A Novel Approach to End-to-End Facial Recognition Framework with Virtual Search Engine ElasticSearch

Dat Nguyen Van[2,3](✉), Son Nguyen Trung[1], Anh Pham Thi Hong[1], Thao Thu Hoang[1], and Ta Minh Thanh[1,4]

[1] Research and Development Department, Sun Asterisk, Hanoi, Vietnam
{nguyen.trung.son,pham.thi.hong.anh,hoang.thu.thao}@sun-asterisk.com
[2] VinAI Research, Hanoi, Vietnam
v.datnv21@vinai.io
[3] University of Engineering and Technology, Hanoi, Vietnam
[4] Le Quy Don Technical University, 236 Hoang Quoc Viet, Hanoi, Vietnam
thanhtm@mta.edu.vn

Abstract. Facial recognition has been one of the most intriguing, interesting research topics over years. It involves some specific face-based algorithms such as facial detection, facial alignment, facial representation, and facial recognition as well; however, all of these algorithms are derived from heavy deep learning architectures, which leads to limitations on development, scalability, flawed accuracy, and deployment into publicity with mere CPU servers. It also requires large datasets containing hundreds of thousands of records for training purposes. In this paper, we propose a full pipeline for an effective face recognition application which only uses a small Vietnamese-celebrity datasets and CPU for training that can solve the leakage of data and the need for GPU devices. It is based on a face vector-to-string tokens algorithm then saves face's properties into Elasticsearch for future retrieval, so the problem of online learning in Facial Recognition is also tackled. In comparison with another popular algorithms on the dataset, our proposed pipeline achieves not only higher accuracy, but also faster inference time for real-time face recognition applications. AQ1 AQ2 AQ3

Keywords: Facial recognition · Visual search engine · End-to-end applications · Online learning · ElasticSearch

1 Introduction

1.1 Overview

Thanks to the rapid development of technologies, facial recognition is increasingly better and evolving. Many companies all over the world are paying great attention to facial recognition technology for their authentication systems

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12951, pp. 1–17, 2021.
https://doi.org/10.1007/978-3-030-86970-0_32

instead of using traditional verification methods such as fingerprints or iris. It thus has been developed in the future as well as making it practical in many other fields. More specifically, the workable applications of the facial recognition technology in every aspect of life are diverse [30,39], such as security [29], internet of things and mobile systems [2], real-time identification systems [5], bio-metric systems based on motion detection and facial features [32].

It is no wonder that a full pipeline for the facial recognition applications requires some underlying, complex algorithms consisting of: extracting frontal human faces in given images, called Face Detection [16,24,26,50], optional face alignments for aligning face's positions [47], the necessity of representing faces in the form of numeric vectors [17,43], and distinguishing faces [46] relied on these continuous vectors from previous steps. Critical reviews of such algorithms are listed below:

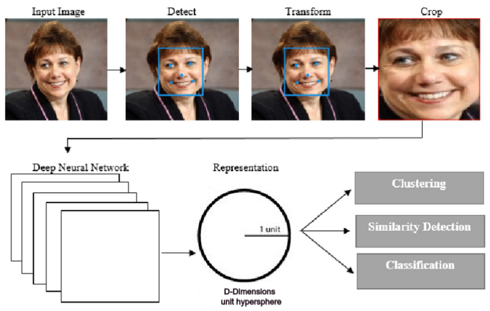


Fig. 1. An overview of full pipeline for face recognition applications

- Face detection, since the 2000s, had a lot of different approaches, but all were either slow or delivered low accuracy, or both. A big innovation came in 2001 when Viola and Jones invented the Haar-based cascade classifier [41], and in 2002, it was continuously improved by Lienhart and Maydt [25]. As a result, the algorithm has become faster and could be run in real-time with 95% accuracy on a difficult dataset. Until 2010, along with the explosion of deep learning [14,38], the research area has reached the state-of-the-art accuracy.
- Attaining facial feature vectors after performing face detection is greatly essential for the recognition step. Algorithms [13,35] are examples to convert these cropped face's images into vectors of specified dimensions that denote the most crucial face's characteristics.
- For identity recognition, well-known approaches based on deep learning architecture like [1,12,35], with the concept of spare representation [46], basing on clustering [36], or Cosine Loss proposed by Hao Wang in the paper [42].

A common full pipeline of facial recognition can be described in Fig. 1.

1.2 Challenging Issues

Each of the algorithms mentioned above has proved its strength and made significant contributions to the research area of face recognition application; however, there are still some issues that need to be worked on. First of all, it can be easily seen that these methods mostly offer only one part amongst the full pipeline of face recognition. The adoption of deep models demonstrate more impressive accuracy than the other approaches, but those models need to be trained on an extremely large, diverse dataset with the size up to hundreds of thousands or millions of images. As a result, it seems to be too slow for face application's development, deployment, management or comes at the expense of high-priced physical devices. The increasing demand for GPU servers for training and deployment has been even more ubiquitous than ever. Another challenge is that the existing online learning problems [33] make these models subject to the requirement of periodic training in order to preserve the system's accuracy whenever new faces are added. Furthermore, a scarcity of body research for the full face recognition pipeline, it need to be researched and widen.

1.3 Our Contributions

Our contributions are summarized as follows:

1. With all these drawbacks discussed above, we have proposed a new approach to an end-to-end facial recognition application in the form of a complete pipeline consisting of: development, deployment, and model version management.
2. In comparison to other well-known methods, our pipeline not only acquires an impressive prediction accuracy with a very challenging dataset that solves the dearth of collection of face data, but it has also gained a very speedy time response for real-time applications.
3. Moreover, the cost for necessary physical devices is reduced exponentially by applying a vector-to-string token algorithm, so that we can train, release the model directly in CPU servers.
4. Last but not least, instead of using a deep learning model for identifying faces, ES is leveraged for storing, creation, retrieval of face identity, thus the online learning problems in face recognition apps are also tackled.

