Face Tracking and FaceRecogniton with Javascript face-api.js

Progetto di Laurea Rodrigo Cossi 917138

1. System Requirements

- Install Git.
- Clone my project from Github. (**https://github.com/RodrigoCossi/progetto-IDS**)
- Install Visual Studio Code for code editing.
- Install 'Live Server' VSCode extension
  https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer

  This project is a web application and thus runs in a browser. After installing 'Live Server' click on the 'Go Live' button that appears at the bottom-right corner of the editor. The app should start running on http://127.0.0.1:5500 in your default browser (I personally recommend the latest versions of Firefox or Chrome).

2. The Project

Intro:
Face-api.js is a JavaScript library for face detection and face recognition in the browser implemented on top of the tensorflow.js core API. The library implements several

CNNs(convolutional neural networks) as a solution to face detection, face recognition ,face landmarks detection optimized for web and mobile.

Goals:

Using face-api.js**,** my goal was to develop a web application - using HTML, Javascript and CSS - capable of:

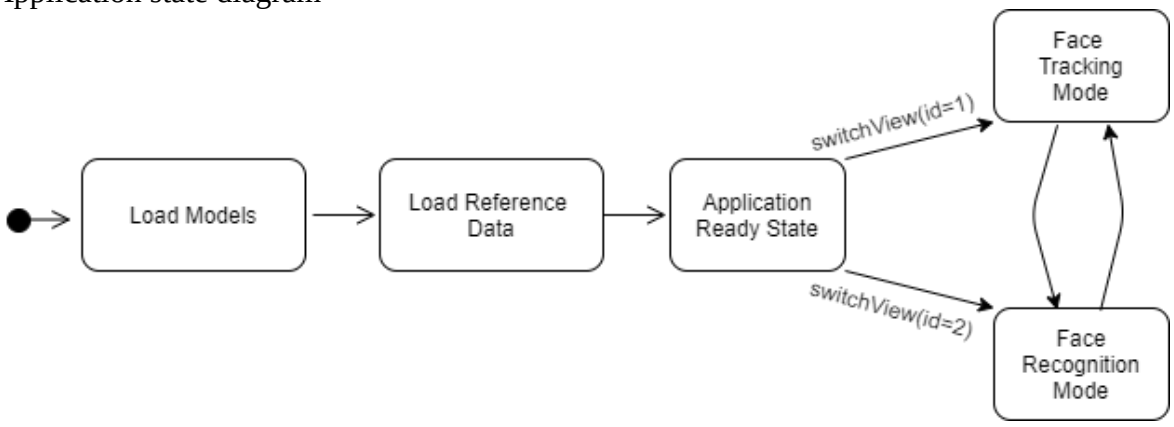
1. recognizing human faces on videos / photos
2. identifying emotions derived from facial expressions
3. estimating age/gender of detected human faces
4. learning to recognize people from a photo database (with probability ≥ 0.6)

The app will cover functionalities 1-2-3 on a webcam based, Face Tracking feature.
The app will cover functionalities 1-4 on a picture upload based, Face Recognition feature.

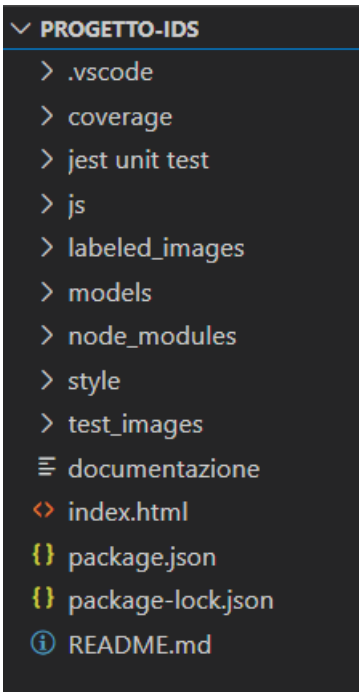
Project Structure:

This section explains in detail the application *folder structure* and the four main segments of the application: *loading the models*, *loading the reference data*, *face-tracking feature*, and *face-recognition feature*.

