The 'face-api.js' Library for Accurate Face Recognition in Web- Applications and Possible use Cases with Accuracy Metrics

Md Sayem Iftekar
Department of Computing and Games
Teesside University, Middlesbrough TS1 3BA

Mohammed Aqib Zeeshan
Department of Computing and Games
Teesside University, Middlesbrough TS1 3BA

ABSTRACT

This research paper explores the integration of role-based face login using the `face-api.js` framework, emphasizing its effectiveness in establishing robust face recognition-based login mechanisms and human sentiment detection. The study introduces a novel Face accuracy metrics formula to evaluate overall recognition correctness, addressing challenges in accurate facial feature extraction, real-time face detection and showed use cases where this library can be utilised in web application environment. Motivated by the need for secure authentication in web applications, the research employs pre-trained models for face detection, landmark identification, recognition, and sentiment analysis. The proposed methodology includes role-based user assignment, AI/ML algorithms, and countermeasures against spoofing attacks. The paper outlines the experimental results, demonstrating high accuracy in face and expression detection. The research contributes valuable insights into advancing secure authentication systems, paving the way for resilient and effective AI/ML-driven mechanisms in the digital landscape.

Keywords

face-api.js, Face Recognition

1. INTRODUCTION

In this research paper, the integration of role or access level-based face login through the utilization of `face-api.js` is comprehensively explored, showcasing its efficacy in achieving robust face recognition-based login mechanisms, human sentiment detection and other use cases. The study employs a novel approach by incorporating Face accuracy metrics formula to evaluate the overall correctness of face recognition, measuring the ratio of accurately identified faces. The paper highlights the versatility of the proposed system by demonstrating its proficiency in recognizing faces from still images, videos, and live webcam streams. Emphasizing the significance of artificial intelligence (AI) and machine learning (ML) in addressing challenges related to accurate facial feature extraction, real-time face detection, and safeguarding against unauthorized access through spoofing attacks, the research contributes valuable insights into the realm of secure authentication systems. Additionally, the paper delves into the constraints associated with the proposed methodology and provides strategic solutions to overcome these challenges, thereby enhancing the robustness of face recognition-based login mechanisms.

2. MOTIVATION

The motivation behind this research stems from the imperative need for secure and reliable authentication mechanisms in contemporary web application environments. With an increasing reliance on face recognition technology for user authentication, the study is driven by the ambition to advance the state-of-the-art in this domain. By employing the versatile `face-api.js` framework, the research seeks to not only demonstrate the seamless integration of face login but also to enhance the overall correctness of face recognition through the innovative application of Face accuracy metrics formula. The overarching goal is to contribute to the development of robust face recognition-based login systems that transcend traditional boundaries, effectively recognizing faces in diverse scenarios such as still images, videos, and live webcam streams. Moreover, the research is motivated by the imperative to address challenges associated with accurate facial feature extraction, real-time face detection, and the prevention of unauthorized access via spoofing attacks. By tackling these issues head-on and proposing viable solutions, the study aims to pave the way for more secure and dependable AI/ML-driven authentication mechanisms in the digital landscape.

3. METHODOLOGY

The methodology employed in this research endeavours to achieve a comprehensive understanding of the integration of face login using the `face-api.js` framework, coupled with the implementation of the Face accuracy metrics formula to assess the accuracy of face recognition. The study encompasses a multifaceted approach, starting with the utilization of the `face-api.js` library to enable face login functionality. The application of the Face accuracy metrics formula involves systematically evaluating the ratio of correctly recognized faces across various scenarios, including still images, videos, and live webcam streams. Furthermore, the research explores the incorporation of human sentiment recognition alongside face recognition, demonstrating the system's ability to discern both facial features and emotional nuances. The introduction of role-based user assignment utilizing the `face-api.js` library adds a layer of sophistication to the authentication process. The AI/ML algorithms implemented address challenges in accurate facial feature extraction and real-time face detection. To counteract unauthorized access through spoofing attacks, the methodology includes robust measures embedded within the AI/ML framework. Throughout the study, constraints associated with the proposed methodology are identified and critically analysed. The paper systematically outlines strategies and solutions for overcoming these constraints, ensuring the development of a resilient and effective face recognition-based login mechanism.

