

Face Verification Expert System

Hilquias de Paiva Araújo^{1,2}

¹PPG-PO, ITA; ²FaceV Inc., fictional company, SP, Brazil

Abstract

Facial Recognition systems have become increasingly prevalent in various domains, from security and law enforcement to smart-phones access and everyday life. This journal uses existing state of the art models in Face Verification to build an expert system focused on the use case of residential building access control. We, at FaceV Inc., developed the system that can automatically define what the best model for a specific residential condominium is, given the condominium residents image database and the degree of security desired. The system also automatically creates an Elasticsearch database environment in which all the experiment and production data is saved and queried. In this journal we also discuss how we built an API to serve our clients and how they can prepare its own database to integrate with our system. Finally, we discuss some of the current approach topics that still need to be tackled in the future.

Index Terms: Face Verification, Face Recognition, Access control expert system

1 Introduction

Face Verification is a computer vision task focused on asserting whether or not two given pictures belongs to the same person. It is a well known task of the Computer Vision knowledge domain that was deeply impacted with the advent of Deep Learning models. In this project, we aim to build a Facial Verification Expert System that selects the best possible model according to a given images database and serve it via API.

1.1 Contribution

This paper has the following contributions.

- It defines an experimental planning capable of automatically selecting the best possible model among all the available ones;
- It also describes an API that can be used to serve the best model selected;
- Finally, it describes a database schema that allows the system to run its experiment and serve the model.

2 Background

Face Verification is a crucial sub-task in the field of Face Recognition systems. It plays a vital role in various applications, ranging from secure access control in smartphones to surveillance systems used to identify criminals in urban areas. Face Recognition has become an integral part of everyday life, and its applications are vast and diverse.

Our attention will be focused on access control of residential buildings, however this approach can be adapted to other domains as well.

In residential buildings, a face recognition system can be implemented to grant access to residents and other registered people. It consists of a device installed in the entrance that detects when someone gets closer to the entrance and a software that handles all the verification system, as well as a database with all the registered people. As a person gets close to a building entrance where the device is located, the system works as follows:

- The device detects the approximation of someone and triggers the camera to take a picture;
- The image taken passes through a Deep Learning model that calculates an embedding for the image;
- This embedding is then compared with all the embeddings in the registration database so that the closest image is identified;
- If the distance between the two embeddings is lower than a threshold, the system grants access to the person;
- Otherwise, the system denies the access.

As explained in [Schroff et al., 2015](#), a Deep Learning model can learn to generate a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. This mapping that represents a face image in Euclidean space is often called embedding. This embedding can be used in several tasks, such as face verification, face recognition and clustering, with an easy implementation once one has the proper embedding for each image.

We will leverage this in our system, so we can easily implement face verification in the access control of a residential building.

3 Methodology

We, at FaceV company, aim to serve a facial verification system that is flexible enough to fulfill our clients needs. As every client may have a different set of registered people with different pictures, we need to be able to serve a model that performs well for each client. Also, as each client may have a different desirable degree of security, our model selection algorithm must find the best model to operate and accomplish all client's constraints.

With these objectives in mind, we decided to use DeepFace as our model framework in order to select the best model for each client. It is a *python* package that encapsulates a set of state-of-the-art Deep Learning models specialized in face recognition task: VGG-Face, FaceNet, OpenFace, DeepFace, DeepID, ArcFace, Dlib, SFace and GhostFaceNet. With this framework, we can seamlessly switch between different models and perform an experiment to define which model best performs for a given client.

To solve the degree of security flexibility constraint, we decided to build a custom metric that evaluates model performance in a way that the client is capable to input a certain level of security, choosing the weight of an incorrect prediction that best aligns with its concerns. As we will discuss later, a facial verification model incorrect prediction can either block the entrance of a registered person or allow the entrance of an unregistered person. As one might imagine, those errors have a different security impact, as the first one only impacts user experience and the second one can severely impact building security. Because of that, we decided to give the client a possibility to choose the weight of the second error, so that the model performance evaluation would choose the model that least commits the second type of error, according to clients security degree input.

