Dat Van Nguyen
Son Trung Nguyen
Thi Hong Anh Pham
Van Toan Pham
Thao Thu Hoang
Ta Minh Thanh

HYBRID END-TO-END APPROACH INTEGRATING ONLINE LEARNING WITH FACE-IDENTIFICATION SYSTEM

Abstract *Facial recognition has been one of the most intriguing and exciting research topics over the last few years. It involves multiple face-based algorithms such as facial detection, facial alignment, facial representation, and facial recognition. However, all of these algorithms are derived from large deep-learning architectures, leading to limitations in development, scalability, accuracy, and deployment for public use with mere CPU servers. Also, large data sets that contain hundreds of thousands of records are often required for training purposes. In this paper, we propose a complete pipeline for an effective face-recognition application that requires only a small data set of Vietnamese celebrities and a CPU for training, solving the problem of data leakage, and the need for GPU devices. The pipeline is based on the combination of a conversion algorithm from face vectors to string tokens and the indexing & retrieval process by Elasticsearch, thereby tackling the problem of online learning in facial recognition. Compared with other popular algorithms on the same data set, our proposed pipeline not only outperforms the counterpart in terms of accuracy but also delivers faster inference, which is essential to real-time applications.*

Keywords facial recognition, visual search engine, end-to-end applications, online learning, ElasticSearch (ES)

Citation Computer Science 24(2) 2023: 141–161

1. Introduction

1.1. Overview

Thanks to the rapid development of technology, facial-recognition systems are becoming more and more robust. Many companies all over the world are paying great attention to and investing in facial recognition technology for their authentication systems instead of using traditional verification methods such as fingerprints or irises. Moreover, the applications of the technology in every aspect of life are diverse [29,39], such as security [28], the Internet of Things and mobile systems [3], real-time identification systems [6], biometric systems based on motion detection and facial features [1].

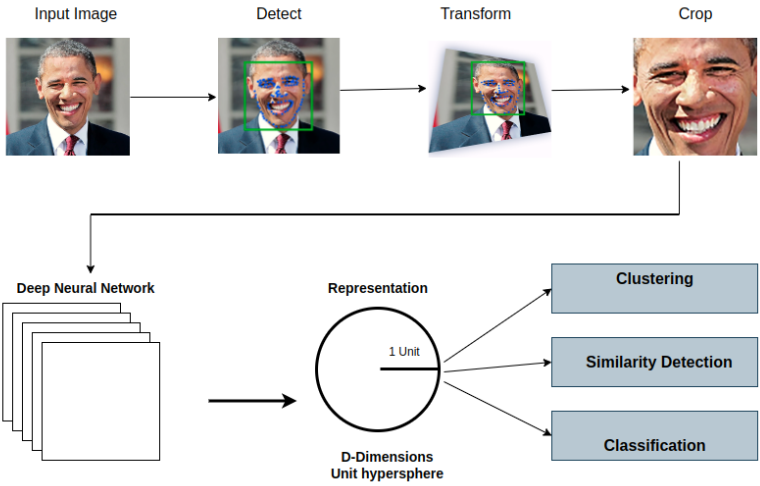

Figure 1. Overview of full pipeline for face-recognition applications

It is no wonder that an end-to-end pipeline for facial-recognition applications is comprised of complex underlying algorithms: the extraction of frontal human faces from given images (called face detection [14, 23, 25, 51]), the alignment (optional) of face positions [48], face representation in the form of numeric vectors [15, 44], and distinguishing faces [47] based on the representative vectors from the previous step [40]. A critical review of these algorithms is listed below:

- Since the 2000s, face detection has had a lot of different approaches; however, they have all been inadequate in terms of their accuracy and/or speed. A big innovation came in 2001 when Viola and Jones invented the Haar-based cascade classifier [42], which was continuously improved in 2002 by Lienhart and Maydt [24]. As a result, the algorithm became faster and could be run in real-time with 95% accuracy on a challenging data set. By 2010, face recognition had reached state-of-the-art accuracy with the explosion of deep learning [10, 37].

- Attaining facial-feature vectors after performing face detection is greatly essential to the recognition step. Algorithms [9,34] are examples of conversions from images of cropped faces to vectors of specified dimensions to denote the most crucial facial characteristics.
- For identity recognition, some approaches have become well-known, such as those that are based on a deep-learning architecture (like [2, 8, 34]), with the concept of sparse representation [47], basing on clustering [35], or cosine loss (proposed by Hao Wang in [43]).

  A common full pipeline of facial recognition would be referred to in Figure 1.

1.2. Challenging issues

All of the algorithms mentioned above have proven their strength and made significant contributions to the research area of face-recognition applications; however, there are still some issues that need to be worked out. First of all, it can be easily seen that most of these methods only make up one part of the full pipeline. Second, the adoption of deep models demonstrates more-impressive accuracy than the other approaches, but these models need to be trained on an extremely large and diverse data set with a size of up to millions of images; hence, they seem to be too slow for the development, deployment, and management of applications (which comes at the expense of high-priced physical devices). As a result, the increasing demand for GPU servers for training and deployment has been even more ubiquitous than ever. Another shortcoming is that the existing online learning problem [32] makes these models subject to the requirement of periodic training in order to maintain a system's accuracy whenever new faces are added. Furthermore, there has been a very little body of research conducted on the topic of complete pipelines for face recognition, which totally deserves more attention.

