**PROJECT REPORT**

**Django based emotion detection using Deepface library**

On

Submitted in partial fulfilment of the requirements for the award of the diploma of

**COMPUTER ENGINEERING**

**By**

D.Uttej sai vikram 20014-CM-015

D.Jhusheenath chandrshekar 20014-CM-016

M.Sai Rajesh 20014-CM-068

Sk.Irshad Ahamad 20014-CM-053

CH.Keerthi 20014-cm-013

P.Pujitha 20014-CM-046

B.Prasanth kumar 20014-CM-010

**Under the Esteemed Guidance of**

**Y.Kiriti babu**

**DEPARTMENT OF COMPUTER ENGINEERING**

**MBTS GOVT POLYTECHNIC COLLEGE**

**NALLAPADU, GUNTUR-522005**

**2020-2023**

**MBTS GOVT POLYTECHNIC COLLEGE**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CERTIFICATE**

This is to certify that the project entitled **“Django based emotion detection using deepface library”**,is a bonafide work D.Uttej sai vikram(20014-CM-015), D.Jhusheenath chandrshekar(20014-CM-016),M.Sai Rajesh (20014-CM-068),Sk.Irshad Ahamad (20014-CM-053),CH.Keerthi (20014-cm-013),P.Pujitha(20014-CM-046),B.Prasanth kumar (20014-CM-010),for the partial fulfilment of the requirements for the award of degree of in **COMPUTER ENGINEERING** from **MBTS GOVT POLYTECHNIC COLLEGE.**

**Project Mentor:- Head Of The Department(CME):-**

**Y.Kirti babu Y.Ravindra Babu**

### DECLARATION

This is to certify that ***D.Uttej sai vikram(20014-CM-015), D.Jhusheenath chandrshekar(20014-CM-016),M.Sai Rajesh (20014-CM-068),Sk.Irshad Ahamad (20014-CM-053),CH.Keerthi (20014-cm-013),P.Pujitha(20014-CM-046),B.Prasanth kumar (20014-CM-010)*,**are completed a Project entitled “**Django based emotion detection using deepface library”** for the partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Engineering affiliated to is the result of work done by us under the guidance of**, Y.Kiriti babu**

Place:-Guntur

Date:-

D.Uttej sai vikram 20014-CM-015

B.Prasanth kumar 20014-CM-010

D.Jhusheenath chandrshekar 20014-CM-016

M.Sai Rajesh 20014-CM-068

Sk.Irshad Ahamad 20014-CM-053

CH.Keerthi 20014-CM-013

P.Pujitha 20014-CM-046

**ACKNOWLEDGEMENT**

We Consider it our privilege to express our gratitude to all those who guided and inspired us in the completion of this Project .We express our sincere thanks to our beloved **PRINCIPAL SRI TALLURI SHEKAR** for providing support and stimulating environment for developing the project.

It is with immense pleasure that we would like to express our indebted gratitude to our guide **Y.KIRITI BABU** Who has guided us a lot and encouraged us in every step of the project work. he has given moral support and guidance throughout the project helped us to a great extent.

Our sincere thanks to **Y.Ravindra Babu Sir HOD DEPT OF CME** for his co-operation and guidance in helping us to make our project successful and complete in all aspects we are grateful to her precious guidance and suggestions.

We also thank all the faculty of the Department of Computer Engineering for their help and guidance of numerous occasions, which has given us the cogency to build-up adamant aspiration over the completion of our project thesis.

### Finally, we thank one and all who directly or indirectly helped us to complete our project thesis successfully.

**PROJECT ASSOCIATIVES**

D.Uttej sai vikram 20014-CM-015

D.Jhusheenath chandrshekar 20014-CM-016

M.Sai Rajesh 20014-CM-068

Sk.Irshad Ahamad 20014-CM-053

CH.Keerthi 20014-CM-013

P.Pujitha 20014-CM-046

B.Prasanth kumar 20014-CM-010

**ABSTRACT**

This project presents a Django-based emotion detection system using the deepface library. Emotion detection plays a crucial role in various applications, including human-computer interaction, affective computing, and sentiment analysis. The deepface library offers pre-trained deep learning models that can accurately recognize facial expressions and infer emotions from images. By integrating this library into a Django web application, users can easily upload images and receive real-time emotion analysis results. The system leverages Django's framework for handling web requests, while deepface provides the underlying deep learning capabilities for emotion detection. The project demonstrates the seamless integration of these technologies to create a user-friendly and efficient emotion detection system

**TABLE OF CONTENTS**

S.NO CONTENTS pg.no

1 Introduction ………………………………………… **6**

2 Literature Survey.......................................................... 7-10

3 exisisting system vs proposed system……………… 10-13

4 SRS DOCUMENT………………………………… 3-16

5 Feasability Study ………………………………… 16-18

6 Methodology……………………………………… 18-60

7 Source Code……………………………………… 60-78

8 Testing…………………………………………… 78-79

9 conclusion………………………………………… 80

10 references………………………………………… 81

**INTRODUCTION**

The project "Django-based Emotion Detection using DeepFace library" combines the power of Django, a popular web framework, with the DeepFace library to create an intelligent web application capable of detecting emotions from facial images. Emotion detection has become an increasingly important field of study, with applications ranging from market research and customer sentiment analysis to improving human-computer interactions. By leveraging the capabilities of deep learning models and the user-friendly nature of Django, this project aims to provide a seamless and accurate emotion detection solution.

Understanding human emotions plays a crucial role in various domains, including psychology, marketing, and human-computer interaction. Traditionally, emotion detection required manual observation and interpretation by human experts, which is time-consuming and subject to bias. However, advancements in deep learning and computer vision have paved the way for automated emotion detection systems. These systems analyze facial expressions and extract meaningful features to classify emotions with high accuracy.

The DeepFace library, a powerful deep learning-based facial recognition and analysis library, forms the foundation of this project. It leverages pre-trained deep neural networks, which have been trained on extensive datasets, to extract facial features and predict emotions from images. By utilizing DeepFace within a Django web application, users can easily upload their facial images and obtain emotion predictions in real-time.

The Django framework provides the ideal platform for developing this emotion detection system. Known for its simplicity, scalability, and robustness, Django simplifies the development process, allowing us to focus on the core functionalities of the application. It provides features such as URL routing, database integration, and user authentication, which are essential for building a complete web application. With Django's intuitive administrative interface, managing the application's backend and user data becomes effortless.

The objectives of this project encompass multiple aspects. Firstly, we aim to develop a user-friendly web application using Django that provides an intuitive interface for users to upload their facial images and receive emotion predictions. Secondly, the integration of the DeepFace library enables accurate emotion detection by leveraging deep learning models trained on vast datasets. Additionally, we focus on optimizing the performance of the application by implementing techniques such as caching, model optimization, and parallel processing.

By combining the strengths of Django and the DeepFace library, this project aims to contribute to the field of emotion detection by providing a reliable and accessible web application. The ability to detect and understand emotions from facial expressions has wide-ranging applications, from market research and sentiment analysis to improving user experiences in various domains. Through this project, we hope to offer a valuable tool that enhances our understanding of human emotions and facilitates better interaction between humans and technology.

**2.Literature survey**

**LITERATURE REVIEW**:-

This literature review aims to provide an overview of the existing research on face emotion detection using Python and highlight the key methodologies, techniques, and challenges in this area.

Feature Extraction Techniques:-

Numerous feature extraction methods have been explored in face emotion detection. One commonly used technique is the Local Binary Patterns (LBP) algorithm, which captures texture information from facial images.

Other techniques include Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Principal Component Analysis (PCA). These techniques help in extracting discriminative facial features for emotion classification.