Application state diagram



Folder Description



.vscode: 'Live Server' configuration
coverage: 'Jest' folder
jest unit test: files containing unit tests
js: javascript files
face-api.min.js:
face detection and recognition javascript api
index.js:
main application javascript

labeled_images: reference images used for facial recognition
models: contains pre-trained models and weights for face detection
style: main application css file
test_images: images used to test the face recognition feature
documentazione: this doc
index.html: main html file
package.json: project dependencies list
package-lock.json: hierarchical view of the project dependencies

labeled_images, is a folder containing subfolders, aka **labels**, which in turn contain images (our reference data for facial recognition).
Each **label** subfolder, is named after one of the 'Friends' TV show main characters: [Monica, Chandler, Ross, Rachel, Phoebe, Joey] and each label contains 8 pictures of the corresponding actor.
test_images contains pictures of all the cast together, used to test the efficiency of the facial recognition algorithm.

Including face-api.js

The complete face-api package is available at the following GitHub repository and can be cloned with Git Bash terminal, typing the following command:

```
git clone https://github.com/justadudewhohacks/face-api.js.git
```

Note: My application includes only the pre-trained AI models (located inside 'weights' folder) and the minified version of the api; face-api.min.js (located inside 'dist' folder).

Loading the models

```
// Load models
Promise.all([
  faceapi.nets.ssdMobilenetv1.loadFromUri('/models'),
  faceapi.nets.faceLandmark68Net.loadFromUri('/models'),
  faceapi.nets.faceRecognitionNet.loadFromUri('/models'),
  faceapi.nets.tinyFaceDetector.loadFromUri('/models'),
  faceapi.nets.faceExpressionNet.loadFromUri('/models'),
  faceapi.nets.ageGenderNet.loadFromUri('/models')
]).then(start);
```

As soon as the index.js file loads, it will start fetching asynchronously and in parallel, all the deep neural net models used by the application. They are:

SSDMOBILENETV1

This Convolutional Neural Network model uses Single Shot Detector based on Mobilenet v1 network architecture. This is an accurate model for face detection but slower compared to other neural net architectures. It's not ideal for real time detection because of its longer inference time. For this reason, this model is being used on the picture face recognition feature.

This neural net will compute the locations of each face in an image and will return the bounding boxes together with it's probability for each face.

FaceLandmark68Net

This CNN is capable of identifying 68 key reference points on a facial structure. From these landmark positions, the face can be framed centrally by bounding boxes. The proper alignment is crucial for the face recognition accuracy.

FaceRecognitionNet

This CNN is based on a ResNet-34 like architecture. it has been trained to map the characteristics of a human face to a unique **face descriptor** (a feature vector with 128 values). The model is **not** limited to the set of faces used for training, meaning you can use it for face recognition of any person. The face recognition algorithm used in the application, compares the face descriptor of an input image with the face descriptors of the reference data and looks for the shortest Euclidean distance between them.

TinyFaceDetector
This CNN is quicker but less accurate than SSDMOBILENETV1. This model works better in real time face detection situations. For this reason, this model is being used on the webcam Face Tracking feature.

FaceExpressionNet
CNN responsible for detecting facial expressions based on the 68 point landmarks.

AgeGenderNet
CNN responsible for predicting age and gender. It's a multitask network, which employs a feature extraction layer, an age regression layer and a gender classifier. The feature extractor employs a tinier but very similar architecture to Xception.

Loading Reference Data

The application facial recognition feature uses reference data from a photo database (represented by the 'labeled_images' folder). Right after loading the models, the application will call the async function **loadLabeledImages()**, which internally fetches all single face images from 'labeled_images' (*with faceapi.fetchImage(uri)*) and uses faceLandmark68Net and faceRecognitionNet to compute and map an unique descriptor to each image for each label *(using faceapi.detectSingleFace(mediaSource) .withFaceLandmarks.withFaceDescriptor()).*
The function **loadLabeledImages()**, will return and array of objects of type **LabeledFaceDescriptors**, which associates each label with the corresponding descriptors. This array of LabeledFaceDescriptors is then passed as a parameter to a **FaceMatcher** object, which will basically look for the best descriptor match and check if it respects a predefined similarity threshold. (more about it on the next section: Face Recognition)

```
async function start() {
  const labeledFaceDescriptors = await loadLabeledImages();
  faceMatcher = new faceapi.FaceMatcher(labeledFaceDescriptors, maxDescriptorDistance);
}
```

To help with the visualization, ***const labeledFaceDescriptors*** *structure is:*
[{ label1, descriptors[] }, { label2, descriptors[] }, ...]