3.1 DeepFace Available Models and Embeddings

As listed above, DeepFace provides quick access to several models, each one having different embeddings properties and predefined distance thresholds. A quick summary of all models with its properties is presented in

Model	Embedding Size	Cosine Distance Threshold
VGG-Face	4096	0.68
FaceNet	128	0.40
FaceNet-512	512	0.30
OpenFace	128	0.10
DeepID	160	0.015
DLib	128	0.07
ArcFace	512	0.68
SFace	128	0.593
GhostFaceNet	512	0.65

Note that changing model's cosine distance threshold does not fit into our current scope and will be properly tackled in the future.

3.2 Database Selection and Pre Processing

As our main database, that is the source of all pictures used in the experimental planning discussed bellow, we chose Labeled Faces in the Wild [Huang et al., 2012]. It is a public benchmark for face verification task. Although the authors of this database disclaims that it is not suitable for a thorough vetting of commercial algorithms, we can use it as a source of images for our model selection experiment to take place. This way, we can select the model that best performs in LFW database and, by doing so, we can validate that our model selection system is working as expected.

It is important to note that for each client we will repeat this experimental planning, but using the clients database, so that we can select the model that would best perform in the clients data. It is also important to note that with DeepFace we are using state-of-the-art models that are thoroughly validated over time, so we do not need to worry about them being suitable to run in production.

This database contains 13233 images for 5749 people, being 1680 with at least 2 pictures and 901 with at least 3 different pictures. For our experiment to take place, we need to consider only people with at least 2 pictures, as we will better explain later.

3.3 Experimental Planning

To select the best possible model provided by DeepFace, we designed an experimental planning that follows a resampling strategy to create random paired databases, one that simulates registration database, called **registration**, and another that simulates what a day in production would be like if this face verification model would be implemented in a residential building, called **operation**. With these databases in place, we simulate a prediction using all possible pair of images, one from **registration** and one from **operation** database, and then we evaluate the model performance by verifying the correctness of the prediction.

3.3.1 Database creation With the main database already created and pre processed, we can proceed to the resampling strategy that creates random paired databases to evaluate models performances.

We follow these assumptions to build **registration** and **operation** databases:

- the residential building have N registered residents;
- each resident passes through the face verification system twice a day, when leaving and entering the building;

- there is $\lfloor N/2 \rfloor$ visitors, or unregistered service providers, that also passes through the face verification system twice a day, when entering and leaving the building.

With the assumptions above, **registration** is a database that contains N registered residents, and for each one of them we save one picture. **Operation** is a database that contains the same N registered people and $\lfloor N/2 \rfloor$ visitors, and for each one we save two different pictures. Note that in **operation** database, we have two different pictures for each person because each person goes through the face verification system twice a day, when entering and leaving the building. We simulate the fact that the real-time picture taken by the face verification system may be slightly different on each occasion.

We perform this database creation each time the resampling strategy is executed, so that we can calculate model performance over different databases, simulating production conditions. With the performances history, we can estimate each model performance's distribution and choose the model with the best performance, meaning the model with higher probability of being better in production.

3.3.2 Model evaluation With both databases in place, we can simulate a prediction using all possible pair of images, one from **registration** and one from **operation** database, by creating an exhaustive combination of images $(I_{R,i}, I_{O,i})$, with $I_{R,i}$ from **registration** database and $I_{O,i}$ from **operation** database. Since there are N registered residents and $\lfloor N/2 \rfloor$ visitors, the total combination number is equal to $N \times (N + \lfloor N/2 \rfloor)$.

All the constructed image pairs are used as inputs, one at a time, of each possible model to generate the predictions we need to assert each model's performance.

Each model prediction takes place by calculating the distance between the images embeddings. As each embedding is a vector representation of the image, it can be interpreted as a point in a hyperspace (a space with more than three dimensions) and we can calculate the distance between these two points in that hyperspace. It can be euclidean distance or, in most cases, cosine distance. The lower the distance, the bigger the chance of the images belonging to the same person, and this chance gets lower as the distance gets bigger. Thus we can choose a distance threshold below which the images are considered to belong to the same person, and belonging to different people otherwise.

A model's prediction can be characterized as:

- **true positive**: correctly predicts that both images belongs to the same person. In our case, this means that it allows the entrance of a registered person; or
- **false positive**: incorrectly predicts that both images belongs to the same person. This means that it allows the entrance of an unregistered person, raising security concerns; or
- **false negative**: incorrectly predicts that both images do not belong to the same person. This means that it blocks the entrance of a registered person; or
- **true negative**: correctly predicts that both images do not belong to the same person. This means that it blocks the entrance of an unregistered person.