1.3. Our contributions

Our contributions are summarized as follows:

1. With all of the drawbacks discussed above, we have proposed a new approach to an end-to-end facial-recognition application in this paper in the form of a complete pipeline that consists of development, deployment, and model version management.
2. As compared to other well-known methods, our pipeline acquires an impressive prediction accuracy with a very challenging data set, which is becoming essential amid all of the difficulties that are faced during the collection of face data. The pipeline is also able to bring out a very quick response time, making it practicable in real-time applications.
3. Moreover, the cost for the necessary physical devices is reduced exponentially by applying a vector-to-string token algorithm so we can train and release the model directly on CPU servers.

4. Instead of using a deep-learning model for identifying faces, Elasticsearch (ES) is leveraged for storing, creating, and retrieving a 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. Some data pre-processing and re-balancing methods are shown in Section 3. Section 4 discusses the proposed methods of facial-recognition systems, and experimental results are presented in Section 5. Finally, our conclusions and future works are in Section 6.

2. Related works

2.1. Face detection

Face detectionThis means that determining the location and size of a human face in a digital image is a fundamental step for many face-related technologies. There are a variety of face-based algorithms that take face detection as the foundation for acquiring adequate accuracy, such as face verification [9], face recognition [49], and face anti-spoofing [22]. So, the purpose of a face-detection algorithm is to improve the accuracy of these algorithm by eliciting only frontal faces as the algorithm's inputs. To date, some adaptations of deep-learning architectures that use the concept of a convolutional neural network (CNN) for this step (like MTCNN [51], Cascade CNN [23], R-CNN [14], SSD [25], *etc.*) have achieved remarkable progress.

2.2. Face representation

Face representation (in other words, facial-features extraction) is the process of encoding raw facial images into a continuous vector representation in high-dimensional feature space [44]. Traditionally, facial features were extracted manually by design patterns such as edges, lines, a four-rectangle features in Viola Jones's algorithm [41] or grids of histograms of oriented gradient (HOG) descriptors [7].

Modernly, statistical facial-feature extraction has been performed automatically and more efficiently (in terms of both time and feature quality [44]) through convolutional neural networks (CNN), a class of deep neural networks [15]. Allowing for spatial feature preservation, CNNs are suitable for learning feature embeddings for a 2-D topology data type, including images and facial pictures [15]. Using facial embeddings that are learned by CNNs out-performs most traditional methods in several downstream tasks, including face recognition and face verification [44].

2.3. Principal component analysis

PCA (principal component analysis) [30] is a dimensionality-reduction technique that was created in 1901 by Karl Pearson. This method makes use of the recognition of statistical design to reduce dimensionality and extract features.

This approach has been used in various applications since its appearance, such as handwriting recognition, neuroscience, quantitative finance, and image compression.

2.4. FaceNet architecture in OpenFace

FaceNet [34] was developed in 2015 by Google researchers for the face-recognition task. Essentially, a CNN is responsible for extracting the features of a face image. The CNN training was performed on large data sets (vggface, MS-Celeb-1M). The key feature of FaceNet is that it uses the triplet loss function to minimize the distances between similar faces and maximize the distances between 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, and α is the margin between positive and negative pairs.

3. Data set

Our proposed model is implemented on the VN-Celeb[1] data set that consists of images of Vietnamese celebrities that were collected from the Viet Nam Wikipedia website[2].

The data set contains 24,125 images that belong to 1020 famous Vietnamese people (1020 classes). More specifically, the average number of images per 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 data set is very challenging in terms of both size and each class' proportion. To elaborate:

- numbers of images of each class fluctuate in a wide range (severely imbalanced data set);
- size of data set is small as compared to other face-recognition data sets;
- some problems such as lighting conditions, face poses, and picture quality affect recognition accuracy.

First, the image's proportion between classes is imbalanced. Table 1 illustrates the percentage of the classes that are classified by the range of the number of photos. We can see that the proportions of the classes in two ranges ([2, 5] and [50, 105]) were only 5 and 3 %, respectively; meanwhile, the classes that belonged to scope [30, 50] accounted for 27 %. Especially, classes that had [5, 30] elements showed the highest percentage (65%).

Table 1
Image proportion

No. of image	[2, 5)	[5, 30)	[30, 50)	[50, 105]
Percentage	5%	65%	27%	3%

[1]https://drive.google.com/drive/folders/1I3KXcGpmm6zpw_y07p-7wIKt5K08iOgc
[2]https://vi.wikipedia.org/

As one can see in Figure 2, the distribution of the images in each class is extremely disproportionate, as the numbers of photos stretched widely (from 2 to 105 per class). Second, the VN-Celeb data set consisted of 24,125 images; on the other hand, the number of classes was 1020. In other words, this task can be classified as few-shot learning [45] – generalizing from a few training examples.

Another issue is the variations of data set images in lighting conditions, poses, 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.