1.4 Roadmap

The rest of the paper is organized as follows. Section 2 presents reviews of related works. Data pre-processing and re-balancing methods are shown in Sect. 3. Section 4 discusses the proposed methods of facial recognition system and experimental results are presented in Sect. 5. Our conclusions and future works are described in Sect. 6.

2 Related Works

2.1 Face Detection

Face detection, which means determining the location and size of a human face in a digital image, is a fundamental step for many face-related technologies. There is a variety of face-based algorithms that take face detection as the foundation to acquire adequate accuracy, such as face verification [13], face recognition [48], face anti-spoofing [22]. In the other works, the purpose of the face detection algorithm is to improve the accuracy of these algorithms by eliciting only the frontal faces as the algorithm's inputs. To date, some adoption of deep learning architectures using the concept of convolutional neural network (CNN) for this step like MTCNN [50], Cascade CNN [24], R-CNN [16], SSD [26], *etc* have achieved remarkable progresses.

2.2 Face Representation

Face representation (in other words, facial-features extraction) is the process of encoding raw facial images into continuous vector representation in high-dimensional feature space [43]. Traditionally, facial features are extracted by manually design patterns as edges, lines, four-rectangle features in Viola Jones algorithm [40] or grids of Histograms of Oriented Gradient (HOG) descriptors [11].

Nowadays, statistical facial-features extraction has been performed automatically and more efficiently (in terms of both time and feature quality [43]), through CNN, a class of deep neural networks [17]. Allowing spatial features preservation, CNNs are suitable for learning feature embedding for 2-D topology data type, including images and facial pictures [17]. Using facial embedding learned by CNNs out-performs most traditional methods in several downstream tasks including Face Recognition, Face Verification [43].

2.3 Principal Component Analysis (PCA)

PCA [15] is a dimensionality reduction technique that was created in 1901 by Karl Pearson. It uses recognition of statistical design to shrink dimensionality and extract features. This approach has been used in various applications since its appearance, such as handwritten recognition, neuroscience, quantitative finance, and image compression.

2.4 FaceNet Architecture in OpenFace

FaceNet [35] was developed in 2015 by Google researchers for the face recognition task. Essentially, a CNN is responsible for extracting features of the face image. The CNN training was performed on large datasets (vggface, MS-Celeb-1M). The key feature of FaceNet is that it uses the Triplet loss function to minimize the distance between similar faces and maximize the distance to dissimilar faces:

$$loss(\mathbf{A}, \mathbf{P}, \mathbf{N}) = max(\|\mathbf{f(A)} - \mathbf{f(P)}\|^2 - \|\mathbf{f(A)} - \mathbf{f(N)}\|^2 + \alpha, 0),$$

where $\mathbf{A}$ is anchor input, $\mathbf{P}$ is a positive input, $\mathbf{N}$ is a negative input, α is a margin between positive and negative pairs.

3 Dataset

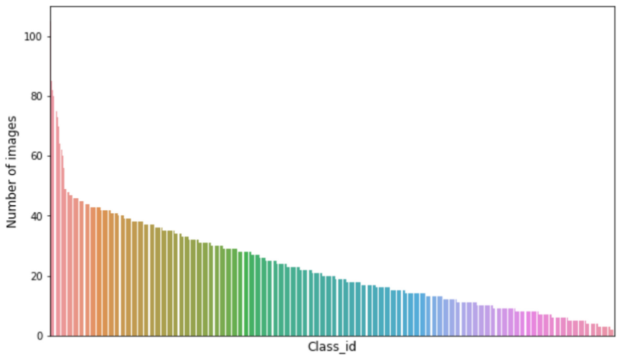


Fig. 2. The number of images in each class

Our proposed model is implemented on the VN-Celeb[1] dataset that is a collection of Vietnamese celebrities's images collected from google image search. The dataset contains 24.125 images belonging to 1020 famous Vietnamese people (1020 classes). More specifically, the average number of photos in each class is around 23; 7 classes have only 2 shots, and the class with the most images has up to 105. It can be said that the dataset is very challenging both in term of size and of each class' proportion. To elaborate,

– The numbers of images of each class are severely imbalanced.
– The size of dataset is small in comparison to other face recognition dataset.
– Some problems affecting differences recognition such as: lighting condition, posing, and picture quality (Fig. 3). AQ4

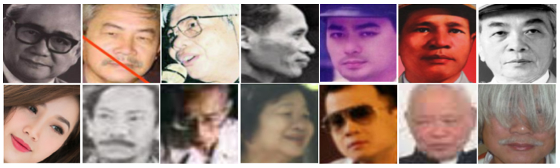


Fig. 3. Some noise images in the VN-Celeb dataset like black and white, blurred, low resolution, masked.

Firstly, the image's proportion between classes is severely imbalanced. Table 1 illustrates the percentage of the classes classified by the range of the number of photos. We can see that the proportions of classes in two ranges: [2, 5) and [50, 105] are only 5%, 3% respectively; meanwhile, the classes belonging to scope [30, 50) account for 27%. Especially, classes having [5, 30) elements show the highest percentage of 65%. As you can see in Fig. 2, the distribution of images in each class is extremely disproportionate, because the number of photos stretch widely from 2 to 105 per class.

[1] https://drive.google.com/drive/folders/1I3KXcGpmm6zpw_y07p-7wIKt5K08iOgc.