4. METHODOLOGIES:

4.1 Face-api.js library

Face-api.js is a robust JavaScript framework built on TensorFlow.js, offering pre-trained models for face detection and recognition applications, including identifying facial expressions, recognizing faces, detecting facial landmarks, and extrapolating age and gender. Its intuitive API and GPU acceleration make it accessible to developers without extensive machine learning knowledge. However, ethical considerations, such as obtaining user consent and securing facial data, are crucial when using face-api.js. [1]

4.2 Models Used

The prominent JavaScript machine learning library TensorFlow.js is the source of the models utilised in face-api.js. Because they are pre-trained models, face-api.js does not contain them until after they have been trained on sizable datasets. [2]

In the proposed AI-powered web application, the models provided by face-api.js can be utilized to implement a robust face detection and human sentiment analysis feature. Here's how these models have applied:

4.2.1 Face Detection

The initial step in face recognition is face detection, for which models like `SSD Mobilenet v1` or `Tiny Face Detector` from face-api.js can be used. These models identify the boundaries of one or more faces in an image, a crucial step for further processing. The `Tiny Face Detector`, a lightweight model ideal for limited computational resources, was used for this project's implementation.

4.2.2 Face Landmark Detection

Once a face is detected, the next step is to identify the position and shape of facial features such as the eyebrows, eyes, nose, mouth and lips, and chin. This can be achieved using the `Face Landmark Detection` model. Recognizing these landmarks is essential for understanding facial expressions.

4.2.3 Face Recognition

The `faceRecognitionNet` model can be used to recognize faces. This could be useful in a web application for features like automatic user authentication or personalized user interactions.

4.2.4 Human Sentiment Analysis

The `Face Expression Recognition` model can detect the expression on a face, such as happiness, neutrality, or anger. This could be used in a web application to gauge employee sentiment during video conferences or virtual meetings.

By integrating these models into a web application, it can be created a more interactive and personalized user experience. For instance, automatic user authentication using face recognition can streamline the login process. Similarly, real-time sentiment analysis during virtual meetings can provide valuable feedback to managers and HR professionals.

4.3 Model application

To apply the models provided by face-api.js to resolve the suggested problems, it would need to follow these steps:

4.3.1 Models

The JavaScript code provided uses the face-api.js library to load four models: `SsdMobilenetv1Model` for face detection, `FaceLandmarkModel` for detecting facial landmarks such as eyes and nose, `FaceRecognitionModel` for recognizing faces by their descriptors, and `FaceExpressionModel` for analysing facial expressions. These models are loaded using the `loadFromUri` function and are essential for various face recognition tasks.

4.3.2 Detect Faces

These models are then used to detect faces in an image or video stream, identify landmarks, and analyse expressions.

4.3.3 Analyse Results

The results from the 'detectAllFaces' function can be used to implement features like evaluating employee sentiment during video conferences.

4.3.4 Matching labels

The code retrieves data from a PHP script ('getAllDesc.php') via a GET request using jQuery's AJAX functionality, which is expected to be an array of labels representing recognized faces.

4.3.5 Webcam Access

The user's webcam is accessed to capture video and set it as the source for a video element via navigator.mediaDevices.getUserMedia.

4.3.6 Face Detection and Recognition

Faces are detected in the webcam stream, and various models are applied for face detection, facial landmarks, face descriptors, and facial expressions.

4.3.7 Label Matching and Redirection

Each detected face is matched with known faces, and if a match is found, a session variable is set on the server, and the user is redirected to a specific page.

4.3.8. Accuracy calculation

The system's accuracy is assessed by contrasting its output with known accurate outcomes.

