Optimization of ResNet50 based on flatten and dense layer insertion applied to facial recognition voting system

Groupe 8: Project II (INF4258)

Group members :

FONGANG MBE D. Brel	20V2004
EMAHA TCHEDJAM Clémence Audrey	20Y028
SASHA Junior Tsamo	20U2746

I. Project presentation

1. Introduction

In this paper, we propose a face recognition-based voter authentication system that captures an input facial image of each voter using deep learning approach, RESNET50 verifies it with the characteristics of the facial images that have been recorded, the result being positive means that he can vote. After voting, the system confirms the vote by capturing the voter's facial image again and delivers a short text message. A summary of the votes can be consulted and thus guarantee that a person has voted only once.

2. Context

Voting is an act by which a citizen participates in the choice of his/her representatives or in decision-making during a ballot (presidential, parliamentary, or municipal elections). During the voting phase, voters go to their polling stations to exercise their voting rights (cast their votes). Each station has a paper identification system to identify the voters in detail. This system has proven its worth, but at the end of each election cycle, the opposition loudly criticizes the results of these elections (during the electoral and post-electoral phases), citing electoral fraud (filled ballot boxes, falsification of results, multiple voting by one voter).

3. Problem to fix

Here, we are interested in multiple voting by one voter. This could occur during the voters' identification simply because the system in place does not control the fact that a voter can't vote several times, at several polling stations.

4. Related works and their limitations

Manual feature extraction from facial images using traditional methods such as LBP and HOG and then use these features in the classification step. Recently, the artificial neural network has been used to extract features from facial images, using large amount of dataset. Given its relevance, several researchers continue to explore this technique. Yuxiang et al. used neural networks in particular ResNet50, which is composed of 50 layers with the Global Average Pool.

The 13,323 normalized facial images at a size of 224 x 224 that they preprocessed from the LFW dataset of nearly 5748 identities, are then aligned to generate a 128-dimensional matrix of each image as output from the neural network. The preprocessing consists of detecting the face region in a complex background using AdaBoost. Afterward, these preprocessed images are fed into a Resnet50 network to extract face and gender features through convolution layers. Before the output of these final images, the Global Average Pool is used to reduce the size of the network features. After this batch training, classification accuracy for each module was found: 99.33% for the face recognition module and 93% for the gender recognition module. The problem we have with this approach is that the classification accuracy could be further improved before feeding the dataset into the network, the authors would crop these images. Omkar et al. used a first 8-layer AlexNet neural network to perform a manual filter of 2.6 million images and obtained 982,803 images. After training, the images obtained in the second 37-layer ConvNet neural network incorporating Dropout was 98.95% with the LFW dataset. Their proposed neural network uses dropout to avoid overlearning, but unfortunately, it lowers the performance of the model. The presence of false positives in the facial images may not cause an unregistered person in the system to be recognized. Mondal et al. used the GoogleNet neural network, which is composed of 22 layers integrating the triple loss function. They extracted a 200-dimensional matrix on 13323 images of 5749 identities from the LFW dataset introduced in this neural network. The model was tested on 100 images of 10 people obtaining an accuracy of 99.1% when classified with a support vector machine. They proposed a system to secure votes, however, in case of bad votes, such as identical voters, although measures of similarity of the characteristics between the two faces of these voters can be too high, one of the two unregistered persons ends up voting and his record is deleted in the database. When the second person arrives to vote, he/she will not be able to (false positive).