Table 1. The image proportion

No shots	[2, 5]	[5, 30]	[30, 50]	[50, 105]
Percentage	5%	65%	27%	3%

Secondly, the size of the VN-Celeb dataset consists of 24.125 images, while the number of classes is 1020. In other words, this task would fall into the problem of few-shot learning [44] - generalizing from a few training examples.

Another issue is the variation of the dataset images in lighting conditions, pose, and image quality... Many photos show only one part of the face, have the head upside-down, or show no face. Moreover, there are both grayscale and RGB color photos included.

4 Proposed Model

In this section, the proposed pipeline would be described step by step and illustrations will be provided in order to demonstrate the idea in more detail.

As mentioned in the previous section, the dataset is challenging. The size is only 24.1k images containing 1020 classes, which is of much smaller size and much more imbalanced comparing to other datasets such as FaceNet's [35] private dataset with 100M-200M, DeepFace [37] using a private dataset with 4.4M images, OpenFace [2] training on combined dataset CASIA-WebFace [49] and FaceScrub [28], and so on. Therefore, before the dataset can be passed into the Face Embedding model to get representing vectors for training purposes in the encoding phase, data cleaning, re-balancing, and some other data preprocessing techniques must be performed. Our proposed pipeline application is shown in Fig. 4.

4.1 Face Detection

This phase plays an integral role in the accuracy of the whole application, and is the foundation for subsequent steps. After experimenting some tools like Dlib [21], Haar Cascades [3], MTCCN [50], *etc* to extract faces from given images, we decide to take 2 methods for this phase, these methods are interchangeable depending on particular spec. Firstly, we train SSD [26] with based network MobileNet on Labeled Faces in the Wild dataset [23] to take advantage of the fast inference time of MobileNet architecture [20]. Another choice is using a pre-trained library, MTCNN [50]. Although detecting faces with MobileNet-SSD gives better performance, the bounding boxes results are not as good as those of MTCNN counterparts. So, hinging on the purposes, we can switch between two methods.

4.2 Data Pre-processing

The dataset implemented is really challenging due to its size, its properties, and the proportion of the image in each class that needs to be preprocessed. To mitigate adverse effects and enhancing the system's accuracy, we generated more data to classes in which the number of images lower than 20 by applying some non-geometric preprocessing techniques including Blurring, Sharpening, Smoothing, Histogram Equalization, Gamma Correction $\Longrightarrow$ DOG Filtering $\Longrightarrow$ Contrast Equalization [34]. The dataset obtained after preprocessing is passed through the next step to get face representation for training in the encoding phase.

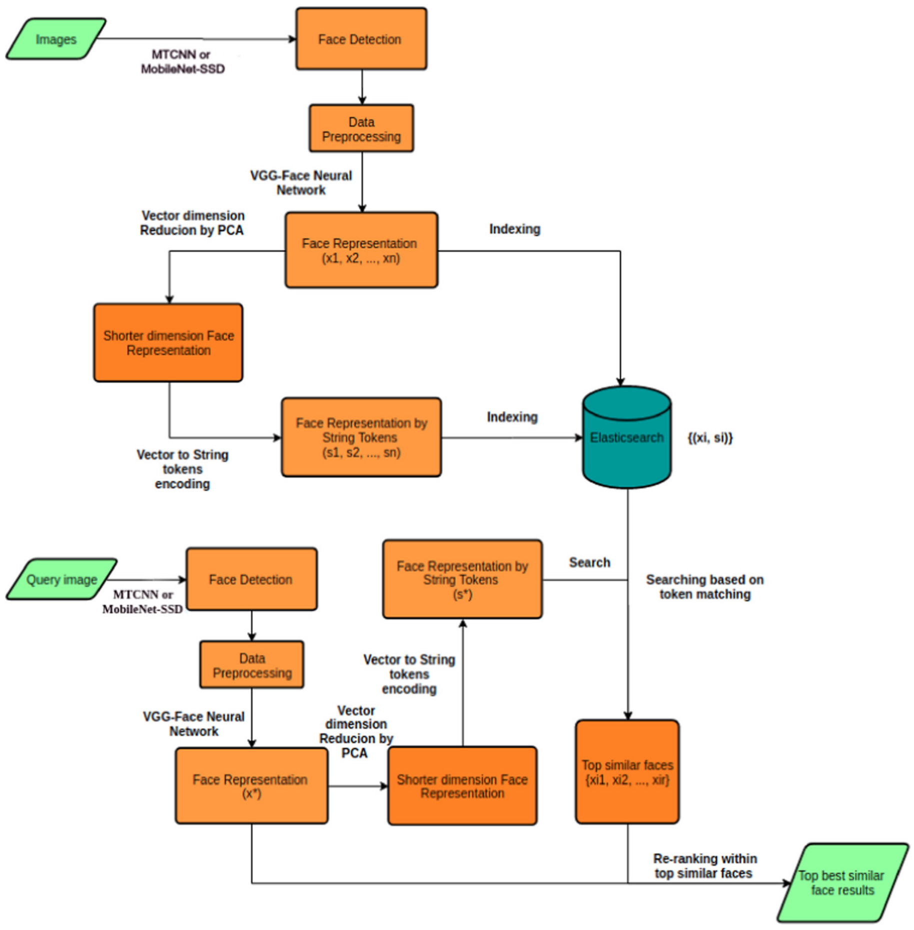

Images
MTCNN or MobileNet-SSD
Face Detection
Data Preprocessing
VGG-Face Neural Network
Vector dimension Reducion by PCA
Face Representation (x1, x2, ..., xn)
Indexing
Shorter dimension Face Representation
Face Representation by String Tokens (s1, s2, ..., sn)
Vector to String tokens encoding
Indexing
Elasticsearch
{(xi, si)}
Query image
MTCNN or MobileNet-SSD
Face Detection
Face Representation by String Tokens (s*)
Search
Searching based on token matching
Data Preprocessing
Vector to String tokens encoding
VGG-Face Neural Network
Vector dimension Reducion by PCA
Face Representation (x*)
Shorter dimension Face Representation
Top similar faces {xi1, xi2, ..., xir}
Re-ranking within top similar faces
Top best similar face results