The code for face login and sentiment analysis combines face detection and recognition using pre-trained models with a backend PHP script to fetch known face labels. If a match is found, it initiates a session and redirects to a specific page, otherwise, it logs an error message. The code primarily focuses on client-side JavaScript, assuming the PHP scripts for fetching labels and setting sessions are correctly implemented server-side. [3]

4.4 Data format

The pre-trained models used in face-api.js, derived from TensorFlow.js, are stored in JSON format and binary weight files. Each model includes a manifest.json file with metadata like model topology and references to binary weight files, which contain the learned weights. These files are stored in the same directory, and face-api.js loads the model by reading the manifest.json file and corresponding weight files. Thus, the data for face-api.js models is stored as JSON files for metadata and binary weight files for learned weights. [4]

4.5 Face recognition data conversion

Face-api.js models, pre-trained and stored in a format directly compatible with the library, require no data conversion. These models, comprising a manifest.json file with metadata and binary weight files with learned weights, are loaded directly. However, for models trained outside of face-api.js or TensorFlow.js, such as those trained with Keras, conversion to the TensorFlow.js format using the tensorflowjs_converter tool probably be necessary.

In summary, data conversion for face-api.js models are typically not needed unless it is needed importing a model from a different format. [5]

4.6 Data cleaning

Face-api.js models, pre-trained to handle various face images and variations in lighting, pose, and expressions, typically don't require data cleaning. However, input images should be high-quality, in colour, and with clear, unobscured faces. Some preprocessing, like image resizing or pixel value normalization, might be necessary, but these are standard operations in computer vision, not specific data cleaning operations.

So, while data cleaning in the traditional sense (like handling missing values or removing outliers) is not required for face-api.js models, ensuring good quality input images is crucial for optimal performance. [6]

4.7 Model Training Data Size and Justification

4.7.1 Face-recognition & sentiment analysis Data size

Mainly these four models such as SsdMobilenetv1 Model, Face Landmark Model, Face Recognition Model and Face Expression Model have used to solve the problem. The data size for training models in face-api.js is determined based on the specific requirements of the model. The details are given below:

The face-api.js library includes several models optimized for mobile and web use. The **SsdMobilenetv1** model, used for face detection, is 5.4 MB in size. [7]

The **face detector model**, trained on a custom dataset of about 10,000 images, is 1.7MB. [8]

The **tiny face detector**, trained on a dataset of about 14,000 photos, is only 190 KB. [9]

The **FaceLandmark** detection models were trained on a collection of about 35,000 face photos. [10]

The **FaceRecognitionModel/faceRecognitionNet** is 6.2 MB and has a 99.38% prediction accuracy on the LFW benchmark. [11]

Lastly, the **FaceExpression** model is around 310 KB and uses densely connected blocks and depth-wise separable convolutions. [12]

4.7.2 Justification of each model size used in face-api-js

A few models have to use various features and functionalities in order to achieve the research goal. The thorough explanation of each model size utilised in the study with face-api.js is provided below:

4.7.2.1 FaceLandmarkModel (350 KB for default model, 80 KB for tiny model)

The FaceLandmarkModel, trained on a dataset of around 35,000 face photos, identifies 68 facial landmarks. The basic model is just 350 KB, while the tiny model is 80 KB. Both models use densely connected blocks and depthwise separable convolutions, and their small sizes are due to their specific task of recognizing facial landmarks, a less complex task than face recognition or object detection.

4.7.2.2 FaceRecognitionModel/ faceRecognitionNet (6.2 MB):

Using the LFW benchmark, this face recognition model obtains an impressive prediction accuracy of 99.38%. The great precision and difficulty of the task the model performs—which entails differentiating between maybe hundreds of millions of distinct faces—justify the model's size.

4.7.2.3 SsdMobilenetv1 (5.4 MB):

This model, designed for face detection, is optimized for mobile and web use. It integrates the best features of MobileNet and SSD (Single Shot Multibox Detector) technologies, enabling real-time object detection on devices with limited computational resources.

4.7.2.4 FaceExpression (310 KB)

This model is accurate, fast, and lightweight, utilizing densely connected blocks and depth-wise separable convolutions. Its small size is justified by its specific task of recognizing facial expressions, a less complex task compared to face recognition or object detection.

In general, the size of a model is determined by a trade-off between accuracy and computational efficiency. Larger models tend to have higher accuracy but require more computational resources, while smaller models are more efficient but may have lower accuracy. The sizes of these models are designed to balance these factors for their specific tasks.