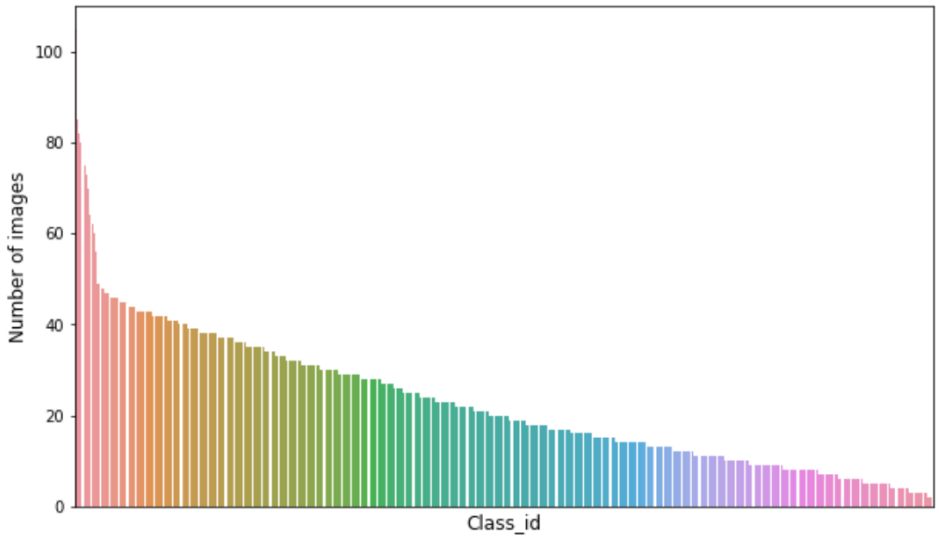

Figure 2. Number of images in each class

4. Proposed model

In this section, the proposed pipeline will 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 data set was challenging. The size was only 24.1k images with 1020 classes, which was much smaller and more imbalanced as compared to other data sets such as FaceNet's [34] private data set (with 100–200M), DeepFace [36] (using a private data set with 4.4M images), OpenFace [3] (trained on combined CASIA-WebFace data set [50] and FaceScrub [27]), and so on. Therefore, before a data set can be passed into the face-embedding model to obtain representing vectors for training purposes in the encoding phase, data cleaning, re-balancing, and some other data pre-processing techniques must be performed. Our proposed pipeline application is shown in Figure 3.

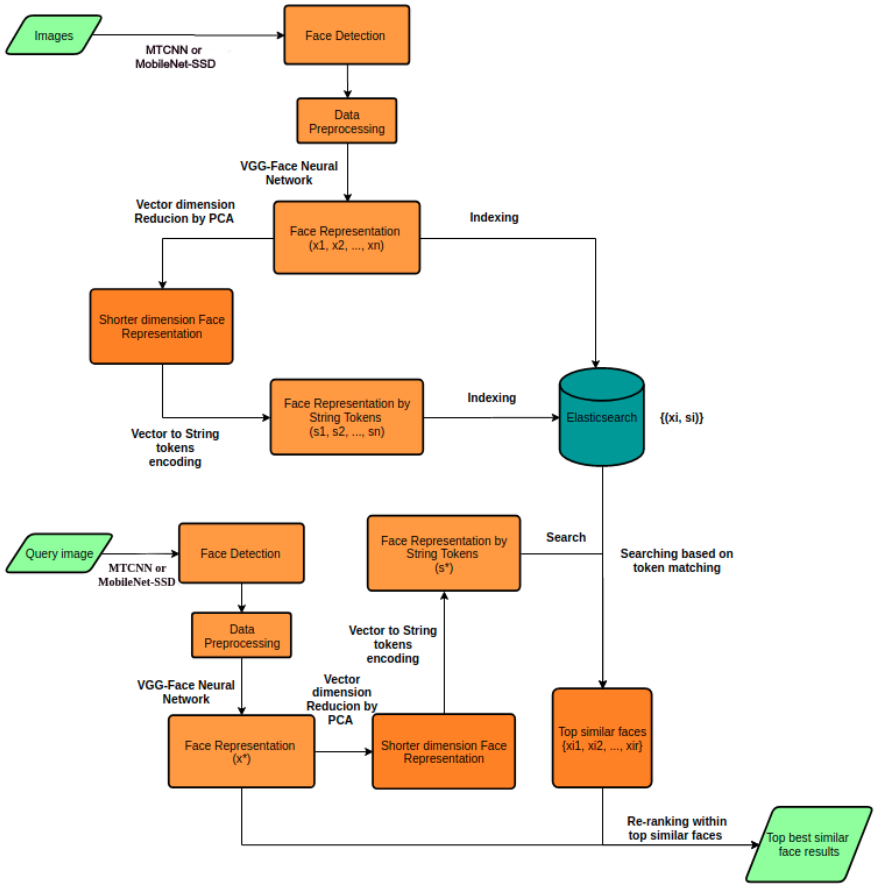

Figure 3. Workflow of application

4.1. Face detection

This phase plays an integral role in the accuracy of the whole application, and it is the foundation for subsequent steps. After carrying out experiments in various tools (Dlib-ml [21], Haar Cascades [4], MTCCN [51], *etc.*) to extract faces from the given images, we decide to take two methods for this phase; these methods are interchangeable depending on a particular spec.