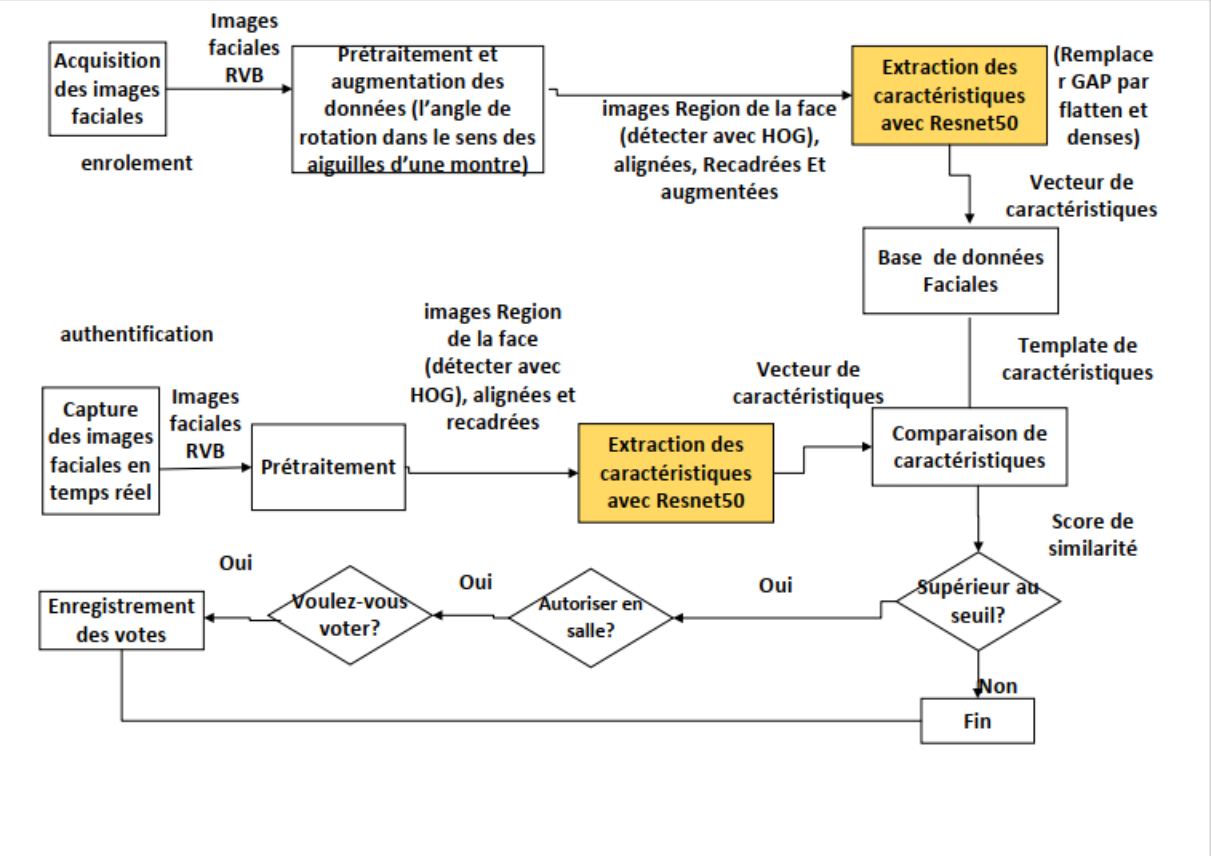
5. Proposed solution

In this study, we use CNN to minimize the false positives of facial recognition to optimize its efficiency in use for multiple vote resolution.

6. Nature of the solution

Il s'agira donc de produire un sytème qui permet d'identifier un individu par simple photograhie. La solution sera mise en œuvre sur plateforme web et mobile.

7. Methodology



Our proposed facial recognition-based voting system consists of two main successive phases: the enrolment phase and the authentication phase. When the system captures an input facial image of each voter using deep learning approach, it checks it with the features of the facial images that have been stored in the facial database. A feature similarity measure helps to compute the similarity comparisons so that the most dominant labels are attached as image labels. If the value of the similarity measured is less than a certain value, the classification result returns an error result, i.e. the system cannot identify the person. This is how facial recognition works, if two people are identical, their features are too similar. Currently, face recognition usually consists of four modules: facial image acquisition, data preprocessing and augmentation, feature extraction with Resnet50, and feature comparison. Furthermore, a facial database is used to store face feature vectors and to perform feature comparisons.

