the case of face recognition, where the variations in lighting, pose, and expression can be quite significant.
  - ResNets and DenseNets use convolutional layers with larger kernel sizes, which allows them to capture more global information about the image. SqueezeNet uses smaller kernels, which can limit its ability to capture complex spatial relationships.
  - ResNets and DenseNets have been fine-tuned for face recognition, which means that they have been specifically optimized for this task. SqueezeNet was not designed with face recognition in mind, and so it may not be as effective as these other architectures for this particular application.

- With Other Losses (only done on top-3 performing DL models)

Models	Acc.	M Acc.	W Acc.
ShuffleNet (ArcFace)	46.23	40	48
ShuffleNet (CosFace)	4.11	1	7
ResNet (ArcFace)	70.12	68	70
ResNet (CosFace)	1.28	1	1
DenseNet (ArcFace)	**78.46**	**77**	**79**
DenseNet (CosFace)	1.87	3	2

The following are the possible reasons why ArcFace loss outperforms Multiclass Cross Entropy while CosFace loss underperforms:

- Angular Margin: ArcFace and CosFace both incorporate angular margins into their loss functions to enhance inter-class separability. However, the margin function of CosFace results in a comparatively larger intra-class variance than ArcFace. The reason is that the margin function in CosFace only applies to the true class, while in ArcFace, it applies to all classes, leading to more robust and discriminative embeddings.
- Normalization: Both CosFace and ArcFace normalize the embedding vector. However, ArcFace further normalizes the weights of the last fully connected layer, resulting in less sensitivity to the magnitude of feature vectors. This normalization of weights helps to preserve feature consistency and leads to better generalization performance.
- Softmax Function: In ArcFace, the softmax function is applied on a modified version of the input logit vector. The modification involves adding an angular margin term to the logits. The use of the angular margin term in the softmax function better aligns the decision boundary with the class boundaries.
- Optimization: The optimization of the ArcFace loss function is simpler and more stable than CosFace. ArcFace loss function optimization involves computing gradients with respect to the modified input logits, while CosFace optimization involves computing gradients with respect to the weights of the last fully connected layer. The direct optimization of ArcFace loss results in a faster convergence rate and better optimization.
- Performance on Large-scale Face Recognition: ArcFace has shown better performance than CosFace on large-scale face recognition datasets such as MS-Celeb-1M and MegaFace. This is because ArcFace produces more discriminative and robust embeddings, leading to better matching performance.
- Hyperparameters: Lastly, it is possible that the hyperparameters of CosFace were not tuned well enough. As CosFace involves more hyperparameters than ArcFace, improper tuning of hyperparameters may lead to a drop in performance.

Regarding class-wise metrics, those are present in the Google Colaboratory notebook. For Multi-Class ROC-AUC Curves, they are present on the next page.

VII. Limitations

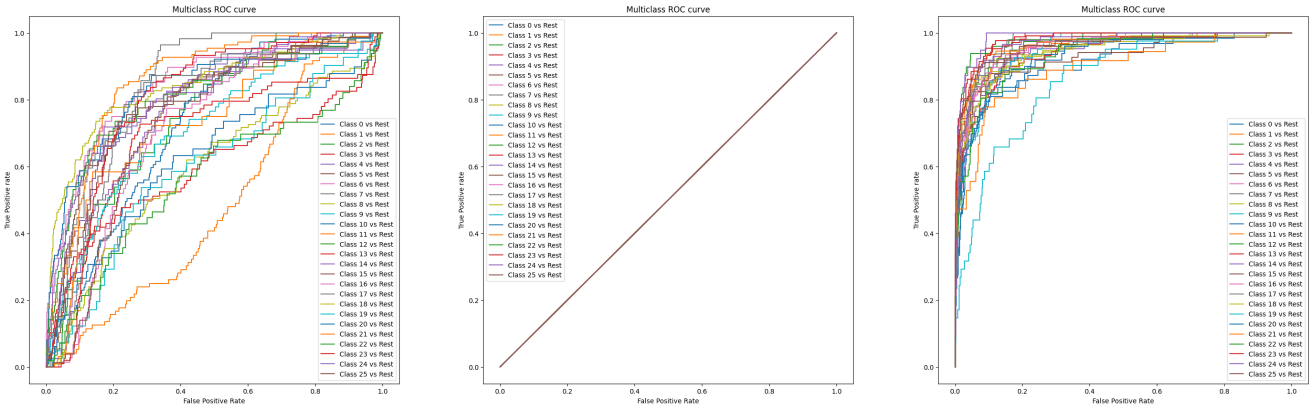
In the case of Labelled Faces in the Wild (LFW) dataset, the model suffers from 0% metrics with the approach of label-image pairs. This is because the LFW dataset consists of face images captured under unconstrained conditions, such as different lighting, angles, expressions, and occlusions, making it challenging to recognize the face. The approach of label-image pairs does not take into account these variations and lacks the ability to generalize to unseen data. Therefore, it is necessary to adopt more advanced techniques that can handle the variability of face images and enable the model to learn robust and discriminative features.

Traditional machine learning models for facial recognition, such as Support Vector Machines (SVM), Random Forests, and k-Nearest Neighbors (k-NN), suffer from several limitations in real-life cases. One of the main limitations is their dependency on handcrafted features, which are manually designed by domain experts based on prior knowledge and experience. These features may not capture the intrinsic characteristics of the face images and lead to suboptimal performance. Moreover, these models are sensitive to the variations in face images, such as pose, illumination, and occlusions, and require a large amount of labeled data for training, which may not be available in real-world scenarios.