First, we use SSD [25] with MobileNet as backbone on the wild data set: Labelled Faces in the Wild (LFW) Datase [19] to take advantage of the fast inference time of the MobileNet architecture [18]. Another option was to use a pre-trained library (MTCNN [51]). Although detecting faces with MobileNet-SSD gives a better performance, the bounding boxes results are not as good as those of its MTCNN counterpart. So, hinging on the purposes, we can switch between the two methods.

4.2. Data pre-processing

The data set in which we implement our pipeline was really challenging due to its size, properties, and proportions of images in each class. Therefore, pre-processing steps needed to be taken. To mitigate any adverse effects and enhance the system's accuracy, we generated more data to classes in which the numbers of images were below 20 by applying some non-geometric pre-processing techniques. These techniques are listed below:

- blurring,
- sharpening,
- smoothing,
- histogram equalization,
- gamma correction => DOG filtering => contrast equalization [33].

The data set that was obtained after pre-processing was passed through the next step in order to obtaining face representation for training in the encoding phase.

4.3. Face embeddings with VGG-Face

At this point, the embedding vectors that contained the most informative facial elements were represented in the form of numeric arrays that were extracted from a deep neural network. Based on experiments and evaluations on deep-learning architectures like FaceNet [34], VGG-Face [5], ArcFace [9], etc., VGG-Face was chosen because of its accuracy and performance running on our framework. The based model is ResNet50 [16], all fully connected layers were removed so as to convert face images into numeric representing vectors (which would be used for recognition purposes at a later stage).

4.4. Reducing face vector dimensions by PCA

The default output of VGG-Face is a 2048-dimensional vector, making it very difficult for real-time applications to run on a server without a GPU, as the computational cost is very high and it might hurt the performance of the whole framework otherwise. So, it was time for some data dimension-reduction algorithms to be implemented. An attempt at training and evaluating a list of array dimensions within a range of [256, 512, 1024] was conducted through a combination between this phase and the encoding phase (which will be explained in the next subsection). Finally, after assessing both the accuracy and performance deliberately, we decided to apply PCA [46] to compress from 2048 to 512 dimensions as the best dimension to represent the face data.

4.5. Encoding face embeddings to string tokens

In this section, we explain the reason why we decided to change the data-type for face representation. Actually, it is possible to recognize a person's identity from facial embedding vectors that are extracted from VGG-Face by using well-known methods

such as Euclidean distance, cosine similarity, and a deep-learning classification model. However, calculating similarity within all high-dimensional facial vectors to find the most similar is computationally expensive and time-consuming. Besides, another option of using deep-learning algorithms for this task was proposed, but it had a long inference time and high latency due to the complexity of the deep-model architectures for recognition (not to mention inefficiency regarding the online learning problems). In short, the application was challenging and difficult to put into real-life practice.

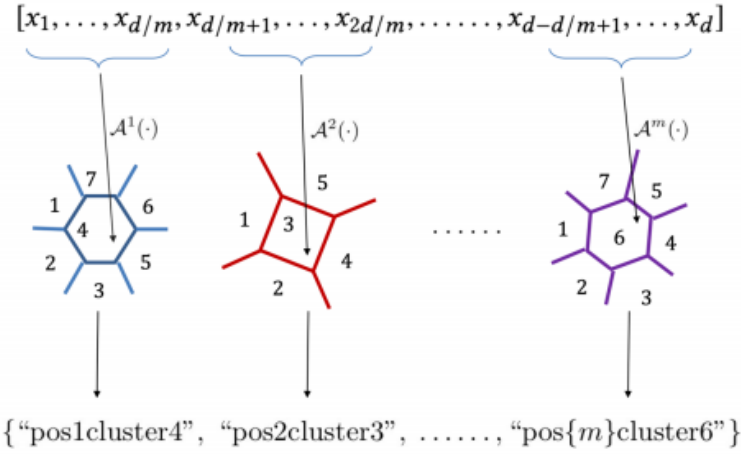

Figure 4. Illustration of subvector-wise clustering

In this paper, we utilized an encoding algorithm from [26] to convert numeric face vectors into collections of string tokens that could be retrieved faster with Elasticsearch, which was beneficial for a real-time application. The encoding algorithm was inspired by the idea of subvector-wise clustering. More specifically, with any numeric face vector $x \in \mathbb{R}^d$, we divided it into m positions (as below, cf. Fig. 4):

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

Considering m positions as m subvectors $= \{x^1, x^2, ..., x^m\}$ and $P^i := \{x_1^i, x_2^i, ..., x_n^i\}$, where $i = \{1, ..., n\}$ as the collection of the i^{th} subvectors merged from each position in the whole data set. We applied the k-means algorithm to cluster each P^i into k clusters ($k > 1$). By doing this, we were able to encode an original numeric face vector into string tokens. For example: [“$pos1cluster2$,” “$pos2cluster\{k\}$,”..., “$pos\{m\}cluster1$”] is an example vector that was encoded from an origin numeric face vector by applying this encoding algorithm with m positions and k clusters.