8. Machine learning algorithms for the facial recognition

Here follows the implementation of a Resnet50 CNN using Keras:

```
9. from keras import Model
10.from keras.layers import Input, Conv2D, BatchNormalization, Activation,
   Add, ZeroPadding2D, MaxPooling2D, AveragePooling2D, Dense, Flatten
11.from keras.initializers import glorot_uniform
12.

13.#Implementation of convolution block
14.def convolutional_block(X, f, filters, stage, block, s):
15.
```

```
    F1, F2, F3 = filters
    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1), padding='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)

    return X


#Implementation of Identity Block

def identity_block(X, f, filters, stage, block):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    F1, F2, F3 = filters

    X_shortcut = X

    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base + '2a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', name=conv_name_base + '2b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    # Skip Connection
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
```

```
55.
56.     return X
57.
58.
59.#Implementation of ResNet-50
60.def ResNet50(input_shape=(224, 224, 3)):
61.
62.     X_input = Input(input_shape)
63.
64.     X = ZeroPadding2D((3, 3))(X_input)
65.
66.     X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
   kernel_initializer=glorot_uniform(seed=0))(X)
67.     X = BatchNormalization(axis=3, name='bn_conv1')(X)
68.     X = Activation('relu')(X)
69.     X = MaxPooling2D((3, 3), strides=(2, 2))(X)
70.
71.     X = convolutional_block(X, f=3, filters=[16, 16, 64], stage=2,
   block='a', s=1)
72.     X = identity_block(X, 3, [16, 16, 64], stage=2, block='b')
73.     X = identity_block(X, 3, [16, 16, 64], stage=2, block='c')
74.
75.     X = convolutional_block(X, f=3, filters=[32, 32, 128], stage=3,
   block='a', s=2)
76.     X = identity_block(X, 3, [32, 32, 128], stage=3, block='b')
77.     X = identity_block(X, 3, [32, 32, 128], stage=3, block='c')
78.     X = identity_block(X, 3, [32, 32, 128], stage=3, block='d')
79.
80.     X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=4,
   block='a', s=2)
81.     X = identity_block(X, 3, [64, 64, 256], stage=4, block='b')
82.     X = identity_block(X, 3, [64, 64, 256], stage=4, block='c')
83.     X = identity_block(X, 3, [64, 64, 256], stage=4, block='d')
84.     X = identity_block(X, 3, [64, 64, 256], stage=4, block='e')
85.     X = identity_block(X, 3, [64, 64, 256], stage=4, block='f')
86.
87.     X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=5,
   block='a', s=2)
88.     X = identity_block(X, 3, [128, 128, 512], stage=5, block='b')
89.     X = identity_block(X, 3, [128, 128, 512], stage=5, block='c')
90.
91.     X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)
92.
93.     model = Model(inputs=X_input, outputs=X, name='ResNet50')
94.
95.     return model
96.def create_model() -> Model:
```

```
97.
98.     # base_model = ResNet50Class(ResNet50(input_shape=(224, 224, 3))())
99.     # x = base_model.nested_model.output
100.          base_model = ResNet50(input_shape=(224, 224, 3))
101.          x = base_model.output
102.          x = Flatten()(x)
103.          x = Dense(512, activation='relu',
    name='fc1',kernel_initializer=glorot_uniform(seed=0))(x)
104.          x = Dense(256, activation='relu',
    name='fc2',kernel_initializer=glorot_uniform(seed=0))(x)
105.          x = Dense(3, activation='softmax',
    name='fc3',kernel_initializer=glorot_uniform(seed=0))(x)
106.
107.          model = Model(inputs=base_model.input, outputs=x)
108.
109.          for layer in base_model.layers:
110.              layer.trainable = False
111.
112.          return model
113.
```

II. Model training

1. Facial images acquisition

Images were acquired using our promotion's Google Photos' album. The album consists of captures of different scenes and persons. All that with different shades of light and face orientations. And of course, all these images were collected and used with the consent of the students in the promotion.

2. Facial region detection (HOG)

The face regions were detected, extracted and saved using a **HOG (Histogram of oriented gradients)**, accessible in the OpenCV library through the XML files **haarcascade_frontalface_default.xml** loaded with OpenCV's **CascadeClassifier** function.

3. Preprocessing and data augmentation

The images went through an augmentation (rotation and scaling) that produced a larger and more exploitable dataset.

4. Training the Resnet50

The Resnet50 was trained on 65 epochs on the final dataset consisting of 2763 images of 33 persons and took around 2h43.

We used checkpoints to save the best metrics model.

```
Epoch 60: val_accuracy did not improve from 0.75968
65/65 [==============================] - 141s 2s/step - loss: 8.3876e-04 - accuracy: 1.0000 - val_loss: 1.3929 - val_accuracy: 0.7468

End of training.
The training lasted: 8727.345619499916 s.

Saving...

Model saved.
```

III. Check list of the voting process

1. Environment setup

The app consists of two parts:

- **A back-end** that consists of a processing server running the model.