Machine Learning Approaches: -

Machine learning algorithms play a crucial role in training models for face emotion detection. Support Vector Machines (SVM), Random Forest, k-Nearest Neighbors (k-NN), and Convolutional Neural Networks (CNN) are commonly employed. SVM and k-NN are traditional algorithms that utilize handcrafted features, while CNNs leverage deep learning to automatically learn features from raw image data.

The use of deep learning models, particularly CNNs, has shown promising results in achieving high accuracy in emotion recognition tasks.

Datasets for Training and Evaluation:

Several datasets have been developed and used for training and evaluating face emotion detection models. Examples include the Facial Expression Recognition Challenge (FERC), CK+ dataset, and the Extended Cohn-Kanade (CK+) dataset.

These datasets provide labeled facial images with corresponding emotions, enabling researchers to train and evaluate their models effectively.

Preprocessing Techniques:

Preprocessing techniques are crucial for enhancing the quality and accuracy of face emotion detection systems. Common preprocessing steps include face detection and alignment, facial landmark detection, and image normalization.

These techniques help in isolating the face region, normalizing variations in pose and illumination, and aligning facial landmarks for accurate feature extraction.

Challenges and Future Directions:

Despite significant progress, face emotion detection still faces several challenges. These include handling occlusions, robustness to variations in pose, lighting conditions, and facial expressions, as well as handling individual differences.

Future research should focus on developing more robust and accurate models that can handle real-world scenarios and improve generalization across different datasets and populations.

Additionally, the combination of multimodal approaches, such as incorporating audio and textual information, can provide more comprehensive emotion recognition systems.

**EXISTING SYSTEM VS PROPOSED SYSTEM**

**EXISTING SYSTEM:-**

The existing system for face emotion detection using Python may vary depending on the specific methods and techniques employed. However, one common approach is to utilize pre-trained deep learning models such as Convolutional Neural Networks (CNNs) to classify facial expressions. Popular pre-trained models for this task include VGGNet, ResNet, and Inception.

The existing systems in face emotion detection using Python typically utilize traditional computer vision techniques combined with machine learning algorithms. These systems often rely on handcrafted features and use classifiers such as Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), or Random Forest for emotion classification.

They may employ feature extraction techniques like Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG), or Principal Component Analysis (PCA) to capture facial information. Preprocessing steps like face detection, alignment, and normalization are also performed to improve accuracy.

While these existing systems have shown reasonably good performance in face emotion detection, they often face challenges in handling variations in pose, lighting conditions, occlusions, and individual differences.

They may struggle to generalize well across different datasets and populations due to limited feature representation and manual feature engineering. Additionally, these systems may not fully leverage the power of deep learning and may require extensive tuning and parameter optimization to achieve optimal results.

In the existing system, the workflow typically involves the following steps:-

Preprocessing: The input image is preprocessed to enhance the quality and extract relevant facial features. This may include resizing, normalization, and possibly applying filters for noise reduction.

Face Detection: A face detection algorithm is used to locate and extract faces from the input image. This is typically done using techniques like Haar cascades, HOG (Histogram of Oriented Gradients), or deep learning-based face detectors like MTCNN (Multi-task Cascaded Convolutional Networks).

Facial Expression Classification: The pre-trained CNN model is utilized to classify the facial expressions of the detected faces. The CNN takes the face image as input and predicts the emotion label, such as happy, sad, angry, etc.

Post-processing: The predicted emotion labels are post-processed as per the requirements of the application. This may include mapping the labels to specific emotions, aggregating results for multiple faces, or performing additional analysis.

Limitations of the Existing System:-

Dependency on pre-trained models: The performance of the existing system heavily relies on the accuracy and generalization of the pre-trained models used. These models may not always capture the full range of emotions or perform optimally for specific datasets or scenarios.

Limited adaptability: The existing system may lack the ability to adapt and improve over time. Once the pre-trained model is chosen, it may not easily incorporate new data or adapt to changing conditions.

Lack of explainability: Deep learning models often lack interpretability, making it challenging to understand the reasoning behind their predictions.

🡪Relies on traditional computer vision techniques and machine learning algorithms.

🡪Uses handcrafted features and classifiers like SVM, k-NN, or Random Forest.

🡪Performs preprocessing steps such as face detection, alignment, and normalization.

🡪Faces challenges in handling variations in pose, lighting conditions, occlusions, and individual differences.

🡪Limited feature representation and manual feature engineering.

🡪May struggle to generalize well across different datasets and populations.

Top of Form

Bottom of Form

**PROPOSED SYSTEM:-**

The proposed system for face emotion detection using Python can address some of the limitations of the existing system. One potential approach is to employ NEAT (NeuroEvolution of Augmenting Topologies) to evolve neural networks for emotion classification. NEAT is a genetic algorithm-based technique that evolves neural network architectures and weights to optimize a specific task.

The proposed system in face emotion detection using Python aims to overcome the limitations of the existing systems by incorporating advanced deep learning techniques and leveraging large-scale datasets.

Instead of relying on handcrafted features, the proposed system utilizes Convolutional Neural Networks (CNNs) for both feature learning and classification. CNNs are capable of automatically learning discriminative features directly from raw image data, thereby eliminating the need for manual feature engineering.

The proposed system can employ pre-trained CNN models such as VGGNet, ResNet, or InceptionNet, which have been trained on large-scale datasets like ImageNet. These pre-trained models can be fine-tuned or used as feature extractors for face emotion detection tasks.

By using deep learning, the proposed system can capture complex spatial relationships in facial expressions, improve robustness to variations, and enhance overall accuracy.

Furthermore, the proposed system can benefit from the availability of large and diverse datasets specifically designed for face emotion detection, such as the Facial Expression Recognition Challenge (FERC) dataset or the CK+ dataset. These datasets provide a broader range of facial expressions and emotions, facilitating more comprehensive training and evaluation of the proposed system.

The integration of multimodal approaches, such as combining facial information with audio or textual data, can also be explored in the proposed system. This can provide a more comprehensive understanding of emotions and improve the system's performance in real-world scenarios.

In summary, the proposed system in face emotion detection using Python aims to leverage deep learning techniques, large-scale datasets, and potentially multimodal information to enhance accuracy, robustness, and generalization capabilities compared to the existing systems that rely on handcrafted features and traditional machine learning algorithms.

The proposed system may involve the following components:-

NEAT-based Architecture: Define the NEAT configuration, including the population size, mutation rates, and other parameters. Specify the input and output dimensions of the neural networks.

Fitness Function: Design a fitness function that evaluates the performance of each neural network based on its ability to classify facial expressions accurately. This function measures the network's success in capturing relevant features and mapping them to correct emotion labels.

Training and Evolution: Use NEAT to evolve neural networks over multiple generations. Evaluate and select the fittest networks for reproduction, apply mutation and crossover operations, and create new variations of networks. The process continues iteratively, gradually improving the networks' performance.

Testing and Evaluation: Assess the performance of the evolved neural networks on a separate test dataset. Measure metrics such as accuracy, precision, recall, or F1-score to evaluate the effectiveness of the proposed system.

Advantages of the Proposed System:-

Adaptability and Continuous Improvement: NEAT allows the neural networks to adapt and improve over time through evolution. This enables the system to incorporate new data, optimize the network architectures, and potentially enhance accuracy and performance.

Customization: The proposed system offers more flexibility in designing neural network architectures and tailoring them specifically for face emotion detection. This customization can potentially result in improved performance compared to relying solely on pre-trained models.

Interpretability: NEAT-based systems can provide insights into the evolved neural networks' structure and functioning.

🡪Incorporates advanced deep learning techniques, particularly Convolutional Neural Networks (CNNs).

🡪Utilizes pre-trained CNN models or fine-tuned CNNs for feature learning and classification.

🡪Captures complex spatial relationships in facial expressions and improves robustness.

🡪Leverages large-scale datasets specifically designed for face emotion detection.