Fig. 4. Workflow of the application

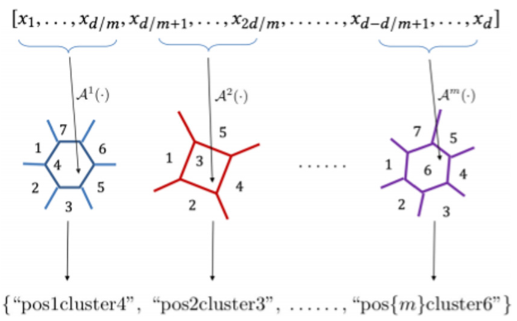
4.3 Face Embeddings with VGG-Face

At this step, the embedding vectors containing the most informative face elements represented as a form of the numeric array is extracted from a DNN. Based on experiments and evaluation on some deep learning architectures like FaceNet [35], VGG-Face [4], ArcFace [13], VGG-Face was chosen because of both its accuracy and performance running on our framework. The based model is ResNet50 [18], all Fully Connected Layers are removed for converting face image into numeric representing vectors that can be used for recognizing purpose.

4.4 Reducing Face Vector Dimensions by PCA

The default dimension output of VGG-Face is 2048. As far as we know, it is very difficult for real-time applications to run on the server without GPU, since the computational cost is very high and may hurt the performance of our whole framework detrimentally. So some data dimension reduction algorithms need to be implemented. An attempt of training and evaluating a list of array dimensions in the range [256, 512, 1024] was conducted through a combination between this phase and the encoding phase to find the best dimension for production deployment (which will be explained in the next subsection). Finally, after assessing both accuracy and performance deliberately, we decided to apply PCA [45] to compress from 2048 to 512 dimensions as the best dimension to represent face data.

4.5 Encoding Face Embeddings to String Tokens



$[x_1, \ldots, x_{d/m}, x_{d/m+1}, \ldots, x_{2d/m}, \ldots\ldots, x_{d-d/m+1}, \ldots, x_d]$
$\mathcal{A}^1(\cdot)$
$\mathcal{A}^2(\cdot)$
$\mathcal{A}^m(\cdot)$
{"pos1cluster4", "pos2cluster3", ..., "pos{m}cluster6"}

Fig. 5. Illustration of the subvector-wise clustering

In this section, we explain the reason why we decided to change the data-type for face representation. Actually, it is possible to recognize which name of a person after getting numeric face embedding vectors extracted from VGG-Face by using well-known similarity methods like Euclidean, Cosin, or a deep learning classification model. However, using these methods to calculate similarity within all face vectors of high dimensions to find the best similar faces is computationally expensive and time-consuming. Besides, another option of using deep learning algorithms for this task is proposed, but it has long inference time and high latency due to the complexity of deep model architectures for recognition, not to mention inefficiency regarding the online learning problems. The problems mentioned above have proved that the application is challenging and difficult to put into practice as well as to apply in real-life. In this paper, we utilize an encoding algorithm from [27] to convert numeric face vectors into collections of string tokens that can be retrieved faster with Elasticsearch, which is beneficial for real-time applications. The encoding algorithm is inspired by the idea of subvector-wise clustering. More specifically, with any numeric face vector $\mathbf{x} \in \mathbb{R}^d$, we divide it into m position as below:

$$[x_1, ..., x_{d/m}, x_{d/m+1}, ..., x_{2d/m}, ..., x_{d-d/m+1}, ..., x_d]$$

Considering m position as m subvectors $= \{x^1, x^2, ..., x^m\}$, and $P^i := \{x^i_1, x^i_2, ..., x^i_n\}$ where $i = \{1, ..., n\}$ as the collection of the i^{th} subvectors merged from each position in the whole dataset. We apply k-means algorithm to cluster each P^i into k clusters ($k > 1$). By doing that, we are able to encode the original numeric face vector into string tokens. For example:

$$[\text{“}pos1cluster2\text{”}, \text{“}pos2cluster\{k\}\text{”}, ..., \text{“}pos\{m\}cluster1\text{”}]$$

is an example vector encoded from an origin numeric face vector applying the encoding algorithm above with m positions and k clusters. The illustration of the encoding algorithm is in Fig. 5.

4.6 Data Indexing and Searching Method

We provide insights about the data indexing progress and the method for retrieving face identity. We elaborate in two parts: Data Indexing and Searching Method as follows.

Data Indexing. After getting string tokens from the previous encoding phase, we need to store them for face retrieval afterward. Adding to the fact that we are aware that retrieving face recognition from string representation is much faster than from high dimension numeric vectors, in order to achieve an optimal string matching mechanism, we leverage the concept of inverted-index-based search engine from Elasticsearch (ES). To index data into ES, we construct a JSON format including numeric vectors, string tokens together with some other properties of a person such as a name, age, address, image path, and so on. Such JSON format depicts for each face in a total of 1020 classes. The format of the JSON is demonstrated below:

```
body = {
        "user_id": user_id,
        "user_name": user_name,
        "image_url": image_url,
        "embed_vector": embed_vector,
        "string_token": string_tokens,
        ...
}
```

In the end, we have an array of JSON with the length equals to that of the training dataset. Then, we simply use available ES API (Application Programming Interface) functions to index data into Elasticsearch Server.