In order to run it, the following should be installed:

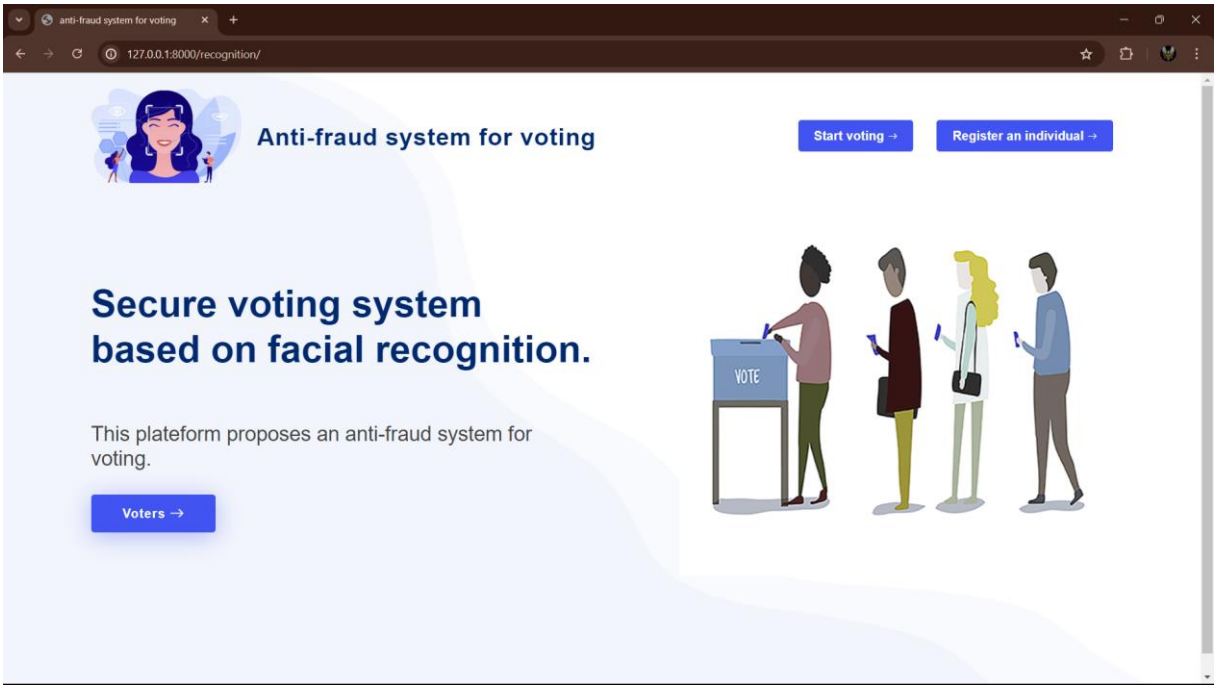
  - Python 3.10 (due building wheel compatibility with Tensorflow 2.10)
  - Conda/miniconda environment
  - Tensorflow version 2.10.0
  - Pillow
  - Opencv-python
  - Flask
  - Flask_json
  - Numpy
- **A web application** in a MVC architecture. This could be set up in a different environment but requires:
  - Python 3.12
  - Pillow
  - Django

All these ca be installed using the **requirements.txt** file inside the project root directory.

2. Launching the components

To launch:

- **The back**: python server3.0.py # using the 3.0 python version of course
- **The front**: python Recon/manage.py runserver (web app can be accessed via http:127.0.0.1:8000)



A. Enrolment

1. Facial images acquisition

The voter is captured using the device webcam and the capture is sent to the web application Django server.

2. Facial region detection (HOG)

The web server then sends the face regions were detected, extracted and saved using a **HOG (Histogram of oriented gradients)**, accessible in the OpenCV library through the XML files **haarcascade_frontalface_default.xml** loaded with OpenCV's **CascadeClassifier** function.

3. Preprocessing and data augmentation

The images went through an augmentation (rotation and scaling) that produced a larger and more exploitable dataset.

4. Characteristics extraction using the pretrained Resnet50

Then, we used the feature engineering part of the Resnet50 trained ealier to extract the characteristics of the extracted face regions.

The characteristic vector was then saved with the corresponding labels

B. Authentication

1. Voting room access

The voter stands in front of the device camera, in a correctly lighted place. Then the app automatically takes a capture of him/her and sends it to the back-end. There, the capture is redirected to the processing server on which it went to all the preprocessing and face region extraction processes done during enrolment before being fed into the pretrained model for feature extraction. Those features are sent back the web application back for the next steps.

The feature vector is then compared to all the registered individuals' using the Euclidian distance: only the closest is kept.

A threshold is also set in place so that someone not ell recognized or someone resembling to the outputted voter can't get inside the voting room.

A message is then back to the client, allowing the voter or not to enter the room while the system unlocks the door accordingly.

The voter's state is updated to "Already voted" and so, next time he/she stands in front of the camera, going through all the previous process, the systems may tell them that they have already voted.

Conclusion

The voting system in place in Cameroon suffers many allegations, putting its credibility at risk.

References

Reference article

https://docs.opencv.org/4.x

https://www.tensorflow.org