🡪Explores multimodal approaches by integrating facial, audio, or textual data.

🡪Enhances accuracy, robustness, and generalization capabilities.

🡪Eliminates the need for manual feature engineering and improves feature representation.

**3.SOFTWARE REQUIREMENETS ANALYSIS**

**1. Introduction**

**1.1 Purpose:**

The purpose of this document is to provide a comprehensive overview of the requirements and specifications for the project "Django-based Emotion Detection using DeepFace Library." It outlines the features, constraints, interfaces, and system models of the application.

**1.2 Scope:**

The project aims to develop a web application using the Django framework for emotion detection from facial images. It utilizes the DeepFace library, integrating pre-trained deep learning models to accurately classify emotions. The application will allow users to upload images, perform emotion detection, and display the results.

**1.3 Document Conventions:**

- The document follows standard industry conventions and practices for documenting software requirements.

- Use of UML diagrams for system modeling.

**1.4 Intended Audience and Reading Suggestions:**

This document is intended for the project development team, stakeholders, and individuals involved in the design, development, and testing of the application. Readers are recommended to review the document in its entirety to gain a comprehensive understanding of the project requirements.

**1.5 Project Overview:**

The Django-based Emotion Detection project aims to create a web application that utilizes the DeepFace library for accurately detecting emotions from facial images. The application will provide a user-friendly interface for image upload, emotion detection, and displaying the results. The project will leverage the Django framework's capabilities for web development, incorporating pre-trained deep learning models to achieve high accuracy in emotion classification.

**2. Overall Description**

**2.1 Product Perspective:**

The Django-based Emotion Detection application will function as a standalone web application, interacting with users through a web browser. It will utilize the DeepFace library for facial image analysis and emotion detection. The application will be developed using the Django web framework and will integrate pre-trained deep learning models to provide accurate emotion predictions.

**2.2 Product Features:**

The key features of

the Django-based Emotion Detection application include:

- User Registration and Authentication: Users can create accounts and log in to the application.

- Image Upload: Users can upload facial images for emotion detection.

- Emotion Detection: The application will analyze uploaded images and predict the emotions present.

- Displaying Emotion Results: Emotion results will be displayed to the users in an intuitive format.

- Image Analysis History: The application will maintain a history of previously analyzed images.

- Admin Dashboard: An administrative dashboard will be provided to manage user accounts and view usage statistics.

**2.3 User Classes and Characteristics:**

The application is designed for the following user classes:

- General Users: Users who want to upload facial images for emotion detection.

- Administrators: Users with administrative privileges to manage user accounts and monitor application usage.

**2.4 Operating Environment:**

The Django-based Emotion Detection application will operate in a web environment, accessible through modern web browsers. The application will be compatible with common operating systems such as Windows, macOS, and Linux. It will require Python, Django, and other necessary dependencies to be installed on the server.

**2.5 Design and Implementation Constraints:**

- The application will utilize the DeepFace library for emotion detection and will be constrained by the capabilities and limitations of this library.

- The system must ensure the privacy and security of user data, adhering to legal and regulatory requirements.

- The application's performance should be optimized to handle multiple concurrent user requests efficiently.

**2.6 User Documentation:**

The application will provide user documentation in the form of a user guide or online help system. This documentation will guide users on how to use the application, upload images, interpret emotion results, and manage their accounts.

**2.7 Assumptions and Dependencies:**

- The project assumes the availability of a suitable server infrastructure to host the Django-based web application.

- The DeepFace library and its dependencies must be installed and configured correctly.

- The application assumes an internet connection for accessing the web application and performing emotion detection.

**3. System Features**

**3.1 Feature 1: User Registration and Authentication**

- Users can create new accounts by providing necessary details.

- Users can log in using their credentials.

- The system will enforce secure password storage and authentication mechanisms.

**3.2 Feature 2: Image Upload**

- Users can upload facial images for emotion detection.

- The application will support common image formats such as JPEG and PNG.

- The system will validate the uploaded images for format and size constraints.

**3.3 Feature 3: Emotion Detection**

- The application will process the uploaded images using the DeepFace library for emotion detection.

- Deep learning models will analyze facial features to predict emotions.

- The system will handle image preprocessing and data transformation for input to the deep learning models.

**3.4 Feature 4: Displaying Emotion Results**

- The application will present the predicted emotions to the user.

- Emotion results will be displayed in a user-friendly format, potentially using visual aids such as graphs or charts.

- The system will provide additional details and insights about the detected emotions.

**3.5 Feature 5: Image Analysis History**

- The application will maintain a history of previously analyzed images.

- Users can view their past image uploads and the corresponding emotion results.

- The system will support filtering and sorting options for efficient navigation of the image analysis history.

**3.6 Feature 6: Admin Dashboard**

- Administrators will have access to a dedicated dashboard for managing user accounts.

- The dashboard will provide statistical insights into application usage and image analysis trends.

- Administrators can perform actions such as user management and system configuration.

**4. External Interface Requirements**

**4.1 User Interfaces:**

- The user interface will be web-based, accessible through standard web browsers.

- The application will provide an intuitive and responsive user interface for image

upload, emotion detection, and result display.

- The user interface will support interactions such as clicking, uploading files, and navigating through different application sections.

**4.2 Hardware Interfaces:**

- The application will run on standard server hardware with adequate processing power and memory to handle concurrent user requests.

- The user's device should have a functional web browser and an internet connection to access the application.

**4.3 Software Interfaces:**

- The application will be built using the Django web framework and the Python programming language.

- The DeepFace library will be integrated into the application for emotion detection.

- The application will interface with the database management system for storing user data and image analysis history.

**4.4 Communication Interfaces:**

- The application will utilize HTTP/HTTPS protocols for communication between the user's device and the web server.

- Secure communication channels will be established for user authentication and protection of sensitive data.

**5. Non-functional Requirements**

**5.1 Performance Requirements:**

- The application should provide real-time or near-real-time emotion detection for uploaded images.

- The system should be able to handle multiple concurrent user requests efficiently.

- Emotion detection should achieve high accuracy in predicting emotions from facial images.

**5.2 Security Requirements:**

- The application should enforce secure user authentication and password storage mechanisms.

- User data and uploaded images should be protected through secure communication channels and encryption.

- The system should adhere to privacy regulations and ensure the confidentiality of user information.

**5.3 Reliability Requirements:**

- The application should be available and accessible to users with minimal downtime.

- The system should handle exceptions and errors gracefully, providing meaningful error messages to users.

- The application should have backup and recovery mechanisms to prevent data loss.

**5.4 Maintainability Requirements:**

- The codebase should follow best practices and coding conventions to ensure maintainability.

- The application should provide easy deployment and installation processes.

- The system should support future updates, improvements, and integration with new technologies.

**5.5 Scalability Requirements:**

- The application should be scalable to accommodate increasing user demand.

- The system should handle a growing image analysis history efficiently.

**5.6 Usability Requirements:**

- The user interface should be intuitive, user-friendly, and responsive.

- The application should provide clear instructions and feedback to guide users throughout the process.

- The system should support multiple languages and internationalization features.

**6. System Models**

**6.1 Use Case Diagram:**

[Include a Use Case Diagram here illustrating the interactions between users and the system, showcasing major use cases such as user registration, image upload, emotion detection, etc.]

**6.2 Class Diagram:**

[Include a Class Diagram here representing the major classes and their relationships within the system.]

**6.3 Sequence Diagrams:**

[Include Sequence Diagrams here to demonstrate the interactions between system components and users for specific use cases such as user registration, image upload, and emotion detection.]

**6.4 Activity Diagrams:**

[Include Activity Diagrams here to illustrate the workflow of major system processes such as image analysis, emotion detection, and result display.]