4.6. Data indexing and searching method

In this subsection, we provide insights into the data-indexing progress and the method for retrieving face identities. We will elaborate in two parts (data indexing, and searching method) as follows.

4.6.1. Data indexing

After getting string tokens from the previous encoding phase, we need to store them for face retrieval afterward. We are aware that retrieving face recognition from string representation is much faster than from high-dimensional numeric vectors. Therefore, in order to achieve an optimal string-matching mechanism, we leverage the concept of an inverted-index-based search engine from Elasticsearch (ES). To index data into ES, we construct a JSON format that included numeric vectors and string tokens together with some other properties of a person such as name, age, address, image path, and so on. Such a JSON format was depicted for each face for 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,
    "address": address,
    "gender": gender,
    "email": email,
    ...
}
```

In the end, we had an array of JSONs with a length that equaled that of the training data set. Then, we simply used available ES API functions to index the data into the Elasticsearch server.

4.6.2. Searching method

In this step, we follow the steps of the implementation that Cun (Matthew) Mu [26] did in his paper (but for face data).

With any query image, we applied the same steps as the training steps that were explained above to attain a string token $\hat{s}$; then, we used it for searching. The top r-similar faces that are obtained rely on the overlap between string token set $\hat{s}$ and the ones stored on the ES server $\{s_1, s_2, ..., s_n\}$:

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

ES provides us with RESTful API for searching conveniently; all we need to do is to build a JSON-encoded request body that would instruct the ES server to compute

and then return the visual search results. The format for the request body is described below:

```
string_tokens_chunks = list()
for i in range(num_positions):
    sub_field = {
        "filter": {
            "term": {
                "string_token": string_tokens[i]
            }
        },
        "weight": 1
    }
    string_tokens_chunks.append(sub_field)