A good model must perform in such a way that both false positive and false negative predictions are minimized. However, different clients may have different criteria about its security concerns. Because of that, we allow each client to decide the weight of a false positive error. This way, we can greatly penalize false positive predictions when measuring model performance and select the model that commits the least amount of errors.

To achieve that, we created a loss function that is as follows:

$$\mathcal{L} = \frac{FN + \omega \cdot FP}{TP + \epsilon} \quad (1)$$

After predicting each pair of images, we calculate the loss for each model and proceed to the next resampling step.

After all resampling rounds, we are able to calculate the mean and standard deviation of each model's error. With that in place, we select the two models with the lowest error means and perform a dependent T-test for paired samples to verify whether or not their error samples have the same expected average values. If we can reject the hypothesis that the distributions have the same expected values, we can properly determine that the model with the lowest error mean is indeed the best model to choose.

The whole experimental planning can be represented as the diagram bellow at Figure 1.

3.4 Database tool

Our database tool of choice is Elasticsearch. It allows fast queries, can be hosted in almost any machine via Docker Container instance, and is very scalable and flexible to handle different kinds of data structures. It also works as a vector database.

We organized our databases in Elasticsearch indexes as follows:

- **people** index: contains all people from the raw LFW dataset. It stores their information: person name, number of pictures and the pictures list;
- **databases** index: contains all the random databases created in the resampling strategy phase. It stores the registration and operation databases people and pictures lists, so we can access them in the prediction step of the experimental planning;
- **model** index: contains model configuration information. It stores the production model configuration: model name, distance threshold and model update timestamp. Every time an experimental planning takes place and the best model is conclusively

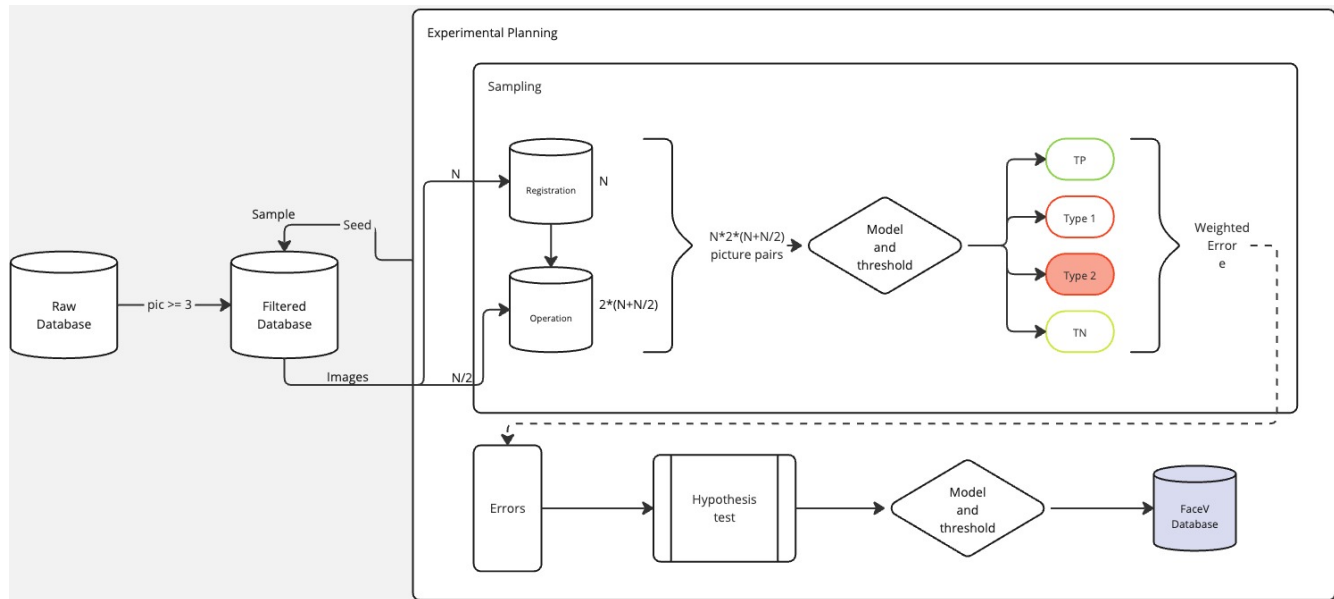


Figure 1. Experimental Planning diagram.