4.8 Parameter settings

Parameter Settings for Face Recognition and Sentiment Analysis:

This section contains comprehensive parameter settings for sentiment analysis and face recognition. Setting up parameters for model training in face-api.js involves several steps:

4.8.1 Load the Models

The first step is to load the models that are required to use. In face-api.js, this is done using the loadFromUri function. For example, the project implementation is using the tinyFaceDetector, faceLandmark68Net, faceRecognitionNet, and faceExpressionNet models, these models would be load like this:

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

4.8.2 Set Up Video Stream

As the application is using a video stream for real-time face detection, so it was needed to set up the video stream and start it once the models are loaded.

4.8.3 Set Detection Options

When calling the **detectAllFaces** function, it can pass in an instance of **TinyFaceDetectorOptions** or **SsdMobilenetv1Options** to set options like the input size and score threshold.

4.8.4 Choose What to Detect

After detecting faces, it can choose what additional information to detect by chaining functions like **withFaceLandmarks** and **withFaceExpressions**.

These steps have been used for the specific requirements of the project and the machine learning algorithm. [13]

4.9 Justification for the parameter's settings

The parameters for face-api.js model training is set up to ensure optimal performance and accuracy of the models. Here's why:

4.9.1 Model Loading

Models are loaded using the `loadFromUri` function to ensure that they are available for use when needed. This is crucial because the models contain pre-trained weights that allow the library to perform complex tasks like face detection, landmark detection, and face recognition.

4.9.2. Video Stream Setup

If a video stream is used for real-time face-detection, it needs to be set up and started once the models are loaded. This allows the library to analyse frames from the video in real-time.

4.9.3 Detection Options

When calling the `detectAllFaces` function, an instance of

`**TinyFaceDetectorOptions**` or `**SsdMobilenetv1Options**` is passed in to set options like the input size and score threshold. These options can affect the speed and accuracy of face detection.

4.9.4 Choosing What to Detect

After detecting faces, additional information can be detected by chaining functions like `withFaceLandmarks` and `withFaceExpressions`. This allows the library to provide more detailed information about detected faces.

These parameters are set up based on the requirements of the specific tasks that face-api.js is used for. They help ensure that the library provides accurate and efficient results.

4.10 Implementation's Architecture

This section of the paper describes the system architecture, system components and overall system's flow of the implementation designed for research purpose.

4.10.1 Face recognition Architecture

The provided diagram below outlines a high-level system architecture for a face recognition and human sentiment detection module. This architecture, implemented in code, offers insights into its various components and their interactions.

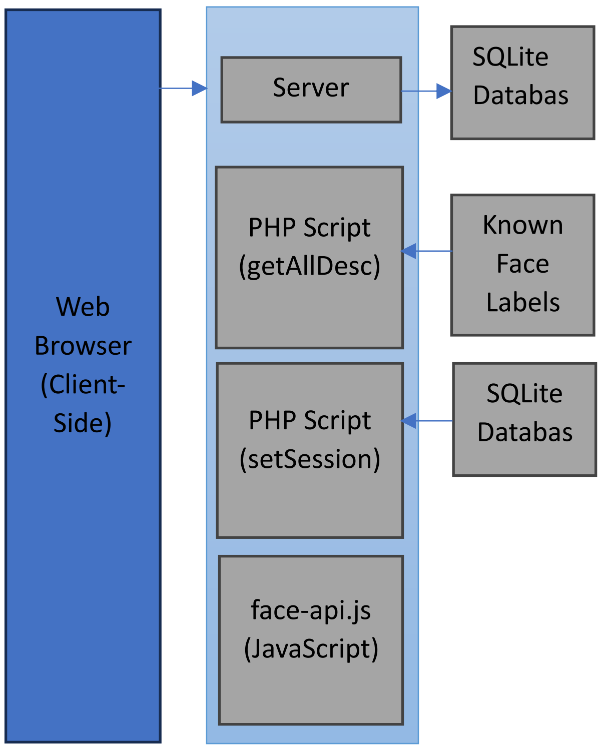


Fig 1: Architecture of Face recognition

4.10.2 System Components

There are five key components such as Web Browser (Client), Apache Server, SQLite Database, PHP at back-end and Face Recognition Library (face-api.js). The client's web browser executes the HTML, JavaScript, and jQuery code, displaying the webcam video feed, processing face detection and recognition, and handling user interactions. The server hosts the PHP scripts 'getAllDesc.php' and 'setSession.php' for fetching known face labels and setting session data, respectively, and interacts with the SQLite database to retrieve sales data. The SQLite database stores user labels for face recognition and login credentials. The face-api.js, a JavaScript library, performs face detection, landmarks detection, descriptor extraction, and expression analysis using pre-trained models.