request_body = {
    "size": r,
    "query": {
        "function_score": {
            "functions": string_tokens_chunks,
            "score_mode": "sum",
            "boost_mode": "replace"
        }
    },
    "rescore": {
        "window_size": 5,
        "query": {
            "rescore_query": {
                "function_score": {
                    "script_score": {
                        "script": {
                            "lang": "custom_scripts",
                            "source": "euclidean_distance",
                            "params": {
                                "vector_field": "image_actual_vector",
                                "query_vector": [0.0212, 0.0512, 0.0337, ...]
                            }
                        }
                    },
                    "boost_mode": "replace"
                }
            },
            "query_weight": 0,
            "rescore_query_weight": 1
        }
    }
}
```

In the JSON format above, we establish two prime parts inside for obtaining the sorted results from ES. The first part ("query") is a function score query [12], responsible for retrieving r faces of which string tokens are the most similar to $\hat{s}$. The second part ("rescore") is a custom rescore function [13] provided by ES. It helps improve precision by reordering just the top r from the first part using Euclidean distance, instead of applying this costly algorithm to all faces in the database. 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, and conveniently. At this point, we have had several trained models from face detection and face embedding to a vector-to-string encoding model that needs to served for inference time whenever it receives requests from the face-recognition server. With these two first deep models, we convert these models into the TensorFlow PB format instead of storing the physical file and loading it directly on the server; then, we use the TensorFlow serving API [38] to serve and manage in a separate server. Regarding the last one, the vector-to-string encoding model (data version control [20]) will be leveraged.

4.8. Django framework for development and deployment

To this point, a framework is necessary in order to make our application available to the community by providing some open API functions. Our application was developed and deployed on the Django framework [11] (one of the most popular frameworks) using Python.

5. Experiments

In this section, we compare our proposed pipeline to another well-known framework (OpenFace [3]) on the VN-celeb data set in order to prove the validity of our proposed pipeline. 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 were conducted on a computer with an IntelCore i5-4460 CPU @3.2GHz, 16GB of RAM, and a 256-TB SSD hard disk. The models were implemented with the Python Version 3.6.8 environment.

5.2. Evaluation method

5.2.1. Accuracy

Accuracy is the most important metric that is used to evaluate the efficiency and generality of almost every model. It describes how well a model performs by providing

a ratio between the number of correct predictions and the total elements of the testing set. The formula is shown below:

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

where N is the length of the test set, c is the number of classes that need to be predicted, n_{ci} provides 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 $\hat{y_j}$ that returns "1" if $y_j^* = \hat{y_j}$ and "0" if otherwise.

5.2.2. Recall

Recall is the fraction of the number of positive predictions of classes to its actual positive. Recall is defined as below:

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

As one can see in the recall formula, c is the number of classes that need to be predicted, n_{pi} illustrates the count of positive predictions, n_{ci} shows the actual positive of each class, and $E(y_j^*, \hat{y_j}) == 1$ indicates one correct prediction. As one can see in the recall formula, 'true positive' is assigned to those data points that are labeled as positive that are truly positive, and 'false negative' is assigned to those data points that are labeled as negative that are actually positive.

5.3. Experimental analysis

In this section, we have re-implemented and trained OpenFace [3] on the data set above to compare with our proposed algorithm to prove the efficiency of our proposed pipeline. Also, we divided the data into two sets for training and testing at rates of 80% and 20%, respectively.

5.3.1. OpenFace with VN-celeb data set

In order to allow for a fair comparison, we re-implement OpenFace [3] following the same steps as we do with our algorithm. It is noteworthy that the data set 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 pre-processing as indicated in Section 4.2 before feeding into FaceNet's triplet loss [34] to train for a total of 150 epochs based on these empirical experiments. Since its weights were trained in 500k images, instead of initializing for the whole deep architecture, we perform fine-tuning techniques by resetting the weights of some last FaceNet's layers and freezing all of the remaining layers; then, we warm up the model in 30 epochs. After this, we unfreeze and train the whole model in the last 120 epochs.

After the training is completed, we have a collection of numeric vectors in 128 dimensions from the trained model that was generated earlier that characterize each face's properties. As mentioned in [3], the final step is to put these vectors through the support vector machine (SVM) [17] from Scikit-learn [31] as the classifier for the distinction between each individual.

5.3.2. Our proposed pipeline

To gain our full pipeline, we follow the steps that are described in Section 4. We also ignore the face-detection step for the reason that was mentioned above. In the next step, the data pre-processing techniques in Section 4.2 are applied; the output is then passed through VGG-Face [5] with all fully connected layers eliminated to obtain face-representation vectors of 2048 dimensions as the 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 [46] to reduce the number of dimensions from 2048 to 512 to select the most useful principal face components. At this point, it is to encode numeric face vectors to string tokens; following Section 4.5, we divide all face vectors of 512 dimensions in the training data set 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 data set. Then, we apply a separate k-means algorithm from the Scikit-learn [31] library to each ith collection vector P^i. The array of trained k-means models are saved for future inference.