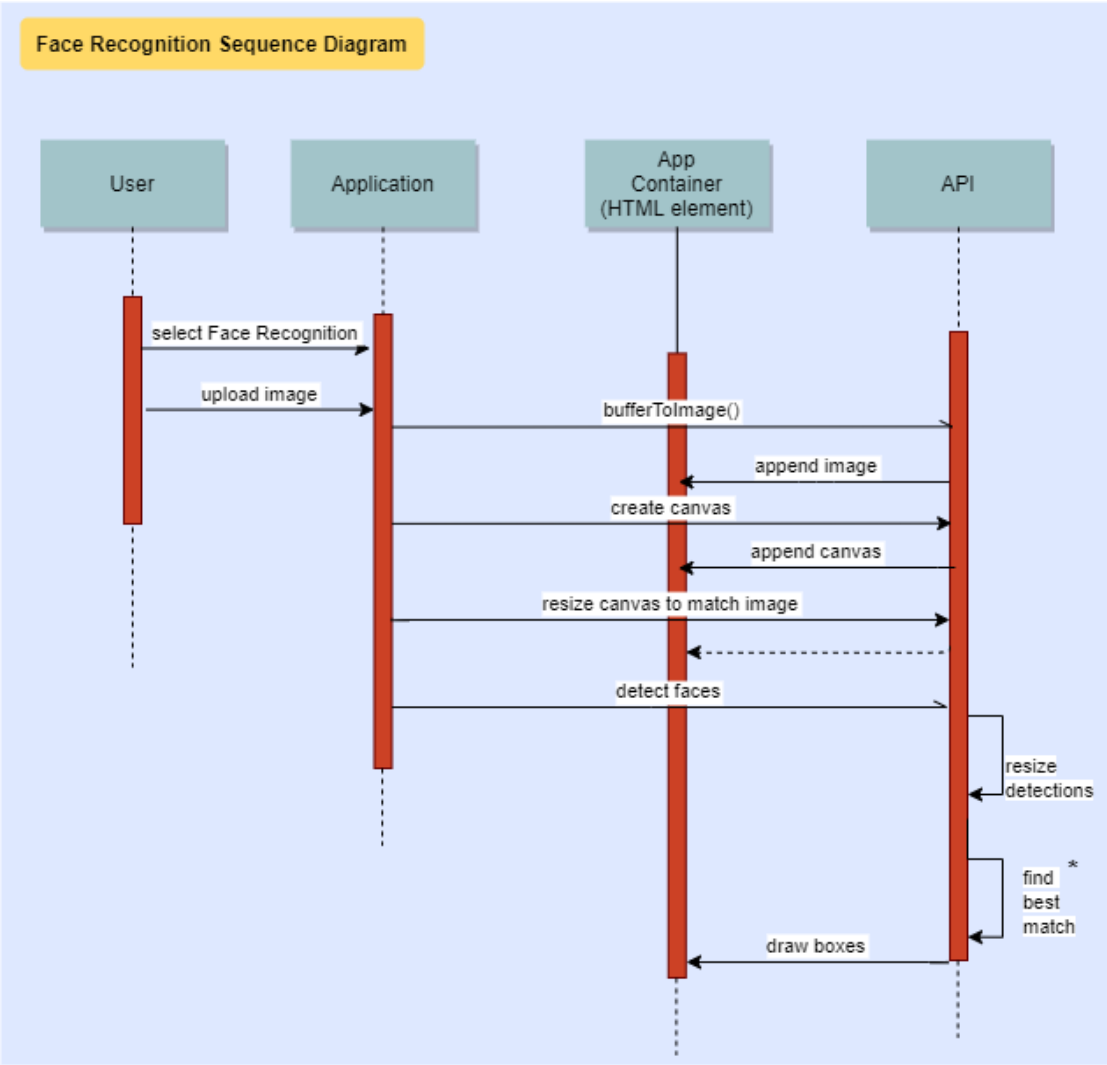
Face Recognition feature (from picture)