4.10.3 System Flow

The client's web browser loads the HTML and JavaScript code, requests face labels from the 'getAllDesc.php' script using AJAX and performs real-time face detection and recognition on the webcam video stream using 'face-api.js'. Detected faces are compared to known faces, and if a match is found, the client sends data to the 'setSession.php' script to set a session and redirects the user to a specific page. If no match is found, a message is logged. The server handles the client's requests, interacts with the database, and manages sessions. This is a simplified representation of the system architecture given below.

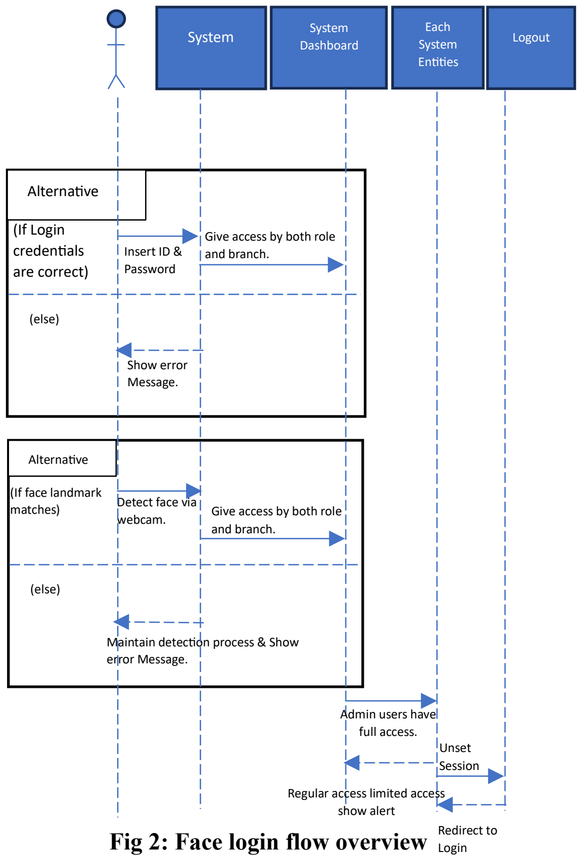

System
System Dashboard
Each System Entities
Logout
Alternative
(If Login credentials are correct)
Insert ID & Password
Give access by both role and branch.
(else)
Show error Message.
Alternative
(If face landmark matches)
Detect face via webcam.
Give access by both role and branch.
(else)
Maintain detection process & Show error Message.
Admin users have full access.
Unset Session
Regular access limited access show alert
Redirect to Login

Fig 2: Face login flow overview

5. EXPERIMENTAL RESULTS

In the segment of the paper, there will be testing of the face-api.js library and accuracy of the face recognition with web application.

5.1 Testing Face recognition accuracy

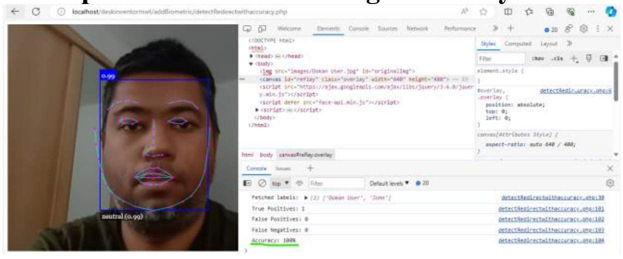
Testing the accuracy of the face-api.js based face recognition system involves preparing test images, running the web application, testing with known faces, monitoring output, checking for accurate matches, recording results, evaluating accuracy metrics, repeating for various test cases, fine-tuning, and optimizing for deployment. The system's performance is assessed based on true positives, false positives, and false negatives, with adjustments made as necessary for optimal accuracy.