selected, it is stored in this database with the timestamp, so that we can always select the latest model while keeping track of our past production models;

- **prediction** index: contains all production predictions. It stores both the **base image** and **prediction image**, as well as the model prediction, the calculated distance between them, the name of the model that generated the prediction and the prediction timestamp;
- **feedback** index: contains all production feedbacks from human operators. It stores both the **base image** and **prediction image**, as well as the model prediction and human prediction, the motive of the feedback and the feedback timestamp;

We will better explain prediction and feedback in detail in the model serving section.

3.5 Model serving

Our choice to serve the model to our clients is via an API, so that we can provide quick access to the model for different clients with the necessity to maintain only a single serving instance. It also allows all clients to request model serving basically at the same time without conflicting.

This API is hosted in a Docker Container that runs in our machine, for now, and it will be hosted in a cloud machine or in a Kubernetes cluster in the future.

3.5.1 API methods

Our API has the following methods:

- **predict**: predicts if the person in one picture is the same as the other picture. It receives as arguments two images: **base image** that is the image to be compared, and **prediction image** that is the image that we want to assert if it belongs to the same person as the other one. It returns a boolean as output that is true if the model predicts both images belonging to the same person, and false otherwise.
- **feedback**: sends a feedback about a model prediction error. It receives as arguments two images, **base image** and **prediction image**, as well as **model prediction**, **human prediction** and **motive**, that is a brief explanation about the feedback.
- **calculate embedding**: calculates and return an embedding for a image. It receives as argument a single **image** and returns an object with the name of the model used to generate the embedding as well as the embedding itself.

With these methods, each client is able to setup a local database ready to be used in production integrated with our API.

3.5.2 Predict This is the method that will be mostly used in production. Every time that someone attempts to enter the residential building, the client's local system will take a picture of the person (**prediction image**), compare it with its own database to find the picture (**base image**) that is the closest to the person image and send both images to this API method. This method will then call the production model's predict method and return the prediction to client's local system.

3.5.3 Feedback Every time that a human operator, often a doorman or a security guard, verifies that a prediction is incorrect, he can send a prediction feedback usign the **feedback** method. This method assumes that model's prediction is incorrect, so either it predicts

that **base image** and **prediction image** belongs to the same person but they do not or it predicts that they do not belong to the same person but they do. In both cases, human operator can send the feedback by calling this method and passing as inputs:

- **base image**: the image that comes from client's database;
- **prediction image**: the real-time image taken by the client of the person trying to access the building;
- **model prediction**: the prediction returned by the model using both images above;
- **human prediction**: the human operator prediction;
- **motive**: a brief explanation about the feedback: "same person" if the human operator predicts that the images belongs to the same person or "not the same person" otherwise.

Every feedback sent by all clients will be stored in our database and will be considered in the future when model re-training or fine-tuning takes place.

3.5.4 Client's initial setup In order to use our API as a solution for face verification, each client must first perform an initial setup in its own database. Our system assumes that, when prediction takes place, our clients will call our API method **predict** and pass as arguments two images: **base image** that comes from clients own database, and **prediction image** that is the picture that is taken from each one that passes through the face verification system. So each client needs to properly setup its local database in order to properly select the **base image** appropriate to each case.

The setup must follow these steps:

- create a database with all registered people, that would be all residents and registered building staff, that would contain one registration picture for each person;
- take all registered images and call the **calculate embedding** method to get their embeddings;
- save all images embeddings in a vector store database;
- create a pipeline that takes real-time photos for each person that passes through the facial recognition system and pass it through the **calculate embedding** method to get its embedding and perform a vector search in client's vector store database to get the closest registration image;

With that setup in place, each client is able to call the **predict** method passing as input the closest registration image, that would be the **base image**, and the real-time taken image, that would be the **prediction image**.

The connection between client's system and FaceV's system can be shown in the Figure 2 below.