Using k-means models that have been trained to obtain string tokens, we combine these tokens with some other personal properties such as name, address, phone number, division, nationality, email, numeric face vectors, *etc.* in order to build a JSON object (as in Section 4.6.1) for indexing the data into the ES server.

For inference, we merely need to build a JSON request body (as in Section 4.6.2) and make use of ES's searching API for face retrieval to obtain the individual identity. Finally, the top-five similar faces will be returned. The individual identity is the name field of a record with the highest score that is calculated by the ES score function.

Particularly, all deep-learning and k-means models are protected and managed by the TensorFlow server and DVC, respectively, as we describe in Section 4.7. Finally, we develop and deploy our full pipeline application with the Django framework[3].

5.4. Experimental results and comparison

In this section, we compare the results of our proposed method to those of OpenFace [3], which we implemented in Section 5.3.1.

According to results' statistics and comparisons, it can be concluded that our proposed application brought about many benefits to face-recognition applications. We show some statistic tables and charts below.

Tables 2, 3, and 4 represent the statistic tables that we built for the purpose of comparing the accuracy, recall, and inference time, respectively, between OpenFace [3]

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

and our pipeline using numeric face vectors of different dimensions within a range of [256, 512, 2048] for the vector-to-string token algorithms. These also demonstrate the impact of the numbers of positions and clusters in the encoding algorithm on the evaluation metrics.

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

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

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

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

Table 4
Our evaluation metrics affected by *N positions* and *N clusters* with our face vector of 2048 dimensions and OpenFace

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

It is important to note that our accuracy completely outperformed that of the OpenFace counterparts throughout all row records in the three tables with relative equivalences in two recall columns; some of ours are higher (especially in Tabs. 3 and 4). Moreover, by encoding high-dimensional vectors into the string tokens, we have the ability to gain the same performance as OpenFace did. The OpenFace lib just takes the face vectors of 128 dimensions; meanwhile, we achieved a comparable inference time with far higher dimensional vectors that are obviously comprised of more facial features.

More specifically, in Table 4, both our accuracy and recall metrics are far better than OpenFace, but the time for face retrieval was not as good for real-time application. Besides, in Tables 2 and 3, we partitioned it into three main parts to demonstrate our performance; this included 64 positions, 32 positions, and 16 positions within a range of [19, 20, 21, 22] clusters (and the additional cluster of 23 in Tab. 3). With a group of 64 positions, there is every likelihood that ours overshadows the others with greater accuracy and recall; however, the search times took an average of 0.09 s per query. In the second group of 32 positions, our pipeline performed more efficiently with a significant improvement in accuracy.

As for other metrics (Recall and Inference time), OpenFace outperformed our pipeline by a small margin The last group (16 positions) outstood by fast searching time and precision which reached a peak of an approximate accuracy of 94%; however, the recall metric seemed to be quite modest.

Taking Tables 2 and 3 into account, the latter definitely showcased more effectively with higher accuracy and recall; however, the response time was a little bit slower than the former. In Figure 5, we constructed a bar chart of the accuracy, recall, and searching time that chooses the best numbers of positions and clusters for the encoding algorithms of different dimensional face-representation vectors.

According to the above experimental results, our proposed method has the flexibility of choosing a number of positions and clusters in order to achieve the desired inference time and accuracy/recall.

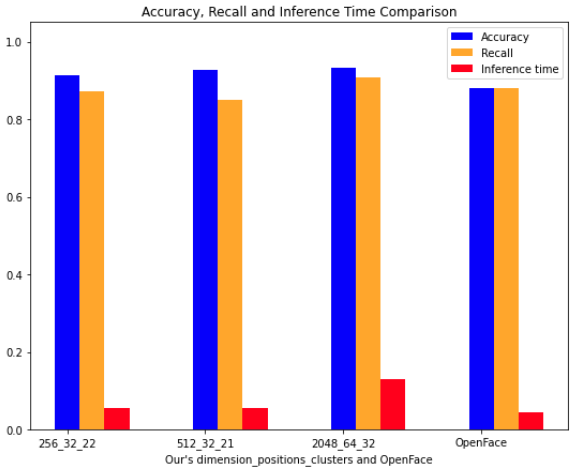

Figure 5. Accuracy, recall, and inference time comparison

6. Conclusion and future works

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

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

In the future, our tendency toward research will be to find a solution for enhancing the accuracy of the vector-to-string token algorithm to get even better face-recognition results.

References

[1] Akter S., Sima R.A., Ullah M.S., Hossain S.A.: Smart Security Surveillance using IoT. In: *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 659–663, 2018. doi: 10.1109/ICRITO.2018.8748703.

[2] Almabdy S., Elrefaei L.: Deep Convolutional Neural Network-Based Approaches for Face Recognition, *Applied Sciences*, vol. 9, 2019. doi: 10.3390/app9204397.

[3] 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.

[4] Bradski G.: The OpenCV Library, *Dr Dobb's Journal of Software Tools*, 2000.

[5] Cao Q., Shen L., Xie W., Parkhi O.M., Zisserman A.: VGGFace2: A dataset for recognising faces across pose and age, arXiv: 171008092, 2018. doi: 10.48550/arXiv.1710.08092.

[6] Chowdhry D.A., Hussain A., Ur Rehman M.Z., Ahmad F., Ahmad A., Pervaiz M.: 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. doi: 10.1109/CSUDET.2013.6670976.

[7] Dalal N., Triggs B.: Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893, 2005. doi: 10.1109/CVPR.2005.177.

[8] Deb D., Nain N., Jain A.K.: Longitudinal Study of Child Face Recognition, arXiv: 171103990, 2017. doi: 10.48550/arXiv.1711.03990.

[9] Deng J., Guo J., Xue N., Zafeiriou S.: ArcFace: Additive Angular Margin Loss for Deep Face Recognition, arXiv: 180107698, 2019. doi: 10.48550/arXiv.1801.07698.

[10] Deng J., Guo J., Zhou Y., Yu J., Kotsia I., Zafeiriou S.: RetinaFace: Single-stage Dense Face Localisation in the Wild, arXiv: 190500641, 2019. doi: 10.48550/arXiv.1905.00641.

[11] Django 3.1, 2020. https://www.djangoproject.com/.

[12] Elasticsearch: Function Score query 6.8, 2019. https://www.elastic.co/guide/en/elasticsearch/reference/6.8/query-dsl-function-score-query.html.