The facial recognition is accomplished by comparing the face descriptors of an uploaded test picture with the face descriptors of the reference data (our *const labeledFaceDescriptors*). More precisely, the prediction is done by computing the **Euclidean distance** of a input face's **descriptor vector** to each descriptor of a label and a **mean value** of all distances is computed. Furthermore, the mean value having the shortest distance from the input face's descriptor vector is returned as our **best match**. This value is later confronted to a threshold value to determine if our 'best match' is valid or not.
The threshold value represents the max distance threshold of two descriptors. The higher the distance the more unsimilar two faces are. If the distance of two descriptors is lower than the threshold value, we have a match. If none of the descriptors match, the face will be called 'unknown'. (0.6 is the arbitrary threshold value being used in the application.)

Considering the widely used formula for representing distance based similarity: 1 / (1 + d) where d is the Euclidean distance between two descriptors, a threshold value of 0.6 would produce results with similarity equal or grater than 62.5%.

<u>Our Test Case:</u>
Given an input image, in our case an image of 'Friends' cast, we want to be able to identify all of it's members by drawing boxes with their names around their faces. In order to achieve this, we will provide individual images of the actors and label their images with their respective character names as our reference data (as explained earlier on 'folder description'). Then, we compare the input image to our reference data, by calling **faceMatcher.findBestMatch()** method. The results will be displayed on a canvas (html element) over the input image.

The above-stated is depicted in the sequential diagram on the next page.