Searching Method. In this step, we follow the steps of implementation Cun (Matthew) Mu [27] did in his paper, but for face data.

With any query image, we apply the same steps as the training steps explained above to attain a string token $\hat{s}$, then use it for searching. Top r similar faces are obtained relied on overlap between string tokens set $\hat{s}$ and the ones stored in ES server $\{s_1, s_2, ..., s_n\}$

$$i_1, i_2, ..., i_r = \underset{i \in \{1,2,...,n\}}{\text{argmax}} \mid \hat{s} \cap s_i \mid \tag{1}$$

ES provides us with RESTful API for searching conveniently, all we need to do is to build a JSON-encoded request body which would instruct the ES server to compute and then return the visual search results.

In the JSON format, we establish 2 prime parts inside for getting sorted results from ES. The first part taking function score query [8] that is responsible for finding top r faces that share the most common in string tokens with $\hat{s}$, then using a custom rescore API function [9] provided by ES to re-sort the collection of top r vectors above. The JSON request body plays an indispensable role in our end-to-end applications.

4.7 Model Serving and Management with Tensorflow

In any AI-related application, model management is necessary in order to operate the system more easily, smoothly, conveniently. At this point, we have had several trained models from Face Detection, Face Embedding to vector-to-string encoding model that needs to served for inference time whenever receiving requests to face recognition server. With these two first deep models, instead of storing the physical file and loading it directly in the server, we convert these models into TensorFlow PB format then using TensorFlow serving API [10] to serve and manage in a separate server. Regarding the last one, the vector-to-string encoding model, Data Version Control [7] is leveraged.

4.8 Django Framework for Development and Deployment

To this point, in order to make our application available to the community by providing open API functions, a framework is necessary. Our application was developed and deployed on the Django Framework [6], one of the most popular framework using Python.

5 Experiments

In this section, to prove the validity of our proposed pipeline, we compare our proposed pipeline to another well-known framework, OpenFace [2] on the VN-celeb dataset. In addition, we also illustrate the impact of the number of positions as well as the number of clusters in the vector-to-string encoding algorithm.

5.1 Experimental Environment

Our proposed pipeline experiments are conducted on a computer with IntelCore i5-4460 CPU @3.2 GHz, 16 GB of RAM, and 256 GB SSD hard disk. The models are implemented with the python 3.6.8 environment.

5.2 Evaluation Method

Accuracy. Accuracy is the most important metric which is used to evaluate the efficiency and generality of almost every model. It describes how well the model performs by providing a ratio between the number of correct predictions and the total elements of testing set. The formula is shown below:

$$accuracy = \frac{\sum_{i=0}^{c-1} \sum_{j=0}^{n_{ci}} E(y_j^*, \hat{y_j})}{N}, \quad (2)$$

where N is the length of test set, c is the number of classes that need to predict, n_{ci} provide how many items belonging to class ci with $i = \{0, 1, ..., c-1\}$, and $E(y_j^*, \hat{y_j})$ is a boolean method to compare y_j^* and $\hat{y_j}$ which return "1" if $y_j^* = \hat{y_j}$ and "0" if otherwise.

Recall. Recall is the fraction of the number of positive prediction of classes to it's actual positive. Recall is defined as below:

$$recall = \frac{1}{c} \sum_{i=0}^{c-1} \frac{\sum_{j=0}^{n_{pi}} (E(y_j^*, \hat{y_j}) == 1)}{n_{ci}} \quad (3)$$

As you can see in the recall formula: c is the number of classes that need to predict, n_{pi} illustrates the count of positive prediction, n_{ci} show the actual positive of each class and $E(y_j^*, \hat{y_j}) == 1$ indicates one correct prediction.

5.3 Experimental Analysis

In this section, to prove the efficiency of our proposed pipeline, we have re-implemented and trained OpenFace [2] on the above dataset to compare with our proposed algorithm.

OpenFace with VN-celeb Dataset. In order to allow a fair comparision, we re-implement OpenFace [2] following the same steps as we do with our algorithm. It is noteworthy that the dataset only contains frontal, portrait images of Vietnamese celebrities, so the face detection step for training can be ignored. In the first place, we do data preprocessing as in Sect. 4.2 before feeding into FaceNet's triplet loss [35] to train for a total of 150 epochs based on this empirical experiments. However, since its weights have been trained in 500k images, instead of initializing for the whole deep architecture, we do fine-tuning techniques by resetting weights of some last FaceNet's layers and freezing all the remaining layers, then warming up the model in 30 epochs. After that, we unfreeze and train the whole model in the last 120 epochs.

After training completed, we have a collection of numeric vectors in 128 dimensions from the trained model generating above that characterize the face's properties. As mentioned in [2], the final step is to put these vectors through the Support Vector Machine (SVM) [19] from Scikit-learn [31] as the classifier for the distinction between each individual.

Our Proposed Pipeline. To gain our full pipeline, we follow the steps described in Sect. 4. We also ignore face detection step for the reason mentioned above. In the next step, data preprocessing techniques in Sect. 4.2 is applied, the output then passed through VGG-Face [4] with all fully connected layers eliminated to get face representation vectors of 2048 dimensions as default. It is clear that the dimension of 2048 is too long for a real-time face recognition application, so we decide to apply PCA [45] to reduce the number of dimensions from 2048 to 512 to select the most useful face principal components. At this point, it is to encode numeric face vectors to string tokens, following Sect. 4.5, we divide all face vectors of 512 dimensions in the training dataset into m positions: $P^i := \{x_1^i, x_2^i, ..., x_n^i\}$, where $i = \{1, ..., n\}$, n is the length of the training dataset. Then, we apply separate k-means algorithm from Scikit-learn [31] library to each i^{th} collection vector P^i. The array of trained k-mean's models are saved for future inference.