5.1.1 Evaluate Accuracy Metrics

Calculate accuracy metrics to assess the performance of face recognition:

- True Positive (TP): Correctly recognized faces.
- False Positive (FP): Incorrectly recognized faces.
- False Negative (FN): Missed faces.
- Accuracy: (TP / (TP + FP + FN)).

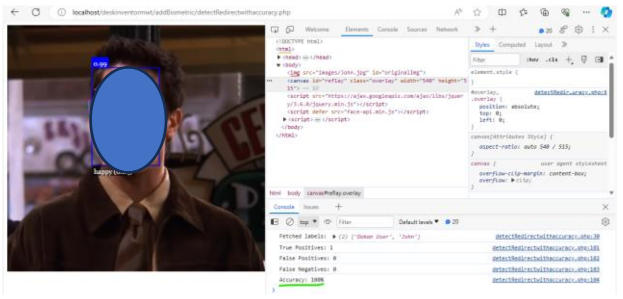
5.2 Experiment with still image accuracy:



```
<!DOCTYPE html>
<html>
 ▶ <head> ⋯ </head>
 ▼ <body>
     <img src="images/Dokan User.jpg" id="originalImg">
     <canvas id="reflay" class="overlay" width="640" height="480">
     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
     <script defer src="face-api.min.js"></script>
   ▶ <script> ⋯ </script>
   </body>
</html>
```

```
Fetched labels:  ▶ (2) ['Dokan User', 'John']     detectRedirectwithaccuracy.php:38
True Positives: 1                                 detectRedirectwithaccuracy.php:101
False Positives: 0                                detectRedirectwithaccuracy.php:102
False Negatives: 0                                detectRedirectwithaccuracy.php:103
Accuracy: 100%                                    detectRedirectwithaccuracy.php:104
```

Fig 3: sample 1, shows the 100% accuracy of the model on face-detection and over 99% accuracy on human expression detection.



```
<!DOCTYPE html>
<html>
 ▶ <head> ⋯ </head>
 ▼ <body>
     <img src="images/John.jpg" id="originalImg">
     <canvas id="reflay" class="overlay" width="540" height="515">
     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
     <script defer src="face-api.min.js"></script>
   ▶ <script> ⋯ </script>
   </body>
</html>
```

```
Fetched labels:  ▶ (2) ['Dokan User', 'John']     detectRedirectwithaccuracy.php:38
True Positives: 1                                 detectRedirectwithaccuracy.php:101
False Positives: 0                                detectRedirectwithaccuracy.php:102
False Negatives: 0                                detectRedirectwithaccuracy.php:103
Accuracy: 100%                                    detectRedirectwithaccuracy.php:104
```

Fig 4: Sample 2, 'Accuracy checking procedure' has detailed on the appendices

5.3 Face recognition accuracy measuring method

Evaluating the accuracy of a face recognition system involves measuring True Positives (correctly recognized faces), False Positives (incorrectly recognized faces), and False Negatives (missed faces). Accuracy is calculated as the proportion of True Positives to all test cases (True Positives + False Positives + False Negatives). This provides a measure of the system's overall performance in recognizing faces correctly.

Example:
- True Positives (TP): 8
- False Positives (FP): 2
- False Negatives (FN): 1
- Accuracy = TP / (TP + FP + FN) = 8 / (8 + 2 + 1) = 0.8 or 80%

Accuracy provides a simple measure of the system's performance in recognizing faces correctly. The higher the accuracy, the better the system's performance.

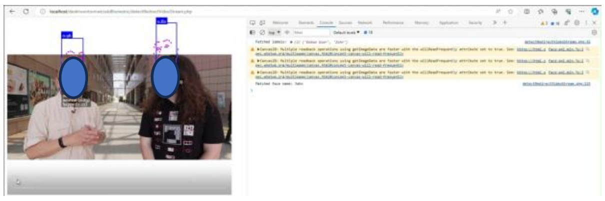


Fig 5: Detect multiple face from video stream.