**7. Other Requirements**

**7.1 Legal and Regulatory Requirements:**

- The application should comply with applicable laws and regulations regarding data privacy and security.

- User consent should be obtained for using and storing their uploaded images and personal information.

**7.2 Privacy Requirements:**

- The application should provide clear privacy policies and inform users about how their data will be used.

- Users should have control over their data, including the ability to delete uploaded images and terminate their accounts.

**7.3 Ethical Requirements:**

- The system should be designed and implemented in an ethical manner, respecting user privacy and ensuring unbiased emotion detection.

- The application should not discriminate or harm users based on their ethnicity, gender, or other attributes.

This Software Requirements Specification (SRS) document provides a comprehensive overview of the project "Django-based

Emotion Detection using DeepFace Library." It outlines the purpose, scope, features, interfaces, and system models of the application. The document also includes non-functional requirements, such as performance, security, reliability, maintainability, scalability, and usability. Additionally, it addresses legal, regulatory, privacy, and ethical considerations.

**4.Feasibility Study**

1. Introduction:

The feasibility study aims to assess the viability and practicality of the project "Django-based Emotion Detection using DeepFace Library." This study evaluates various aspects to determine if the project can be successfully implemented and if it aligns with the organization's goals and objectives.

2. Objectives of the Project:

The objectives of the project are as follows:

a) Develop a Django-based web application for emotion detection.

b) Integrate the DeepFace library for accurate emotion classification.

c) Provide users with a user-friendly interface for image upload and emotion result display.

d) Maintain a history of analyzed images for reference and analysis.

e) Implement an admin dashboard for user management and system monitoring.

3. Technical Feasibility:

a) Software Requirements: The project requires Python, Django, and the DeepFace library. These technologies have matured and have extensive community support.

b) Development Team: A skilled development team familiar with Django, Python, and deep learning concepts is essential for successful implementation.

c) Hardware Requirements: The application can be deployed on standard server hardware, and user devices only require a web browser and an internet connection.

4. Economic Feasibility:

a) Cost Analysis: The project's cost involves personnel, infrastructure, and software licensing. The costs should be analyzed against the potential benefits and return on investment.

b) Revenue Generation: The project's revenue potential can be explored through various avenues, such as offering the application as a paid service, implementing advertisements, or providing premium features for a fee.

5. Operational Feasibility:

a) User Acceptance: The project's success relies on user adoption and satisfaction. User feedback and usability testing should be conducted to ensure the application meets user expectations.

b) Training and Support: Proper documentation and user guides should be provided to assist users in understanding and effectively utilizing the application. Adequate support channels should be established to address user queries and issues.

6. Legal and Ethical Feasibility:

a) Data Privacy and Security: The project should comply with privacy regulations and implement measures to protect user data. Clear consent and usage policies should be provided to users.

b) Ethical Considerations: The application should be developed in an ethical manner, ensuring fairness, inclusivity, and non-discrimination in emotion detection algorithms.

7. Schedule Feasibility:

a) Timeline: A realistic project timeline should be developed, considering factors such as development, testing, deployment, and potential iterations based on user feedback.

b) Resource Allocation: Adequate resources, including personnel, infrastructure, and budget, should be allocated to meet the project's timeline and objectives.

8. Conclusion:

Based on the feasibility study, the project "Django-based Emotion Detection using DeepFace Library" appears to be feasible and promising. The technical requirements are readily available, and the project aligns with the organization's objectives. Economic, operational, legal, and ethical aspects have been addressed, and a realistic schedule can be developed for successful implementation. However, continuous monitoring, user feedback, and updates should be incorporated to ensure the application's long-term viability and relevance in the dynamic field of emotion detection.

**5.methodlogy(all languages and frame works)**

**4.1 INTRODUCTION TO DJANGO**

Django is a Web Application Framework which is used to develop web applications. Our Django Tutorial includes all topics of Django such as introduction, features, installation, environment setup, admin interface, cookie, form validation, Model, Template Engine, Migration, MVT etc. All the topics are explained in detail so that reader can get enough knowledge of Django.

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.This framework uses a famous tag line:

**The web framework for perfectionists with deadlines.**

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

Django was design and developed by Lawrence journal world in 2003 and publicly released under BSD license in July 2005. Currently, DSF (Django Software Foundation) maintains its development and release cycle. Django was released on 21, July 2005. Its current stable version is 2.0.3 which was released on 6 March, 2018.

Django is widely accepted and used by various well-known sites such as:

* Instagram
* Mozilla
* Disqus
* Pinterest
* Bitbucket
* The Washington Times

# Features of Django

* Rapid Development
* Secure
* Scalable
* Fully loaded
* Versatile
* Open Source
* Vast and Supported Community

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

Django is an open-source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

To install Django, first visit to **Django official site (https://www.djangoproject.com)** and download Django by clicking on the download section. Here, we will see various options to download The Django.

Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions **pip3** is used to manage packages.we are installing Django in Ubuntu operating system.

The complete installation process is described below. Before installing make sure **pip is installed** in local system.Here, we are installing Django using pip3, the installation command is given below.

$ pip3 install Django==2.0.3

After installing Django, we need to verify the installation. Open terminal and write **python3** and press enter. It will display python shell where we can verify Django installation. Look at the Django version displayed by the print method of the python. Well, Django is installed successfully. Now, we can build Django web applications.

## The model layer

Django provides an abstraction layer (the “models”) for structuring and manipulating the data of your web application. Learn more about it below:

* Models: Introduction to models|Field types|Indexes|Meta options|Model class
* QuerySets: Making queries|QuerySet method reference|Lookup expressions
* Model instances: Instance methods|Accessing related objects
* Migrations:Introductionto Migrations|Operations reference|SchemaEditor|Writing migrations
* Advanced:Managers|RawSQL|Transactions|Aggregation|Search|Custom fields|Multipledatabases|Customlookups|QueryExpressions|Conditional Expressions | Database Functions
* Other: Supported databases|Legacy databases|Providing initial data|Optimize database access|PostgreSQL specific features.

## The view layer

Django has the concept of “views” to encapsulate the logic responsible for processing a user’s request and for returning the response. Find all you need to know about views via the links below:

* **Thebasics:** [URLconfs](https://docs.djangoproject.com/en/4.0/topics/http/urls/) | [View functions](https://docs.djangoproject.com/en/4.0/topics/http/views/) | [Shortcuts](https://docs.djangoproject.com/en/4.0/topics/http/shortcuts/) | [Decorators](https://docs.djangoproject.com/en/4.0/topics/http/decorators/) | [Asynchronous Support](https://docs.djangoproject.com/en/4.0/topics/async/)
* **Reference:** [Built-in Views](https://docs.djangoproject.com/en/4.0/ref/views/) | [Request/response objects](https://docs.djangoproject.com/en/4.0/ref/request-response/) | [TemplateResponse objects](https://docs.djangoproject.com/en/4.0/ref/template-response/)
* **File uploads:** [Overview](https://docs.djangoproject.com/en/4.0/topics/http/file-uploads/) | [File objects](https://docs.djangoproject.com/en/4.0/ref/files/file/) | [Storage API](https://docs.djangoproject.com/en/4.0/ref/files/storage/) | [Managing files](https://docs.djangoproject.com/en/4.0/topics/files/) | [Custom storage](https://docs.djangoproject.com/en/4.0/howto/custom-file-storage/)
* **Class-based views:** [Overview](https://docs.djangoproject.com/en/4.0/topics/class-based-views/) | [Built-in display views](https://docs.djangoproject.com/en/4.0/topics/class-based-views/generic-display/) | [Built-in editing views](https://docs.djangoproject.com/en/4.0/topics/class-based-views/generic-editing/) | [Using mixins](https://docs.djangoproject.com/en/4.0/topics/class-based-views/mixins/) | [API reference](https://docs.djangoproject.com/en/4.0/ref/class-based-views/) | [Flattened index](https://docs.djangoproject.com/en/4.0/ref/class-based-views/flattened-index/)
* **Advanced:** [Generating CSV](https://docs.djangoproject.com/en/4.0/howto/outputting-csv/) | [Generating PDF](https://docs.djangoproject.com/en/4.0/howto/outputting-pdf/)
* **Middleware:** [Overview](https://docs.djangoproject.com/en/4.0/topics/http/middleware/) | [Built-in middleware classes](https://docs.djangoproject.com/en/4.0/ref/middleware/)