Using k-mean's models which have been trained to get string tokens, we combine these tokens with some other personal properties such as name, address, phone number, division, nationality, email, numeric face vectors, *etc* to build a JSON object as in Sect. 4.6 for indexing data into ES server.

For inference, to get the individual identity, we just need to build a JSON request body as in Sect. 4.6 and make use of ES's searching API for face retrieval. Finally, the top best 5 similar faces would be returned. The individual identity is the name field of a record with the highest score calculated by ES score function.

Particularly, all deep learning, as well as k-means models, are protected and managed by TensorFlow serving and DVC respectively as we describe in Sect. 4.7. Finally, we develop and deploy our full pipeline application with the Django framework[2].

5.4 Experimental Results and Comparison

In this section, we compare the results of our proposed method to that of OpenFace [2], which we implemented in Sect. 5.3.

According to result's statistic and comparison, it can be concluded that our proposed application bring about many benefits to face recognition applications. Let's see some statistic tables and chart below:

Table 2, Table 3, Table 4 represent statistic tables we built for the purpose of comparing the accuracy, recall and inference time between OpenFace [2] and ours pipeline using numeric face vectors of different dimensions in range of [256, 512, 2048] for vector-to-string tokens algorithms. They also demonstrate the impact of the number of positions, clusters in the encoding algorithm on the evaluation metrics. It is important to note that our accuracy completely outperforms that of the OpenFace counterparts throughout all row records in the three tables with relatively equivalent in two recall columns, some of ours are higher, especially in Table 3 and Table 4. Moreover, by encoding high-dimensional vectors into string tokens, we have the ability to gain the same performance as OpenFace did. The OpenFace lib just takes face vectors of 128 dimensions, meanwhile, we got a comparable inference time with far higher dimensional vectors that obviously comprise more facial features.

[2] https://www.djangoproject.com/.

Table 2. Ours evaluation metrics affected by *Npositions* and *Nclusters* with our face vector of ***256*** dimensions and OpenFace

N positions	N clusters	Accuracy		Recall		Inference time	
		Ours	OpenFace	Ours	OpenFace	Ours	OpenFace
64	19	**90.93**	87.92	**88.40**	87.90	0.078 s	**0.0445 s**
64	20	**90.74**	87.92	**88.33**	87.90	0.097 s	**0.0445 s**
64	21	**91.03**	87.92	87.66	**87.90**	0.106 s	**0.0445 s**
64	22	**91.52**	87.92	87.84	**87.90**	0.083 s	**0.0445 s**
32	19	**90.97**	87.92	86.76	**87.90**	0.064 s	**0.0445 s**
32	20	**91.20**	87.92	86.84	**87.90**	0.063 s	**0.0445 s**
32	21	**91.47**	87.92	86.89	**87.90**	0.06 s	**0.0445 s**
32	22	**91.39**	87.92	87.18	**87.90**	0.055 s	**0.0445 s**
16	19	**92.51**	87.92	81.21	**87.90**	**0.042 s**	0.0445 s
16	20	**92.97**	87.92	81.24	**87.90**	**0.039 s**	0.0445 s
16	21	**91.95**	87.92	81.84	**87.90**	**0.041 s**	0.0445 s
16	22	**92.05**	87.92	81.32	**87.90**	**0.03 s**	0.0445 s

Table 3. Ours evaluation metrics affected by *Npositions* and *Nclusters* with our face vector of ***512*** dimensions and OpenFace

N positions	N clusters	Accuracy		Recall		Inference time	
		Ours	OpenFace	Ours	OpenFace	Ours	OpenFace
64	19	**90.03**	87.92	**89.67**	87.90	0.147 s	**0.0445 s**
64	20	**90.88**	87.92	**90.07**	87.90	0.087 s	**0.0445 s**
64	21	**91.03**	87.92	**89.01**	87.90	0.078 s	**0.0445 s**
64	22	**90.89**	87.92	**89.81**	87.90	0.095 s	**0.0445 s**
64	23	**91.78**	87.92	**89.48**	87.90	0.095 s	**0.0445 s**
32	19	**92.62**	87.92	84.95	**87.90**	**0.044 s**	0.0445 s
32	20	**92.50**	87.92	83.78	**87.90**	0.052 s	**0.0445 s**
32	21	**92.77**	87.92	85.07	**87.90**	0.054 s	**0.0445 s**
32	22	**92.21**	87.92	84.42	**87.90**	0.067 s	**0.0445 s**
32	23	**92.52**	87.92	84.36	**87.90**	0.055 s	**0.0445 s**
16	19	**93.94**	87.92	79.02	**87.90**	0.05 s	**0.0445 s**
16	20	**93.89**	87.92	79.78	**87.90**	0.049 s	**0.0445 s**
16	21	**93.47**	87.92	80.50	**87.90**	**0.044 s**	0.0445 s
16	22	**93.57**	87.92	80.83	**87.90**	**0.039 s**	0.0445 s
16	23	**93.68**	87.92	80.49	**87.90**	0.046 s	**0.0445 s**

Table 4. Ours evaluation metrics affected by *Npositions* and *N clusters* with our face vector of ***2048*** dimensions and OpenFace

N positions	N clusters	Accuracy		Recall		Inference time	
		Ours	OpenFace	Ours	OpenFace	Ours	OpenFace
128	32	**93.52**	87.92	**91.59**	87.90	0.175 s	**0.0445 s**
64	32	**93.40**	87.92	**90.70**	87.90	0.131 s	**0.0445 s**
32	32	**94.02**	87.92	**88.56**	87.90	0.1 s	**0.0445 s**