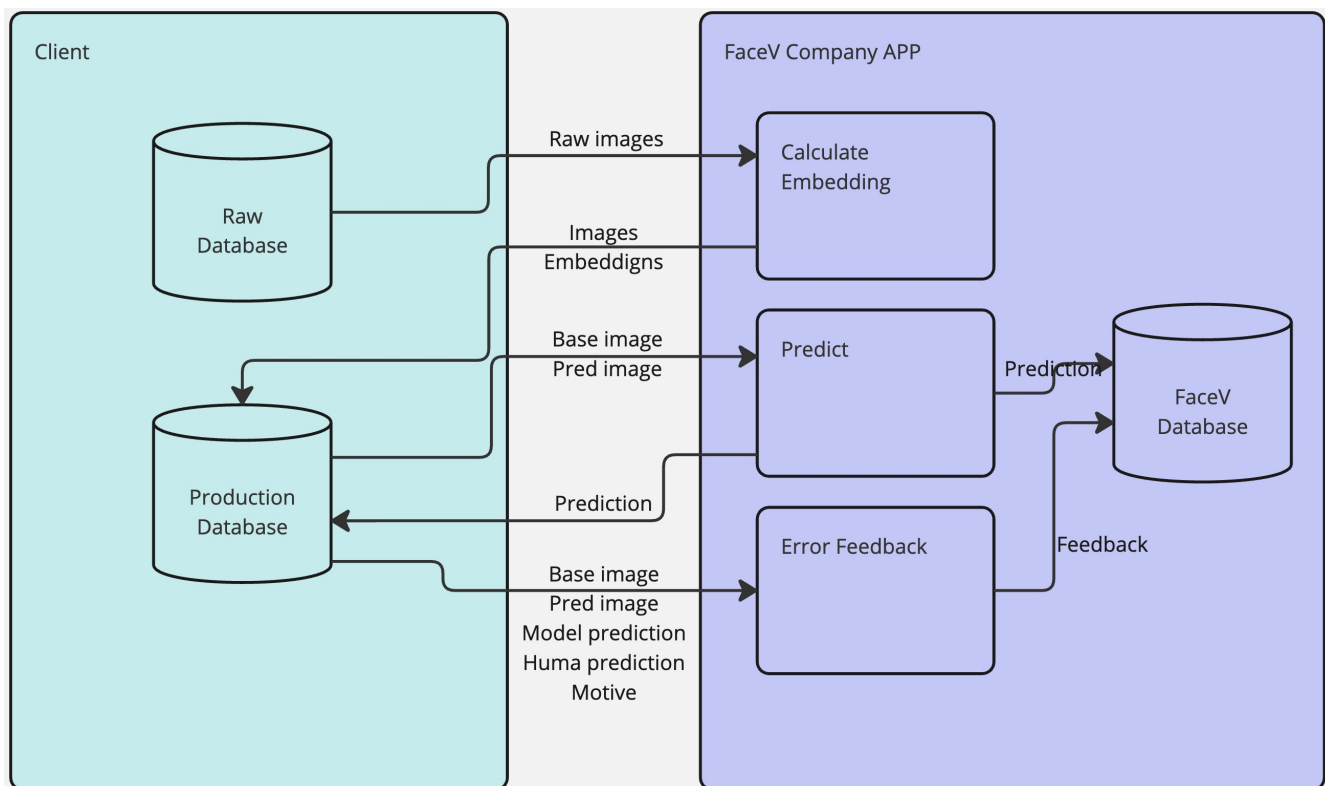


Figure 2. This is a wide figure that spans both columns.

4 Results

4.1 Experimental Planning

With the main database in place and the experimental planning already performed, we can check whether or not our approach of determining the best model given a dataset is properly working. We expect our system to automatically determine the best model after all rounds of resampling runs.

Using the Labeled Faces in the Wild dataset, choosing only people with at least 3 images to be in the **registration** database and people with at least 2 images to be the visitors in **operation** database, we can run the resampling strategy for a given number of times. In our experiment, we chose 50 rounds of random sampling, choosing 100 people to be our registered people and 50 more to be the visitors. In this case, the **registration** database will have 100 pictures and the **operation**, 300.

With the 50 errors gathered in the 50 round resampling strategy, we calculated the mean error and its standard deviation, selected the two best models FaceNet512 and GhostFaceNet and performed the dependent T-test for paired samples. The T-test were conclusive that we can reject the hypothesis that both models have the same error distribution and, therefore, select the best possible model.