## The template layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. Learn how this syntax can be used by designers and how it can be extended by programmers:

* **The basics:** [Overview](https://docs.djangoproject.com/en/4.0/topics/templates/)
* **For designers:** [Language overview](https://docs.djangoproject.com/en/4.0/ref/templates/language/) | [Built-in tags and filters](https://docs.djangoproject.com/en/4.0/ref/templates/builtins/) | [Humanization](https://docs.djangoproject.com/en/4.0/ref/contrib/humanize/)
* **For programmers:** [Template API](https://docs.djangoproject.com/en/4.0/ref/templates/api/) | [Custom tags and filters](https://docs.djangoproject.com/en/4.0/howto/custom-template-tags/) | [Custom template backend](https://docs.djangoproject.com/en/4.0/howto/custom-template-backend/)

## Forms

Django provides a rich framework to facilitate the creation of forms and the manipulation of form data.

* **The basics:** [Overview](https://docs.djangoproject.com/en/4.0/topics/forms/) | [Form API](https://docs.djangoproject.com/en/4.0/ref/forms/api/) | [Built-in fields](https://docs.djangoproject.com/en/4.0/ref/forms/fields/) | [Built-in widgets](https://docs.djangoproject.com/en/4.0/ref/forms/widgets/)
* **Advanced:** [Forms for models](https://docs.djangoproject.com/en/4.0/topics/forms/modelforms/) | [Integrating media](https://docs.djangoproject.com/en/4.0/topics/forms/media/) | [Formsets](https://docs.djangoproject.com/en/4.0/topics/forms/formsets/) | [Customizing validation](https://docs.djangoproject.com/en/4.0/ref/forms/validation/)

## The development process

Learn about the various components and tools to help you in the development and testing of Django applications:

* **Settings:** [Overview](https://docs.djangoproject.com/en/4.0/topics/settings/) | [Full list of settings](https://docs.djangoproject.com/en/4.0/ref/settings/)
* **Applications:** [Overview](https://docs.djangoproject.com/en/4.0/ref/applications/)
* **Exceptions:** [Overview](https://docs.djangoproject.com/en/4.0/ref/exceptions/)
* **django-admin and manage.py:** [Overview](https://docs.djangoproject.com/en/4.0/ref/django-admin/) | [Adding custom commands](https://docs.djangoproject.com/en/4.0/howto/custom-management-commands/)
* **Testing:** [Introduction](https://docs.djangoproject.com/en/4.0/topics/testing/) | [Writing and running tests](https://docs.djangoproject.com/en/4.0/topics/testing/overview/) | [Included testing tools](https://docs.djangoproject.com/en/4.0/topics/testing/tools/) | [Advanced topics](https://docs.djangoproject.com/en/4.0/topics/testing/advanced/)
* **Deployment:** [Overview](https://docs.djangoproject.com/en/4.0/howto/deployment/) | [WSGI servers](https://docs.djangoproject.com/en/4.0/howto/deployment/wsgi/) | [ASGI servers](https://docs.djangoproject.com/en/4.0/howto/deployment/asgi/) | [Deploying static files](https://docs.djangoproject.com/en/4.0/howto/static-files/deployment/) | [Tracking code errors by email](https://docs.djangoproject.com/en/4.0/howto/error-reporting/) | [Deployment checklist](https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/)

## The admin

## Find all you need to know about the automated admin interface, one of Django’s most popular features:

* [Admin site](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/)
* [Admin actions](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/actions/)
* [Admin documentation generator](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/admindocs/)

## Security

Security is a topic of paramount importance in the development of web applications and Django provides multiple protection tools and mechanisms:

* [Security overview](https://docs.djangoproject.com/en/4.0/topics/security/)
* [Disclosed security issues in Django](https://docs.djangoproject.com/en/4.0/releases/security/)
* [Clickjacking protection](https://docs.djangoproject.com/en/4.0/ref/clickjacking/)
* [Cross Site Request Forgery protection](https://docs.djangoproject.com/en/4.0/ref/csrf/)
* [Cryptographic signing](https://docs.djangoproject.com/en/4.0/topics/signing/)
* [Security Middleware](https://docs.djangoproject.com/en/4.0/ref/middleware/#security-middleware)

## Internationalization and localization

Django offers a robust internationalization and localization framework to assist you in the development of applications for multiple languages and world regions:

* [Overview](https://docs.djangoproject.com/en/4.0/topics/i18n/) | [Internationalization](https://docs.djangoproject.com/en/4.0/topics/i18n/translation/) | [Localization](https://docs.djangoproject.com/en/4.0/topics/i18n/translation/#how-to-create-language-files) | [Localized web UI formatting and form input](https://docs.djangoproject.com/en/4.0/topics/i18n/formatting/)
* [Time zones](https://docs.djangoproject.com/en/4.0/topics/i18n/timezones/)

## Performance and optimization

There are a variety of techniques and tools that can help get your code running more efficiently - faster, and using fewer system resources.

* [Performance and optimization overview](https://docs.djangoproject.com/en/4.0/topics/performance/)

## Geographic framework

[GeoDjango](https://docs.djangoproject.com/en/4.0/ref/contrib/gis/) intends to be a world-class geographic web framework. Its goal is to make it as easy as possible to build GIS web applications and harness the power of spatially enabled data.

## Common web application tools

Django offers multiple tools commonly needed in the development of web applications:

* **Authentication:** [Overview](https://docs.djangoproject.com/en/4.0/topics/auth/) | [Using the authentication system](https://docs.djangoproject.com/en/4.0/topics/auth/default/) | [Password management](https://docs.djangoproject.com/en/4.0/topics/auth/passwords/) | [Customizing authentication](https://docs.djangoproject.com/en/4.0/topics/auth/customizing/) | [API Reference](https://docs.djangoproject.com/en/4.0/ref/contrib/auth/)
* [Caching](https://docs.djangoproject.com/en/4.0/topics/cache/)
* [Logging](https://docs.djangoproject.com/en/4.0/topics/logging/)
* [Sending emails](https://docs.djangoproject.com/en/4.0/topics/email/)
* [Syndication feeds (RSS/Atom)](https://docs.djangoproject.com/en/4.0/ref/contrib/syndication/)
* [Pagination](https://docs.djangoproject.com/en/4.0/topics/pagination/)
* [Messages framework](https://docs.djangoproject.com/en/4.0/ref/contrib/messages/)
* [Serialization](https://docs.djangoproject.com/en/4.0/topics/serialization/)
* [Sessions](https://docs.djangoproject.com/en/4.0/topics/http/sessions/)
* [Sitemaps](https://docs.djangoproject.com/en/4.0/ref/contrib/sitemaps/)
* [Static files management](https://docs.djangoproject.com/en/4.0/ref/contrib/staticfiles/)
* [Data validation](https://docs.djangoproject.com/en/4.0/ref/validators/)

## Other core functionalities

Learn about some other core functionalities of the Django framework:

* [Conditional content processing](https://docs.djangoproject.com/en/4.0/topics/conditional-view-processing/)
* [Content types and generic relations](https://docs.djangoproject.com/en/4.0/ref/contrib/contenttypes/)
* [Flatpages](https://docs.djangoproject.com/en/4.0/ref/contrib/flatpages/)
* [Redirects](https://docs.djangoproject.com/en/4.0/ref/contrib/redirects/)
* [Signals](https://docs.djangoproject.com/en/4.0/topics/signals/)
* [System check framework](https://docs.djangoproject.com/en/4.0/topics/checks/)
* [The sites framework](https://docs.djangoproject.com/en/4.0/ref/contrib/sites/)
* [Unicode in Django](https://docs.djangoproject.com/en/4.0/ref/unicode/)

**VIEW**:

Django Views are one of the vital participants of [M**V**T Structure of Django](https://www.geeksforgeeks.org/django-project-mvt-structure/). As per Django Documentation, A view function is a Python function that takes a [Web request and returns a Web response](https://www.geeksforgeeks.org/django-request-and-response-cycle-httprequest-and-httpresponse-objects/). This **response** can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser can display.