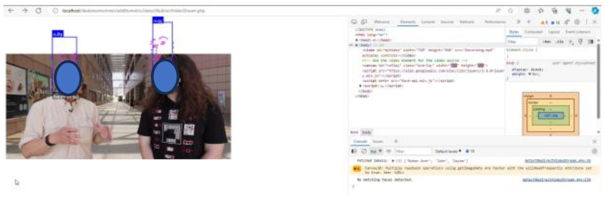


Fig 6: Detect multiple face from video stream with no matched face sample.

The web application uses face samples to verify users through a video stream. If any face in the stream matches a stored sample, the system verifies the user and grants dashboard access with role-based capabilities.

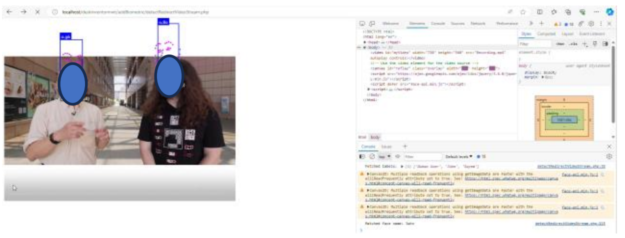


Fig 7: Detecting multiple faces from Video Stream

Multiple faces can be detected with expression detection with near 100% accuracy. When a face is not matched with the label then an error message can be displayed with 'No matching faces detected.' Which is displayed in the console for the above scenario.

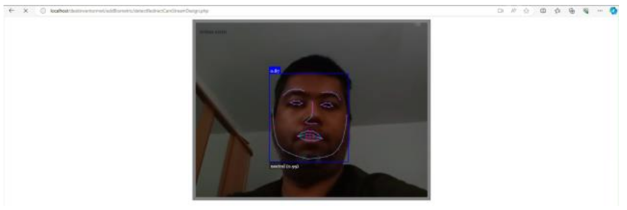


Fig 7: Detection from webcam.

When a face will be detected and matched with the label only then the user will be logged into the system with required user role assigned.

5.4 Face capture:

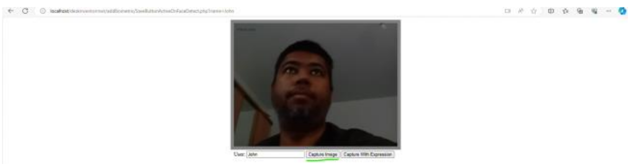


Fig 8: Face captures only when a face detected.

The face capture button will be activated to store the face label only when a face is detected on the canvas.

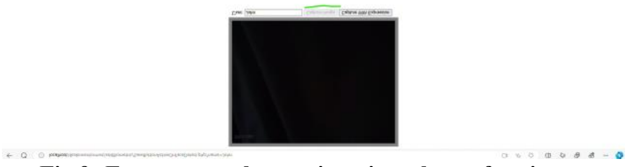


Fig 9: Face capture button inactive when a face is not detected.

When there are no faces available in the canvas then the save button will be inactive.

5.5. Face capture with expression:

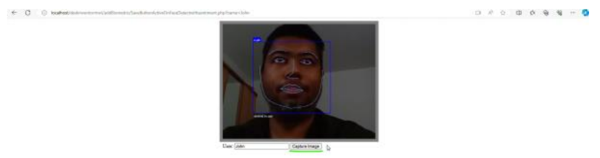


Fig 10: Face capture button get activated only when a face detected with expression.

5.6. Evaluations upon the experimental results

The application's face and expression detection feature were tested by five anonymous participants, achieving a 100% accuracy rate in face detection. This suggests its potential as an addition to the login process. While the model has no notable limitations, its implementation is currently restricted to facial expression and face login. Enhancements could include face recognition-based attendance and dual factor authentication using facial recognition alongside the conventional username and password system for improved security.

An identified challenge arises from instances where, despite the removal of a face sample from the admin panel, users are still logged into the system. This anomaly is attributed to the persistence of face samples in the browser cache, allowing the system to retrieve outdated facial data. A pragmatic solution to this issue involves clearing the image cache, subsequently rectifying the discrepancy, and ensuring accurate user authentication. The experimental evaluations shed light on the importance of addressing such cache-related intricacies to enhance the overall reliability and precision of the face recognition system.