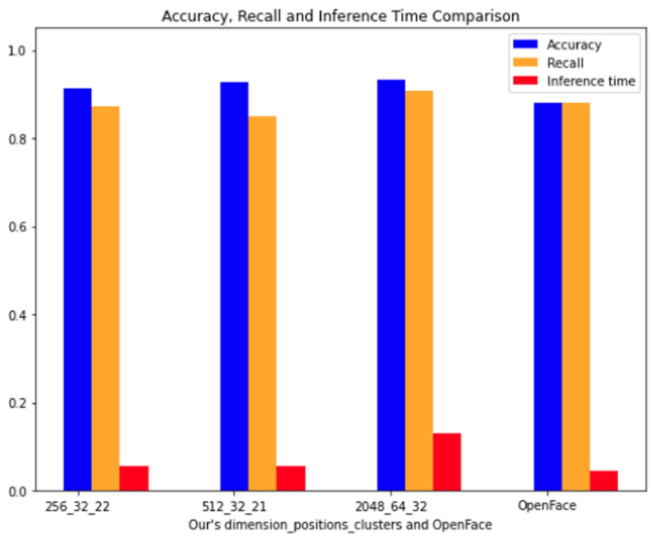


Fig. 6. Accuracy, recall and inference time comparison

More specifically, in Table 4 both our accuracy and recall metrics are far better than the OpenFace, but the time for face retrieval is not as good for real-time application. Besides, in Table 2 and Table 3, we partition it into 3 main parts to demonstrate our performance that includes: 64 positions, 32 positions, 16 positions with a range of [19–22] clusters, an addition cluster of 23 in Table 3. With group of 64 positions, there is every likelihood that ours overshadows the other with greater accuracy and recall, but the search time gets an average of 0.09 s per query. More balanced in the second group, our pipeline acts more efficiently with a significant improvement of accuracy and marginally lower in the other metrics. Last but not least, the last group come faster search time, much more precise and reach a peak of an approximate accuracy of 94%; however, the recall metric seems to be quite modest.

Taking account into Table 2 and Table 3, all in all, the second one definitely showcases more effectively with higher accuracy and recall, but the response time is a little bit slower than the other one. Referring to Fig. 6, we construct a bar chart of accuracy, recall and searching time that chooses the best numbers of positions, clusters for the encoding algorithm of different dimensional face representation vectors.

6 Conclusion and Future Works

In this paper, we have proposed a new approach for an end-to-end facial recognition application with full pipeline for both development and deployment. As shown in evaluation analysis above, our pipeline acquires an impressive prediction accuracy when facing with a very challenging dataset, which help solve the problem related to the dearth of face data. Also, the proposed pipeline has resulted in very quick prediction response time in real-time application. Furthermore, by applying a vector-to-string token algorithm, we can train the model directly in computers without the need of GPU, which means the cost for expensive physical devices needed for training purpose could be reduced.

Finally, instead of using a deep learning model for identifying faces, ES is leveraged for better storing, creation, and retrieval of face identity, thus the online learning problems in face recognition apps are also tackled. AQ5

In the future, our tendency research is finding a solution for enhancing accuracy of the vector-to-string tokens algorithm to get even better face recognition results.

References

1. Almabdy, S., Elrefaei, L.: Deep convolutional neural network-based approaches for face recognition. Appl. Sci. **9**, 4397 (2019). https://doi.org/10.3390/app9204397
2. Amos, B., Ludwiczuk, B., Satyanarayanan, M.: OpenFace: a general-purpose face recognition library with mobile applications. Technical report CMU-CS-16-118, CMU School of Computer Science (2016)
3. Bradski, G.: The OpenCV library. Dr. Dobb's J. Softw. Tools (2000)
4. Cao, Q., et al.: VGGFace2: A dataset for recognising faces across pose and age (2018). arXiv: 1710.08092 [cs.CV]
5. Chowdhry, D.A., et al.: Smart security system for sensitive area using face recognition. In: 2013 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET), pp. 11–14 (2013)
6. Django Contributors. Django 3.1 (2020). https://www.djangoproject.com/
7. DVC Contributors. Iterative, DVC: Data Version Control - Git for Data & Models (2020). https://doi.org/10.5281/zenodo.012345
8. Elasticsearch Contributors. Function Score query 6.8 (2019). https://www.elastic.co/guide/en/elasticsearch/reference/6.8/query-dslfunction-score-query.html
9. Elasticsearch Contributors. Rescoring 6.8 (2019). https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-request-rescore.html
10. Tensorflow Contributors. Tensorflow Serving 6.8 (2019). https://github.com/tensorflow/serving
11. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), vol. 1, pp. 886–893 (2005)
12. Deb, D., Nain, N., Jain, A.K.: longitudinal study of child face recognition (2017). arXiv: 1711.03990 [cs.CV]
13. Deng, J., et al.: ArcFace: additive angular margin loss for deep face recognition (2019). arXiv: 1801.07698 [cs.CV]
14. Deng, J., et al.: RetinaFace: single-stage dense face localisation in the wild (2019). arXiv: 1905.00641 [cs.CV]
15. Karl Pearson, F.R.S.: LIII. On lines and planes of closest fit to systems of points in space. London Edinburgh Dublin Philos. Mag. J. Sci. **2**(11), 559–572 (1901). https://doi.org/10.1080/14786440109462720
16. Girshick, R., et al.: Rich feature hierarchies for accurate object detection and semantic segmentation (2014). arXiv: 1311.2524 [cs.CV]

17. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
18. He, K., et al.: Deep residual learning for image recognition (2015). arXiv: 1512.03385 [cs.CV]
19. Hearst, M.A., et al.: Support vector machines. IEEE Intell. Syst. Their Appl. **13**(4), 18–28 (1998)
20. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications (2017). arXiv: 1704.04861 [cs.CV]
21. King, D.E.: Dlib-Ml: a machine learning toolkit. J. Mach. Learn. Res. **10**, 1755–1758 (2009). ISSN: 1532-4435
22. Komulainen, J., Hadid, A., Pietikainen, M.: Context based face anti-spoofing, pp. 1–8, September 2013. https://doi.org/10.1109/BTAS.2013.6712690
23. Huang, G.B., Learned-Miller, E.: Labeled faces in the wild: updates and new reporting procedures. Technical report UM-CS-2014-003. University of Massachusetts, Amherst, May 2014
24. Li, H., et al.: A convolutional neural network cascade for face detection. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5325–5334 (2015)
25. Lienhart, R., Maydt, J.: An extended set of Haar-like features for rapid object detection. In: Proceedings of International Conference on Image Processing, vol. 1, p. I (2002)
26. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2. ISBN 978-3-319-46447-3
27. Mu, C., et al.: Towards practical visual search engine within elasticsearch (2019). arXiv: 1806.08896 [cs.CV]
28. Ng, H.-W., Winkler, S.: A data-driven approach to cleaning large face datasets. In: 2014 IEEE International Conference on Image Processing, ICIP 2014, pp. 343–347, January 2015. https://doi.org/10.1109/ICIP.2014.7025068
29. Owayjan, M., et al.: Face recognition security system, December 2013
30. Parmar, D., Mehta, B.: Face recognition methods & applications. Int. J. Comput. Technol. Appl. **4**, 84–86 (2014)
31. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
32. Rima, S., et al.: Smart security surveillance using IoT, pp. 659–663, August 2018. https://doi.org/10.1109/ICRITO.2018.8748703
33. Sahoo, D., et al.: Online deep learning: learning deep neural networks on the fly (2017). arXiv: 1711.03705 [cs.LG]
34. Satish, A., Devarajan, N.: Preprocessing technique for face recognition applications under varying illumination conditions. Glob. J. Comput. Sci. Technol. Graph. Vis. **12**, 13–18 (2012)
35. Schroff, F., Kalenichenko, D., Philbin, J.: FaceNet: a unified embedding for face recognition and clustering. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015. https://doi.org/10.1109/cvpr.2015.7298682. http://dx.doi.org/10.1109/CVPR.2015
36. Shi, Y., Otto, C., Jain, A.K.: Face clustering: representation and pairwise constraints. IEEE Trans. Inform. Forensics Secur. 13(7), 1626–1640 (2018). https://doi.org/10.1109/tifs.2018.2796999. https://dx.doi.org/10.1109/TIFS.2018
37. Taigman, Y., et al.: DeepFace: closing the gap to human-level performance in face verification, September 2014. https://doi.org/10.1109/CVPR.2014.220

38. Tang, X., et al.: PyramidBox: a context-assisted single shot face detector (2018). arXiv: 1803.07737 [cs.CV]
39. Tolba, A., El-Baz, A., El-Harby, A.: Face recognition: a literature review. Int. J. Signal Process. **2**, 88–103 (2005)
40. Vikram, K., Padmavathi, S.: Facial parts detection using Viola Jones algorithm. In: 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 1–4 (2017)
41. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features, vol. 1, p. I-511, February 2001. ISBN: 0-7695-1272-0. https://doi.org/10.1109/CVPR.2001.990517
42. Wang, H., et al.: CosFace: large margin cosine loss for deep face recognition (2018). arXiv: 1801.09414 [cs.CV]
43. Wang, M., Deng, W.: Deep face recognition: a survey (2018). arXiv: 1804.06655 [cs.CV]
44. Wang, Y., Yao, Q.: Few-shot learning: a survey. CoRR abs/1904.05046 (2019). arXiv: 1904.05046. http://arxiv.org/abs/1904.05046
45. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometr. Intell. Lab. Syst. **2**(1), 37–52 (1987). Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists. ISSN 0169-7439. https://doi.org/10.1016/0169-7439(87)80084-9. http://www.sciencedirect.com/science/article/pii/0169743987800849
46. Wright, J., et al.: Robust face recognition via sparse representation. IEEE Trans. Pattern Anal. Mach. Intell. **31**, 210–227 (2009). https://doi.org/10.1109/TPAMI.2008.79
47. Yang, H., et al.: An empirical study of recent face alignment methods, November 2015. https://doi.org/10.13140/RG.2.1.4603.8484
48. Yang, J., et al.: Nuclear norm based matrix regression with applications to face recognition with occlusion and illumination changes. IEEE Trans. Pattern Anal. Mach. Intell. **39**(1), 156–171 (2017)
49. Yi, D., et al.: Learning face representation from scratch (2014). arXiv: 1411.7923 [cs.CV]
50. Zhang, K., et al.: Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Process. Lett. **23**, 1499–1503 (2016). https://doi.org/10.1109/LSP.2016.2603342

Author Queries

Chapter 32

Query Refs.	**Details Required**	**Author's response**
AQ1	This is to inform you that corresponding author has been identified as per the information available in the Copyright form.	
AQ2	Please check and confirm if the authors given and family names have been correctly identified.	
AQ3	Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city and country names "Hanoi, Vietnam" in second affiliation. Please check and confirm if the inserted city and country names are correct. If not, please provide us with the correct city and country names.	
AQ4	Please check and confirm if the inserted citation of Fig. 3 is correct. If not, please suggest an alternate citation.	
AQ5	Kindly provide the volume number and page range for Ref. [3], if applicable.	