Django views are part of the user interface — they usually render the HTML/CSS/Javascript in your Template files into what you see in your browser when you render a web page. (Note that if you’ve used other frameworks based on the [MVC (Model-View-Controller)](https://www.geeksforgeeks.org/mvc-design-pattern/), do not get confused between Django views and views in the MVC paradigm. Django views roughly correspond to controllers in MVC, and Django templates to views in MVC.)

**USER**

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The auth system consists of:

* Users
* Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
* Groups: A generic way of applying labels and permissions to more than one user.
* A configurable password hashing system
* Forms and view tools for logging in users, or restricting content
* A pluggable backend system

The authentication system in Django aims to be very generic and doesn’t provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

* Password strength checking
* Throttling of login attempts
* Authentication against third-parties (OAuth, for example)
* Object-level permissions

## Installation

Authentication support is bundled as a Django contrib module in **django.contrib.auth**. By default, the required configuration is already included in the **settings.py** generated by [**django-admin startproject**](https://docs.djangoproject.com/en/4.0/ref/django-admin/#django-admin-startproject), these consist of two items listed in your [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-INSTALLED_APPS) setting:

1. **'django.contrib.auth'** contains the core of the authentication framework, and its default models.
2. **'django.contrib.contenttypes'** is the Django [content type system](https://docs.djangoproject.com/en/4.0/ref/contrib/contenttypes/), which allows permissions to be associated with models you create.

and these items in your [**MIDDLEWARE**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-MIDDLEWARE) setting:

1. [**SessionMiddleware**](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.sessions.middleware.SessionMiddleware) manages [sessions](https://docs.djangoproject.com/en/4.0/topics/http/sessions/) across requests.
2. [**AuthenticationMiddleware**](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.auth.middleware.AuthenticationMiddleware) associates users with requests using sessions.

With these settings in place, running the command **manage.py migrate** creates the necessary database tables for auth related models and permissions for any models defined in your installed apps.

**URL**:

The **route** argument should be a string or [**gettext\_lazy()**](https://docs.djangoproject.com/en/4.0/ref/utils/#django.utils.translation.gettext_lazy) (see [Translating URL patterns](https://docs.djangoproject.com/en/4.0/topics/i18n/translation/#translating-urlpatterns)) that contains a URL pattern. The string may contain angle brackets (like **<username>** above) to capture part of the URL and send it as a keyword argument to the view. The angle brackets may include a converter specification (like the **int** part of **<int:section>**) which limits the characters matched and may also change the type of the variable passed to the view. For example, **<int:section>** matches a string of decimal digits and converts the value to an **int**. See [How Django processes a request](https://docs.djangoproject.com/en/4.0/topics/http/urls/#how-django-processes-a-request) for more details.The **view** argument is a view function or the result of [**as\_view()**](https://docs.djangoproject.com/en/4.0/ref/class-based-views/base/#django.views.generic.base.View.as_view) for class-based views. It can also be an [**django.urls.include()**](https://docs.djangoproject.com/en/4.0/ref/urls/#django.urls.include).

**Admin**

One of the most powerful parts of Django is the automatic admin interface. It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site. The admin’s recommended use is limited to an organization’s internal management tool. It’s not intended for building your entire front end around.

The admin has many hooks for customization, but beware of trying to use those hooks exclusively. If you need to provide a more process-centric interface that abstracts away the implementation details of database tables and fields, then it’s probably time to write your own views.In this document we discuss how to activate, use, and customize Django’s admin interface.The admin is enabled in the default project template used by [**startproject**](https://docs.djangoproject.com/en/4.0/ref/django-admin/#django-admin-startproject).

If you’re not using the default project template, here are the requirements:

1. Add**'django.contrib.admin'**anditsdependencies [**django.contrib.auth**](https://docs.djangoproject.com/en/4.0/topics/auth/#module-django.contrib.auth), [**django.contrib.contenttypes**](https://docs.djangoproject.com/en/4.0/ref/contrib/contenttypes/#module-django.contrib.contenttypes),[**django.contrib.messages**](https://docs.djangoproject.com/en/4.0/ref/contrib/messages/#module-django.contrib.messages),and [**django.contrib.sessions**](https://docs.djangoproject.com/en/4.0/topics/http/sessions/#module-django.contrib.sessions) - to your [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-INSTALLED_APPS) setting.
2. Configure a [**DjangoTemplates**](https://docs.djangoproject.com/en/4.0/topics/templates/#django.template.backends.django.DjangoTemplates) backend in your [**TEMPLATES**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-TEMPLATES) setting with **django.template.context\_processors.request**, **django.contrib.auth.context\_processors.auth**,and **django.contrib.messages.context\_processors.messages** in the **'context\_processors'** option of [**OPTIONS**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-TEMPLATES-OPTIONS).
3. Ifyou’vecustomizedthe [**MIDDLEWARE**](https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-MIDDLEWARE) setting, [**django.contrib.auth.middleware.AuthenticationMiddleware**](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.auth.middleware.AuthenticationMiddleware) and [**django.contrib.messages.middleware.MessageMiddleware**](https://docs.djangoproject.com/en/4.0/ref/middleware/#django.contrib.messages.middleware.MessageMiddleware) must be included.
4. [Hook the admin’s URLs into your URLconf](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/#hooking-adminsite-to-urlconf).After you’ve taken these steps, you’ll be able to use the admin site by visiting the URL you hooked it into (**/admin/**, by default).

If you need to create a user to login with, use the [**createsuperuser**](https://docs.djangoproject.com/en/4.0/ref/django-admin/#django-admin-createsuperuser) command. By default, logging in to the admin requires that the user has the [**is\_staff**](https://docs.djangoproject.com/en/4.0/ref/contrib/auth/#django.contrib.auth.models.User.is_staff) attribute set to **True**.Finally, determine which of your application’s models should be editable in the admin interface. For each of those models, register them with the admin as described in [**ModelAdmin**](https://docs.djangoproject.com/en/4.0/ref/contrib/admin/#django.contrib.admin.ModelAdmin).

Certainly! Let's dive into the details of the functions used in this project:

analyze\_frame(frame) function:

This function takes a frame (image) captured from the webcam as an input.

It utilizes the DeepFace.analyze() function from the deepface library to analyze the emotions in the given frame.

The DeepFace.analyze() function performs deep learning-based facial analysis and returns various facial attributes, including the detected emotions.

In this case, we are specifying 'emotion' as one of the actions to perform, which instructs the function to analyze the emotions in the given frame.

The result of the analysis, which includes information about the detected emotions, is returned from the function.

generate() function:

This is a generator function that continuously captures frames from the webcam, analyzes each frame using the analyze\_frame() function, and yields the frame and analysis result.

It starts by creating a video capture object (cv2.VideoCapture(0)) that represents the webcam.

It enters an infinite loop to continuously capture frames from the webcam.