In the figure 3 below we can see the error distribution for the champion model FaceNet512 and the challenger model GhostFaceNet.

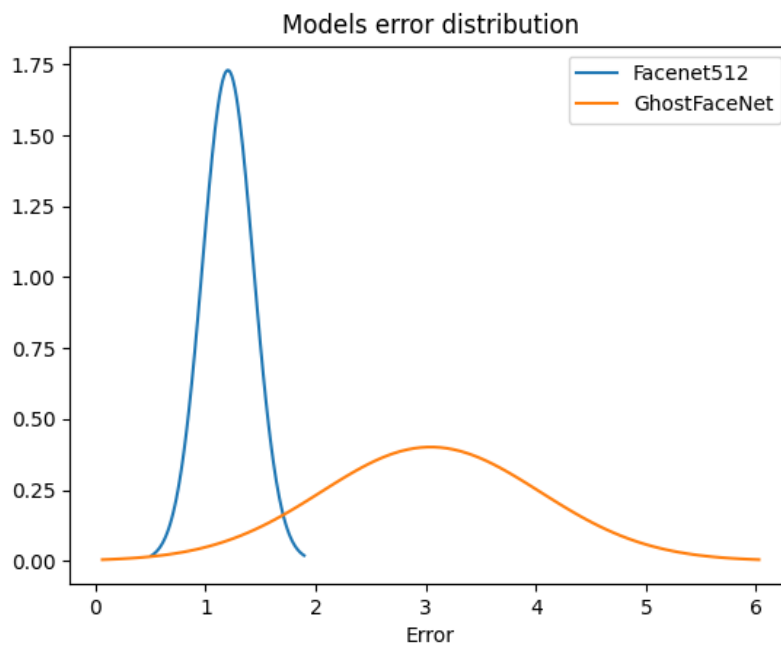


Figure 3. Best models errors distributions

After defining the best model, our experiment automatically saves its configuration into the model index in Elasticsearch, so that we can easily access the current production model while serving our production system via API.

4.2 API serving

Our API, with its methods, were properly built and runs seamlessly integrated with our database, and is properly documented, as shown in the figures 4 through 7 in the Appendix A section.

With all these methods provided via our API, each client can easily prepare its own database and use our Face Verification system to grant or deny access in its residential building.

5 Discussion

Our Face Verification expert system is able to properly determine which model best fits a client's use case, given its image database. With little initial setup, it can automatically create all the necessary Elasticsearch indexes, perform the experimental planning, select the two best models according to clients security needs and verify if both models are statistically different so that it can define the one best model. With the model properly selected, it automatically saves its configuration into an index in Elasticsearch and then the client is already able to access our API to implement a face recognition system to access control its residential buildings.

Although this system works perfectly with the LFW database, some minor changes might be needed if clients database does not have enough images so that the experiment can run. In some examples that we ran, when the selected N number of residents is low, the

model error standard deviation becomes significantly greater and the models performance can no longer be distinguish with a T-test. In these cases, when client's database is not large enough, we need to gather some data and enrich the database so that the experiment can properly run successfully. We expect at least a few hundred people with more than 3 pictures so that the experiment succeed in finding the best model. However this is a currently known problem, we will properly deal with it in the future.

5.1 Future work

Some topics and techniques were set apart this project scope and will be tackled in the future. It is in FaceV's plans to properly develop solutions that can handle the topics bellow:

- Remove **base image** and **prediction image** from feedback API method, as these images are already saved in the prediction database;
- Develop an automated distance threshold optimizer that is able to choose the best possible threshold for each client, taking into consideration not only the image population distribution but also the desirable degree of security;
- Define the best strategy to enrich clients database in the case of it not being large enough to run the experiment.

6 Conclusion

We have developed a Facial Verification Expert System capable of automatically selecting the best model among all the available ones, given an images database, and following clients security requirements. This system serves as a proof of concept that the facial verification task can be solved in a way that the client is capable to input a certain level of degree of security expected, chosing the weight of an incorrect prediction that best aligns with its concerns. It can be implemented to solve the model selection phase in similar use cases, like comercial buildings for access control, or even universities for attendance monitoring. This system also serves as a base for developing similar expert systems that performs face verification task in other scenarios.