Convolutional Neural Networks (CNNs) have been widely used for facial recognition due to their ability to automatically learn hierarchical features from raw pixel values. However, they also suffer from some limitations in real-life cases. One of the main limitations is their sensitivity to variations in face images, such as pose, illumination, and occlusions, which can significantly affect the performance of the model. Moreover, CNNs require a large amount of labeled data for training, which may not be available in real-world scenarios, and suffer from overfitting when the dataset is small. Another limitation is the difficulty of interpreting the learned features, which may limit their applicability in some domains, such as forensic science. Finally, CNNs are computationally intensive and require high-end hardware for training and inference, which may not be affordable for some users.

References

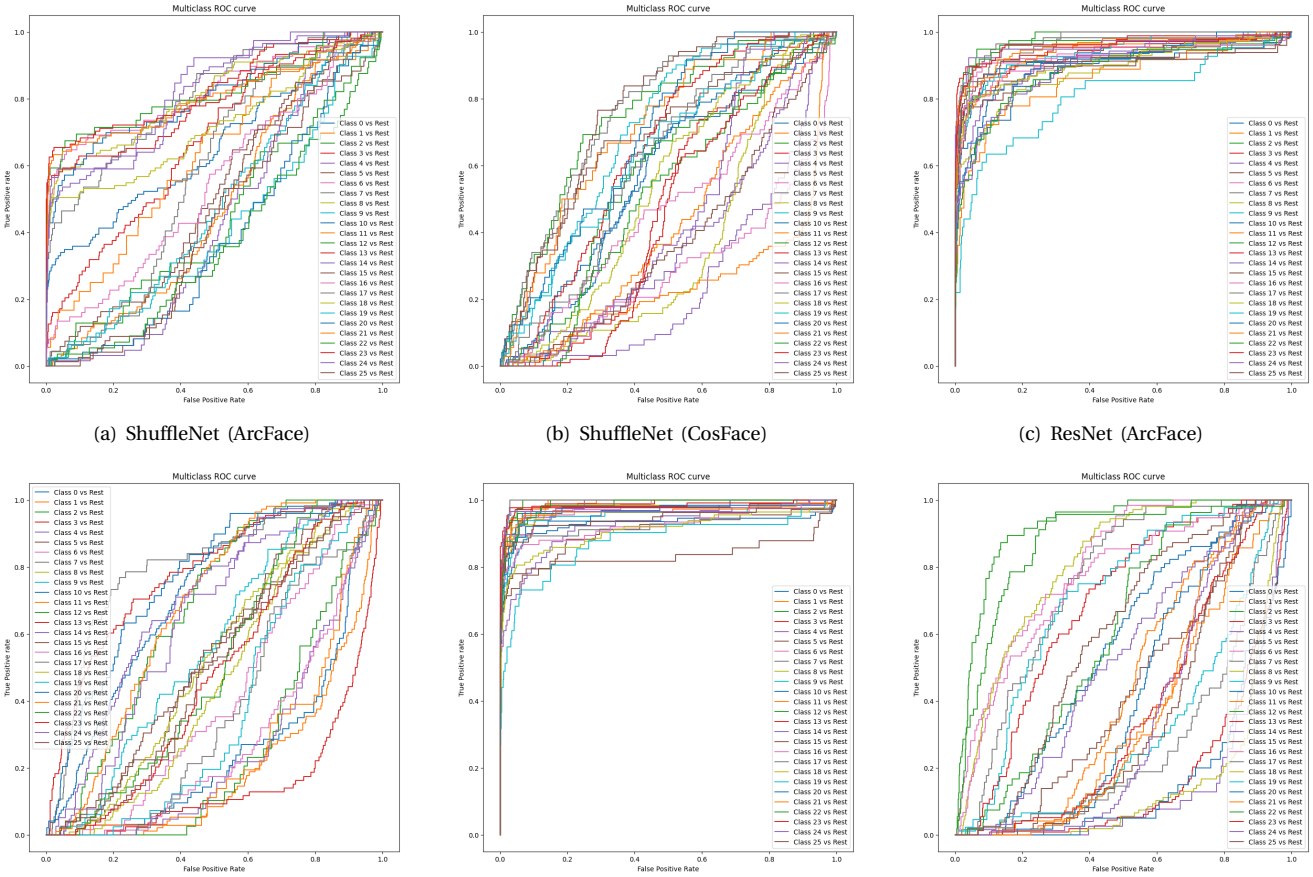
[1] https://cvit.iiit.ac.in/projects/IMFDB/
[2] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9582652&tag=1
[3] http://vis-www.cs.umass.edu/lfw/.
[4] https://arxiv.org/abs/1801.09414.
[5] https://arxiv.org/abs/1801.07698.
[6] https://arxiv.org/abs/1905.02244
[7] https://arxiv.org/abs/1602.07360
[8] https://arxiv.org/abs/1707.01083
[9] https://arxiv.org/abs/1512.03385
[10] https://arxiv.org/abs/1608.06993



(a) MobileNet (b) SqueezeNet (c) ShuffleNet

(d) ResNet (e) DenseNet

Fig. 1. MultiClass-ROC Curve for IMFDB Dataset of subset with 26-classes and Multi-Class Cross Entropy



(a) ShuffleNet (ArcFace) (b) ShuffleNet (CosFace) (c) ResNet (ArcFace)

(d) ResNet (CosFace) (e) DenseNet (ArcFace) (f) DenseNet (CosFace)

Fig. 2. MultiClass-ROC Curve for IMFDB Dataset of subset with 26-classes and Other Losses