For each captured frame:

It calls the analyze\_frame() function to analyze the emotions in the frame.

It uses cv2.imencode() to encode the frame as a JPEG image in memory.

The resulting image buffer is converted to bytes using the tobytes() method.

It yields the bytes of the frame along with the analysis result as a multipart response.

The generator function continues to capture and analyze frames as long as the loop is running.

detect\_emotion(request) view function:

This is the main view function that handles the HTTP request and generates the response.

It starts by defining the nested functions analyze\_frame(frame) and generate() as explained above.

It then calls the render() function to render the detect\_emotion.html template and passes the generate() function as a context variable named stream.

The stream variable is used in the template to generate the live video stream and display the detected emotions.

The rendered HTML template, along with the context variables, is returned as the response.

These functions work together to continuously capture frames from the webcam, analyze the emotions in each frame, and generate a real-time video stream with detected emotions in the Django web application.

Please note that the explanation provided here is a simplified overview of the functions' behavior. The actual implementation and intricacies of the deepface library and video processing are more complex.

**Python:**

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language](https://en.wikipedia.org/wiki/Programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python's design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its notable use of [significant whitespace](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) and [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is [dynamically typed](https://en.wikipedia.org/wiki/Dynamic_programming_language) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigms), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming),) object-oriented, and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). Python is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

Python was conceived in the late 1980s as a successor to the [ABC language](https://en.wikipedia.org/wiki/ABC_(programming_language)). Python 2.0, released in 2000, introduced features like [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) system capable of collecting [reference cycles](https://en.wikipedia.org/wiki/Reference_cycle). Python 3.0, released in 2008, was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility), and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)), only Python 3.5.xand later are supported.

Python [interpreters](https://en.wikipedia.org/wiki/Interpreter_(computing)) are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). A global community of programmers develops and maintains [CPython](https://en.wikipedia.org/wiki/CPython), an [open source](https://en.wikipedia.org/wiki/Open-source_software)[reference implementation](https://en.wikipedia.org/wiki/Reference_implementation). A [non-profit organization](https://en.wikipedia.org/wiki/Nonprofit_organization), the [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation), manages and directs resources for Python and CPython development.

**Python is used for:**

* web development (server-side),
* software development,
* mathematics,
* system scripting.

**Python do?:**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

**Why Python?:**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

**Python compared to other programming languages**

* Python was designed for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**4.7 Python installation procedure**:

**Windows Based**

It is highly unlikely that your Windows system shipped with Python already installed. Windows systems typically do not. Fortunately, installing does not involve much more than downloading the Python installer from the [python.org website](https://www.python.org/) and running it. Let’s take a look at how to install Python 3 on Windows:

### Step 1: Download the Python 3 Installer

1. Open a browser window and navigate to the [Download page for Windows](https://www.python.org/downloads/windows/) at [python.org](https://www.python.org/).
2. Underneath the heading at the top that says **Python Releases for Windows**, click on the link for the **Latest Python 3 Release - Python 3.x.x**. (As of this writing, the latest is Python 3.6.5.)
3. Scroll to the bottom and select either **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit. (See below.)

#### Sidebar: 32-bit or 64-bit Python?

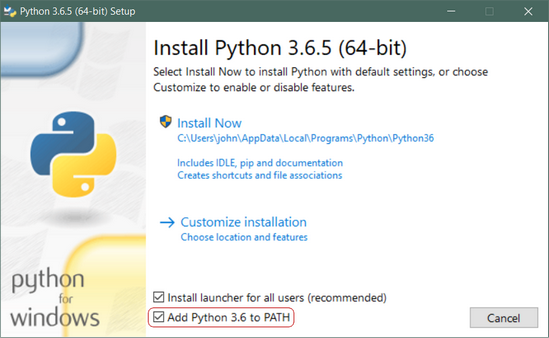
For Windows, you can choose either the 32-bit or 64-bit installer. Here’s what the difference between the two comes down to:

* If your system has a 32-bit processor, then you should choose the 32-bit installer.
* On a 64-bit system, either installer will actually work for most purposes. The 32-bit version will generally use less memory, but the 64-bit version performs better for applications with intensive computation.
* If you’re unsure which version to pick, go with the 64-bit version.

**Note:** Remember that if you get this choice “wrong” and would like to switch to another version of Python, you can just uninstall Python and then re-install it by downloading another installer from [python.org](https://python.org/).

### Step 2: Run the Installer

Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:

[](https://files.realpython.com/media/win-install-dialog.40e3ded144b0.png)

**Important:** You want to be sure to check the box that says **Add Python 3.x to PATH** as shown to ensure that the interpreter will be placed in your execution path.

Then just click **Install Now**. That should be all there is to it. A few minutes later you should have a working Python 3 installation on your system.

## Mac OS based

While current versions of macOS (previously known as “Mac OS X”) include a version of Python 2, it is likely out of date by a few months. Also, this tutorial series uses Python 3, so let’s get you upgraded to that.

The best way we found to install Python 3 on macOS is through the [Homebrew package manager](https://brew.sh/). This approach is also recommended by community guides like [The Hitchhiker’s Guide to Python](http://docs.python-guide.org/en/latest/starting/install3/osx/).

**4.8 Installation of packages:**

Syntax for installation of packages via cmd terminal using the basic

**Step:1- First check pip cmd**

First check pip cmd

If ok then

**Step:2- pip list**

Check the list of packages installed and then install required by following cmds

**Step:3- pip install package name**

The package name should as requirement

**4.9 USED LIBRARIES**

**1. OPEN CV :**

OpenCV is an open-source software library for computer vision and machine learning. The OpenCV full form is Open Source Computer Vision Library. It was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products. OpenCV, as a BSD-licensed software, makes it simple for companies to use and change the code. There are some predefined packages and libraries that make our life simple and OpenCV is one of them.

Gary Bradsky invented OpenCV in 1999 and soon the first release came in 2000. This library is based on optimised C / C++ and supports Java and Python along with C++ through an interface. The library has more than 2500 optimised algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art.Using OpenCV it becomes easy to do complex tasks such as identify and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high resolution image and many more.

Python is a user friendly language and easy to work with but this advantage comes with a cost of speed, as Python is slower to languages such as C or C++. So we extend Python with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. Doing this, the code is fast, as it is written in original C/C++ code (since it is the actual C++ code working in the background) and also, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

Let’s get started!

## ****What is Computer Vision?****

The term Computer Vision (CV) is used and heard very often in artificial intelligence (AI) and deep learning (DL) applications. The term essentially means giving a computer the ability to see the world as we humans do.

Computer Vision is a field of study which enables computers to replicate the human visual system. As already mentioned above, It’s a subset of artificial intelligence which collects information from digital images or videos and processes them to define the attributes. The entire process involves image acquiring, screening, analysing, identifying and extracting information. This extensive processing helps computers to understand any visual content and act on it accordingly.

Computer vision projects translate digital visual content into explicit descriptions to gather multi-dimensional data. This data is then turned into a computer-readable language to aid the decision-making process. The main objective of this branch of artificial intelligence is to teach machines to collect information from pixels.

## ****How does a computer read an image?****

How does a human mind apprehend an image? When you see the image below, what do you actually see and how do you say what is in the Image?

You  most probably look for different shapes and colours in the Image and that might help you decide that this is an image of a dog. But does a computer also see it in the same way? The answer is no.

A digital image is an image composed of picture elements, also known as pixels, each with finite, discrete quantities of numeric representation for its intensity or grey level. So the computer sees an image as numerical values of these pixels and in order to recognise a certain image, it has to recognise the patterns and regularities in this numerical data.

Here is a hypothetical example of how pixels form an image. The darker pixels are represented by a number closer to the zero and lighter pixels are represented by numbers approaching one. All other colours are represented by the numbers between 0 and 1.