Acknowledgements

This system was developed during the PO-235 Data Science Project class, instructed by Professor Filipe Verri, PhD, at Aeronautics Institute of Technology, São Paulo, Brazil. I am grateful for all the help, guidance and precious insights that Professor Verri gave throughout this project.

References

Huang, Gary B. et al. (2012). "Learning to Align from Scratch". In: *NIPS*.

Schroff, Florian, Dmitry Kalenichenko, and James Philbin (June 2015). "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. doi: 10.1109/cvpr.2015.7298682. URL: <http://dx.doi.org/10.1109/CVPR.2015.7298682>.

A API Images

This section contains all the API methods images.

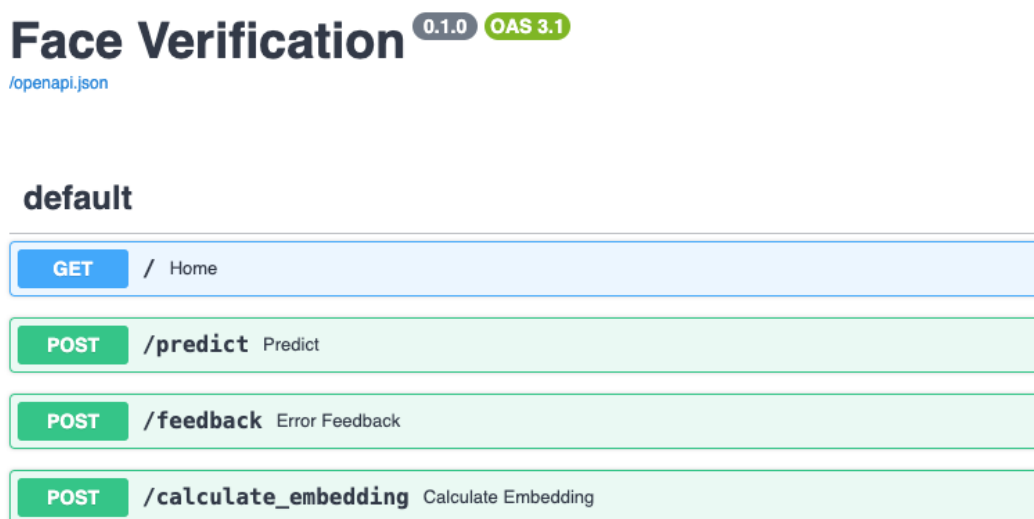


Figure 4. API methods list

Face Verification

0.1.0

OAS 3.1

/openapi.json

default

GET

/ Home

POST

/predict Predict

Predicts whether two uploaded images are of the same person or not.

Args:

base_image (UploadFile): The base image to compare.

pred_image (UploadFile): The prediction image to compare.

Returns:

bool: True if the images are of the same person, False otherwise.

Figure 5. API predict method

Face Verification

0.1.0

OAS 3.1

/openapi.json

default

GET

/

Home

POST

/predict

Predict

POST

/feedback

Error Feedback

Gathers feedback from a user about the accuracy of the model.

Args:

base_image (UploadFile): The base image to compare.

pred_image (UploadFile): The prediction image to compare.

model_prediction (bool): The prediction of the model.

human_prediction (bool): The prediction made by a human operator.

motive (str, optional): The reason for the feedback. Defaults to 'Not the same person'. Must be one of the following:

- 'Not the same person'
- 'Same person'

Returns:

dict: A dictionary with a single key 'message' and a value of 'Your feedback was successfully saved in our database'.

Raises:

HTTPException:

- If the images are the same, or
- if model prediction and human prediction are the same,
- or if model prediction is True but the motive is not 'Not the same person'.
- or if model prediction is False but the motive is not 'Same person'.

Figure 6. API feedback method

Face Verification

0.1.0OAS 3.1

/openapi.json

default

GET

/ Home

POST

/predict Predict

POST

/feedback Error Feedback

POST

/calculate_embedding Calculate Embedding

Calculate the embedding of an image.

Args:

image (UploadFile): The image file to calculate the embedding for.

Returns:

Embedding: The calculated embedding of the image.

Parameters

Figure 7. API calculate embedding method