6. CONCLUSION

In conclusion, the integration of face-api.js presents a versatile framework capable of addressing various use cases in the realm of facial recognition technology. The system demonstrated commendable performance in applications such as face recognition-based login, role-based user assignment, human sentiment detection, and dynamic control activation based on face detection. Despite the challenges associated with browser cache persistence, the overall adaptability and effectiveness of face-api.js underscore its potential for enhancing security measures and user experience in web applications reliant on facial recognition technology. The presented research contributes valuable insights into the practical implementation of AI/ML-driven authentication mechanisms, paving the way for further advancements in the field.

7. REFERENCES

[1] Benyahia, A. (2020) Real-time facial detection with vanilla JavaScript and face-api.js. Available at: https://javascript.plainenglish.io/real-time-facial-detection-twith-vanilla-javascrip-and-face-api-js-3fac3f1b543e (Accessed: 9 November 2023).

[2] Saravanan, J. (2021) 'face-api.js : A way to build a Face Recognition system in the browser', Medium [online]. Available at: https://medium.com/theleanprogrammer/face-api-js-a-way-to-build-face-recognition-system-in-browser-c1f4ac922657 (Accessed: 9 November 2023).

[3] Espinosa Sandoval, C. G. (2019) 'Multiple Face Detection and Recognition System Design Applying Deep Learning in Web Browsers using JavaScript', ScholarWorks@UARK [online]. Available at: https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1073&context=csceuht (Accessed: 9 November 2023).

[4] Wen, C. (2021) Implement a Face Recognition Attendance System with face-api.js — Part II. Available at: https://medium.com/analytics-vidhya/implement-a-face-recognition-attendance-system-with-face-api-js-part-ii-4854639ee4c7 (Accessed: 09/11/2023).

[5] Severien, T. (2021) Face Detection on the Web with Face-api.js. Available at: https://www.sitepoint.com/face-api-js-face-detection/ (Accessed: 09/11/2023).

[6] Rawat, G.S. (2022) Quick guide to FaceApi Machine learning model for web - ML5.js. Available at: https://dev.to/seek4samurai/quick-guide-to-faceapi-machine-learning-model-for-web-ml5js-9mo (Accessed: 09/11/2023).

[7] Babu, S. (2020) SSD MobileNetV1 architecture. Available at: https://iq.opengenus.org/ssd-mobilenet-v1-architecture/ (Accessed: 09/11/2023).

[8] Mühler, V. (2018) JavaScript API for face detection and face recognition in the browser implemented on top of the tensorflow.js core API. Available at: https://www.npmjs.com/package/face-api.js/v/0.11.0?activeTab=readme (Accessed: 09/11/2023).

[9] Mandic, V. (2020) FaceAPI Tutorial. Available at: https://vladmandic.github.io/face-api/tutorial/ (Accessed: 9 November 2023).

[10] Benyahia, A. (2011) Boost Your Website's Capabilities with Real-Time Facial Recognition Using Face-API.js and Vanilla JavaScript. Available at: https://javascript.plainenglish.io/boost-your-websites-capabilities-with-real-time-facial-recognition-using-face-api-js-46261ec464e2 (Accessed: 09/11/2023).

[11] Mühler, V. (2020) face-api.js - JavaScript API for face detection and face recognition in the browser and nodejs with tensorflow.js. Available at: https://github.com/justadudewhohacks/face-api.js(Accessed: 09/11/2023).

[12] Mühler, V. (2020). face-api.js — JavaScript API for Face Recognition in the Browser with tensorflow.js. [Online]. Available at: 1. (Accessed: 9 November 2023)

[13] Ayala, R. (2020) Facial Recognition with JavaScript. Available at: https://reynaldo-ayala.medium.com/facial-recognition-with-javascript-4bf928320957 (Accessed: 09/11/2023).

8. APPENDICES

There were five participants and individuals had their data removed. The tests were carried out on these areas, accuracy, login, and expression.

Participants	Accuracy	Login	Expression
Participant 1	100%	Succeed	Detected
Participant 2	100%	Succeed	Detected
Participant 3	100%	Succeed	Detected
Participant 4	100%	Succeed	Detected
Participant 5	100%	Succeed	Detected