[13] Elasticsearch: Rescoring 6.8, 2019. https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-request-rescore.html.

[14] Girshick R., Donahue J., Darrell T., Malik J.: Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. doi: 10.48550/arXiv.1311.2524.

[15] Goodfellow I., Bengio Y., Courville A.: *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[16] He K., Zhang X., Ren S., Sun J.: Deep Residual Learning for Image Recognition, arXiv: 151203385, 2015. doi: 10.48550/arXiv.1512.03385.

[17] Hearst M.A., Dumais S.T., Osuna E., Platt J., Scholkopf B.: Support vector machines, *IEEE Intelligent Systems and their Applications*, vol. 13(4), pp. 18–28, 1998. doi: 10.1109/5254.708428.

[18] Howard A.G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M., Adam H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, arXiv: 170404861, 2017. doi: 10.48550/arXiv.1704.04861.

[19] 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, 2014. https://vis-www.cs.umass.edu/lfw/lfw_update.pdf.

[20] Iterative, DVC: Data Version Control – Git for Data & Models, 2020. https://github.com/iterative/dvc.

[21] King D.E.: Dlib-ml: A Machine Learning Toolkit, *The Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[22] Komulainen J., Hadid A., Pietikäinen M.: Context based face anti-spoofing. In: *2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pp. 1–8, 2013. doi: 10.1109/BTAS.2013.6712690.

[23] Li H., Lin Z., Shen X., Brandt J., Hua G.: A convolutional neural network cascade for face detection. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5325–5334, 2015.

[24] 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, pp. I–I, 2002. doi: 10.1109/ICIP.2002.1038171.

[25] Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.Y., Berg A.: SSD: Single Shot MultiBox Detector. In: *Computer Vision – ECCV 2016*, pp. 21–37, 2016. doi: 10.1007/978-3-319-46448-0_2.

[26] Mu C., Zhao J., Yang G., Zhang J., Yan Z.: Towards Practical Visual Search Engine within Elasticsearch, arXiv: 180608896, 2019. doi: 10.48550/arXiv.1806.08896.

[27] Ng H.W., Winkler S.: A data-driven approach to cleaning large face datasets, *2014 IEEE International Conference on Image Processing, ICIP 2014*, pp. 343–347, 2015. doi: 10.1109/ICIP.2014.7025068.

[28] Owayjan M., Dergham A., Haber G., Fakih N., Hamoush A., Abdo E.: Face Recognition Security System. In: *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 2013)*, 2013.

[29] Parmar D.N., Mehta B.B.: Face Recognition Methods & Applications, *International Journal of Computer Technology and Applications*, vol. 4, pp. 84–86, 2014.

[30] Pearson K.F.R.S.: LIII. On lines and planes of closest fit to systems of points in space, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2(11), pp. 559–572, 1901. doi: 10.1080/14786440109462720.

[31] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E.: Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] Sahoo D., Pham Q., Lu J., Hoi S.C.H.: Online Deep Learning: Learning Deep Neural Networks on the Fly, arXiv: 1711.03705, 2017. doi: 10.48550/arXiv.1711.03705.

[33] Satish A., Devarajan N.: Preprocessing technique for face recognition applications under varying illumination conditions, *Global Journal of Computer Science and Technology Graphics and Vision*, vol. 12, pp. 13–18, 2012.

[34] Schroff F., Kalenichenko D., Philbin J.: FaceNet: A unified embedding for face recognition and clustering, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. doi: 10.1109/cvpr.2015.7298682.

[35] Shi Y., Otto C., Jain A.K.: Face Clustering: Representation and Pairwise Constraints, *IEEE Transactions on Information Forensics and Security*, vol. 13(7), pp. 1626–1640, 2018. doi: 10.1109/tifs.2018.2796999.

[36] Taigman Y., Yang M., Ranzato M., Wolf L.: DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014. doi: 10.1109/CVPR.2014.220.

[37] Tang X., Du D.K., He Z., Liu J.: PyramidBox: A Context-assisted Single Shot Face Detector, arXiv:180307737, 2018. doi: 10.48550/arXiv.1803.07737.

[38] Tensorflow Serving 6.8, 2019. https://github.com/tensorflow/serving.

[39] Tolba A., El-Baz A., El-Harby A.: Face Recognition: A Literature Review, *International Journal of Signal Processing*, vol. 2, pp. 88–103, 2005.

[40] Van T.P., Nguyen T.M., Tran N.N., Nguyen H.V., Doan L.B., Dao H.Q., Minh T.T.: Interpreting the Latent Space of Generative Adversarial Networks using Supervised Learning. In: *2020 International Conference on Advanced Computing and Applications (ACOMP)*, pp. 49–54, 2020. doi: 10.1109/ACOMP50827.2020.00015.

[41] 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.

[42] Viola P., Jones M.: Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001. doi: 10.1109/CVPR.2001.990517.

[43] Wang H., Wang Y., Zhou Z., Ji X., Gong D., Zhou J., Li Z., Liu W.: CosFace: Large Margin Cosine Loss for Deep Face Recognition, 2018.

[44] Wang M., Deng W.: Deep face recognition: A survey, *Neurocomputing*, vol. 429, pp. 215–244, 2021. doi: 10.1016/j.neucom.2020.10.081.

[45] Wang Y., Yao Q.: Few-shot Learning: A Survey, arXiv: 190405046v1, 2019. http://arxiv.org/pdf/1904.05046v1.

[46] Wold S., Esbensen K., Geladi P.: Principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, vol. 2(1), pp. 37–52, 1987. doi: 10.1016/0169-7439(87)80084-9.

[47] Wright J., Yang A.Y., Ganesh A., Sastry S.S., Ma Y.: Robust Face Recognition via Sparse Representation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, pp. 210–227, 2009. doi: 10.1109/TPAMI.2008.79.

[48] Yang H., Jia X., Loy C.C., Robinson P.: An Empirical Study of Recent Face Alignment Methods, 2015. doi: 10.13140/RG.2.1.4603.8484.

[49] Yang J., Luo L., Qian J., Tai Y., Zhang F., Xu Y.: Nuclear Norm Based Matrix Regression with Applications to Face Recognition with Occlusion and Illumination Changes, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39(1), pp. 156–171, 2017.

[50] Yi D., Lei Z., Liao S., Li S.Z.: Learning Face Representation from Scratch, arXiv: 1411.7923, 2014. doi: 10.48550/arXiv.1411.7923.

[51] Zhang K., Zhang Z., Li Z., Qiao Y.: Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks, *IEEE Signal Processing Letters*, vol. 23(10), pp. 1499–1503, 2016. doi: 10.1109/LSP.2016.2603342.

Affiliations

Dat Van Nguyen
e-mail: dat.nguyen@uni.lu

Son Trung Nguyen
e-mail: nguyen.trung.son@sun-asterisk.com

Thi Hong Anh Pham
email: pham.thi.hong.anh@sun-asterisk.com

Van Toan Pham
email: pham.van.toan@sun-asterisk.com

Thao Thu Hoang
email: hoang.thu.thao@sun-asterisk.com

Ta Minh Thanh
email: thanhtm@lqdtu.edu.vn

Received: 12.05.2022
Revised: 14.10.2022
Accepted: 24.10.2022