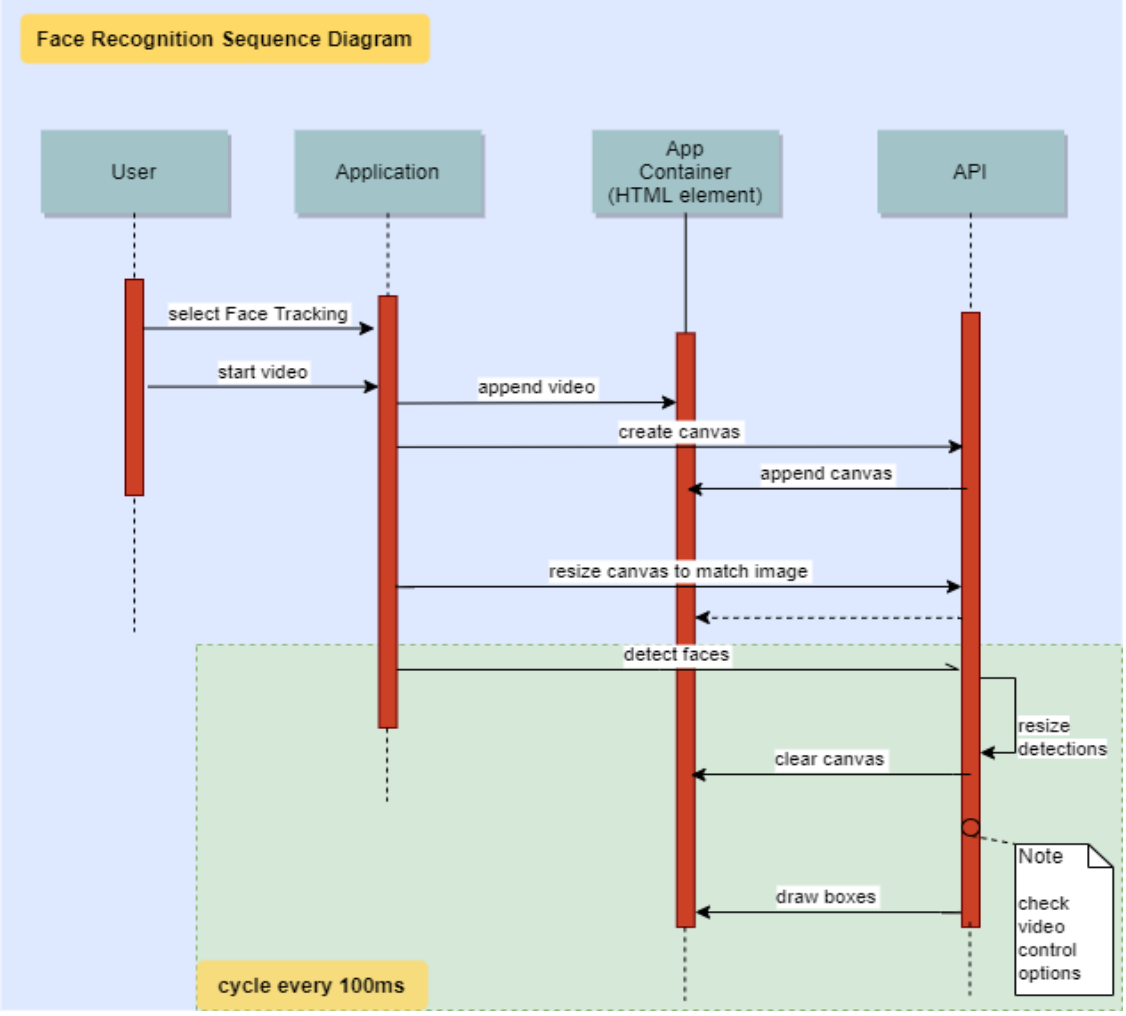


Face Tracking feature (from webcam)

The webcam face tracking feature can detect faces on camera, draw face landmarks, estimate age and gender of a detected face, and detect facial expressions.
We start the video data stream by attaching our webcam stream to the video element using **navigator.mediaDevices.getUserMedia(input).**
The video 'play' event will get triggered and create an html canvas that will overlay the video element and faceapi will take care of matching it's dimensions. The canvas will display the **bounding boxes** of each face, with their corresponding **scores**: the probability of each bounding box to be detecting a face.

As a live streaming event, ‘play’ will call **setInterval(func, delay)** every 100ms to repaint the face tracking results on canvas. The callback function will be using Face-api.js function ***faceapi.detectAllFaces(input, options) .withFaceLandmarks().withFaceDescriptors() .withFaceExpressions().withAgeGender()*** with *video* as *input* and *tinyFaceDetectorOptions()* as *options* to tell the API to use tinyFaceDetectorNet for face detection instead of the default net (ssdMobileNetV1).
After refreshing the canvas each cycle, the API will draw on screen: detections, landmarks, age/gender, and facial expressions; based on what options are selected on video control panel (html element).

The above steps are illustrated on the following sequence diagram.



3. Test Plan

The tests are divided in **component tests** and **performance tests**.

Component tests:

Encompasses unit tests of some of classes, methods and procedures used in the application.
They are:

- faceapi inbuilt methods and Classes
- load models
- loadLabeledImages()
- video controls - startVideo()/stopVideo()
- helper functions – loadingComplete()/clean()

The tests involving faceapi pre-built methods and Classes are documented on the **faceapi.*js* *github page***, inside the 'test' folder. The remaining unit tests were produced using **JEST** and are located inside the project's folder '*jest unit tests'.* To run the test coverage, simply run ***npm test*** from the terminal.

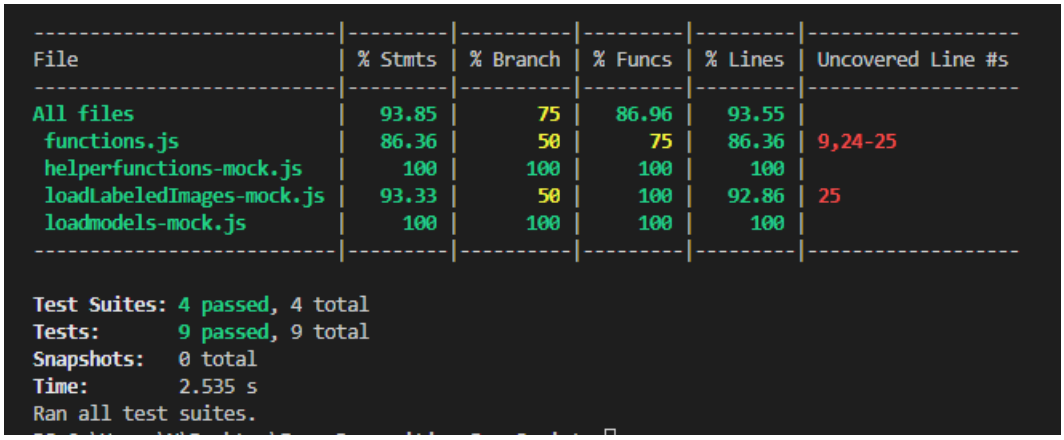
Performance tests:

They are:

- face recognition model accuracy
- Age and Gender, prediction test

4. Test progress Report

Unit Test Coverage:



```
---------------------------|---------|----------|---------|---------|-------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-------------------
All files                  |   93.85 |       75 |   86.96 |   93.55 |
 functions.js              |   86.36 |       50 |      75 |   86.36 | 9,24-25
 helperfunctions-mock.js   |     100 |      100 |     100 |     100 |
 loadLabeledImages-mock.js |   93.33 |       50 |     100 |   92.86 | 25
 loadmodels-mock.js        |     100 |      100 |     100 |     100 |
---------------------------|---------|----------|---------|---------|-------------------

Test Suites: 4 passed, 4 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        2.535 s
Ran all test suites.
```

Face recognition model accuracy:

According to the official faceapi.js ***documentation***, ***faceRecognitionNet*** - the neural net used by the library - achieves a prediction accuracy of 99.38% on the LFW (Labeled Faces in the Wild) benchmark for face recognition.(http://vis-www.cs.umass.edu/lfw/)

Euclidean distance is currently the only available data mining technique implemented by faceapi.js. Other techniques, like kmeans clustering or SVM classifiers, could be developed but the euclidean distance proved to be fast and to produce good enough results.

The model was used to recognize 'Friends' main actors on tests in 4 sets: providing *one*, *two*, *four* and *eight* images of each actor. Measuring the **best match** (*shortest Euclidean distance*) found for each actor, over two different group pictures, the results wore:

Actors\Images	1	2	4	8		1	2	4	8
Monica	0.46	0.53	0.49	0.49		0.5	0.59	0.57	0.53
Phoebe	0.5	0.42	0.46	0.44		0.49	0.49	0.49	0.49
Rachel	0.62	0.62	0.59	0.61		0.38	0.41	0.44	0.45
Joey	0.42	0.45	0.45	0.44		0.52	0.5	0.46	0.48
Ross	0.42	0.38	0.36	0.39		0.47	0.46	0.45	0.47
Chandler	0.51	0.49	0.47	0.48		0.57	0.53	0.53	0.55

Conclusion: Overall the tests have shown good and stable results, independently of the number of pictures used. There was a general improvement on the reliability when more images were used, but not a significant one. Computing the $\frac{correct\ predictions}{total\ predictions}$ ratio, we come across an accuracy of $\frac{45}{48}$ = 93.75%

Age and Gender prediction tests:

The Age-Gender model, according to the documentation, has been trained and tested on the following databases with an 80/20 train/test split each: UTK, FGNET, Chalearn, Wiki, IMDB*, CACD*, MegaAge, MegaAge-Asian. The $\star$ indicates, that these databases have been algorithmically cleaned up, since the initial databases are very noisy.

Total Test Results

Total MAE (Mean Age Error): **4.54**

Total Gender Accuracy: **95%**

Test results for each database

The `-` indicates, that there are no gender labels available for these databases.

Database	UTK	FGNET	Chalearn	Wiki	IMDB*	CACD*	MegaAge	MegaAge-Asian
MAE	5.25	4.23	6.24	6.54	3.63	3.20	6.23	4.21
Gender Accuracy	0.93	-	0.94	0.95	-	0.97	-	-

Test results for different age category groups

Age Range	0 - 3	4 - 8	9 - 18	19 - 28	29 - 40	41 - 60	60 - 80	80+
MAE	1.52	3.06	4.82	4.99	5.43	4.94	6.17	9.91
Gender Accuracy	0.69	0.80	0.88	0.96	0.97	0.97	0.96	0.9