But usually, you will find that for any colour image, there are 3 primary channels – Red, green and blue and the value of each channel varies from 0-255. In more simpler terms we can say that a digital image is actually formed by the combination of three basic colour channels  Red, green, and blue whereas for a grayscale image we have only one channel whose values also vary from 0-255.

## ****OpenCV installation****

There are many ways in which you can install OpenCV on your computer. Here are some:

### ****Install using Anaconda****

Anaconda is a conditional free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. You can download it from[here](https://www.anaconda.com/products/individual) and install it.

After successfully installing anaconda, just go to the anaconda prompt and use this command to install OpenCV:

conda install -c conda-forge opencv

After this command is successfully executed, OpenCV will be available on your computer.Now let us see some other ways to install OpenCV

### ****For Windows****

You can use pip to install OpenCV on windows. Pip is a de facto standard package-management system used to install and manage software packages written in Python and it usually comes in installed when you install Python. If you do not have Python installed, I would suggest download it from here. Use this command in the command prompt to install OpenCV:

pip install opencv-python

After installing it,do check if it is installed successfully.For that just go to the command prompt and type ‘python’ and hit enter.You should see some message like this:

If this is not the message you see, I suggest reinstalling python into your system. Next type import cv2 and if there is no error then it is installed successfully.

### ****For Mac****

You can use homebrew to install OpenCV as it makes it really easy and you just have to use this command for installing:

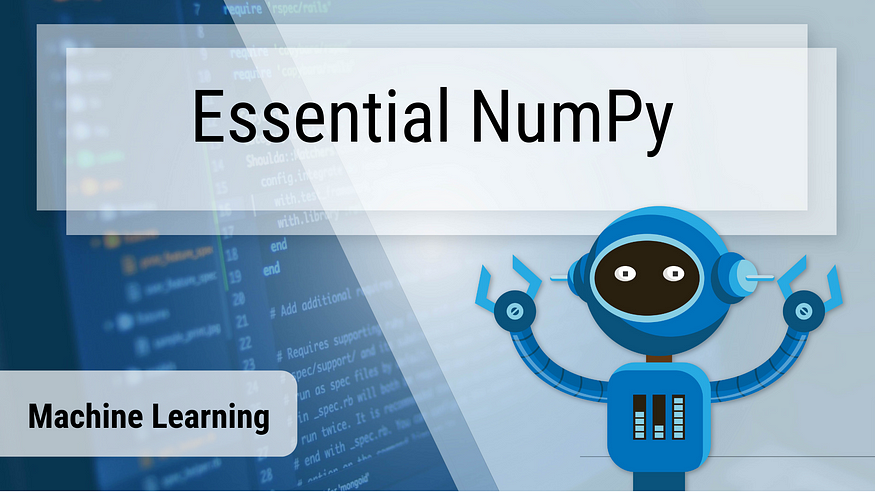
brew install opencv

Now that you have installed the OpenCV onto your system, let’s see how it works.

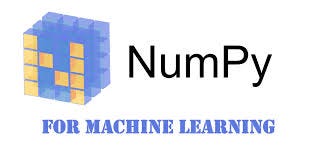
## ****Read & Save Images****

Now for OpenCV to work on any image, it must be able to read it. Here we will see how to read a file and save it after we are done with it.

**2.NUMPY**

In this Blog, I will be writing about all the basic stuff you need to know about numpy such as what is numpy, why we use numpy, why numpy is more useful than lists in python, getting started with numpy etc. 

# ****What is NumPy?****



NumPy, which stands for **Numerical Python**, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc.

# Why Use NumPy?

n Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

# Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

**3.DEEPFACE**

Deepface is a lightweight [face recognition](https://sefiks.com/2018/08/06/deep-face-recognition-with-keras/) and facial attribute analysis ([age](https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/), [gender](https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/), [emotion](https://sefiks.com/2018/01/01/facial-expression-recognition-with-keras/) and [race](https://sefiks.com/2019/11/11/race-and-ethnicity-prediction-in-keras/)) framework for python. It is a hybrid face recognition framework wrapping **state-of-the-art** models: [VGG-Face](https://sefiks.com/2018/08/06/deep-face-recognition-with-keras/), [Google FaceNet](https://sefiks.com/2018/09/03/face-recognition-with-facenet-in-keras/), [OpenFace](https://sefiks.com/2019/07/21/face-recognition-with-openface-in-keras/), [Facebook DeepFace](https://sefiks.com/2020/02/17/face-recognition-with-facebook-deepface-in-keras/), [DeepID](https://sefiks.com/2020/06/16/face-recognition-with-deepid-in-keras/), [ArcFace](https://sefiks.com/2020/12/14/deep-face-recognition-with-arcface-in-keras-and-python/), [Dlib](https://sefiks.com/2020/07/11/face-recognition-with-dlib-in-python/) and SFace.

Experiments show that human beings have 97.53% accuracy on facial recognition tasks whereas those models already reached and passed that accuracy level.

## Installation

The easiest way to install deepface is to download it from [PyPI](https://pypi.org/project/deepface/). It's going to install the library itself and its prerequisites as well.

$ pip install deepface

Secondly, DeepFace is also available at [Conda](https://anaconda.org/conda-forge/deepface). You can alternatively install the package via conda.

$ conda install -c conda-forge deepface

Thirdly, you can install deepface from its source code.

$ git clone https://github.com/serengil/deepface.git

$ cd deepface

$ pip install -e .

Then you will be able to import the library and use its functionalities.

from deepface import DeepFace

**Facial Recognition** - [Demo](https://youtu.be/WnUVYQP4h44)

A modern [**face recognition pipeline**](https://sefiks.com/2020/05/01/a-gentle-introduction-to-face-recognition-in-deep-learning/) consists of 5 common stages: [detect](https://sefiks.com/2020/08/25/deep-face-detection-with-opencv-in-python/), [align](https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/), [normalize](https://sefiks.com/2020/11/20/facial-landmarks-for-face-recognition-with-dlib/), [represent](https://sefiks.com/2018/08/06/deep-face-recognition-with-keras/) and [verify](https://sefiks.com/2020/05/22/fine-tuning-the-threshold-in-face-recognition/). While Deepface handles all these common stages in the background, you don’t need to acquire in-depth knowledge about all the processes behind it. You can just call its verification, find or analysis function with a single line of code.

**Face Verification** - [Demo](https://youtu.be/KRCvkNCOphE)

This function verifies face pairs as same person or different persons. It expects exact image paths as inputs. Passing numpy or base64 encoded images is also welcome. Then, it is going to return a dictionary and you should check just its verified key.

result = DeepFace.verify(img1\_path = "img1.jpg", img2\_path = "img2.jpg")



Verification function can also handle many faces in the face pairs. In this case, the most similar faces will be compared. 

**Face recognition** - [Demo](https://youtu.be/Hrjp-EStM_s)

an out-of-the-box find function to handle this action. It's going[Face recognition](https://sefiks.com/2020/05/25/large-scale-face-recognition-for-deep-learning/) requires applying face verification many times. Herein, deepface has to look for the identity of input image in the database path and it will return list of pandas data frame as output. Meanwhile, facial embeddings of the facial database are stored in a pickle file to be searched faster in next time. Result is going to be the size of faces appearing in the source image. Besides, target images in the database can have many faces as well.

dfs = DeepFace.find(img\_path = "img1.jpg", db\_path = "C:/workspace/my\_db")



**Embeddings**

Face recognition models basically represent facial images as multi-dimensional vectors. Sometimes, you need those embedding vectors directly. DeepFace comes with a dedicated representation function. Represent function returns a list of embeddings. Result is going to be the size of faces appearing in the image path.

embedding\_objs = DeepFace.represent(img\_path = "img.jpg")

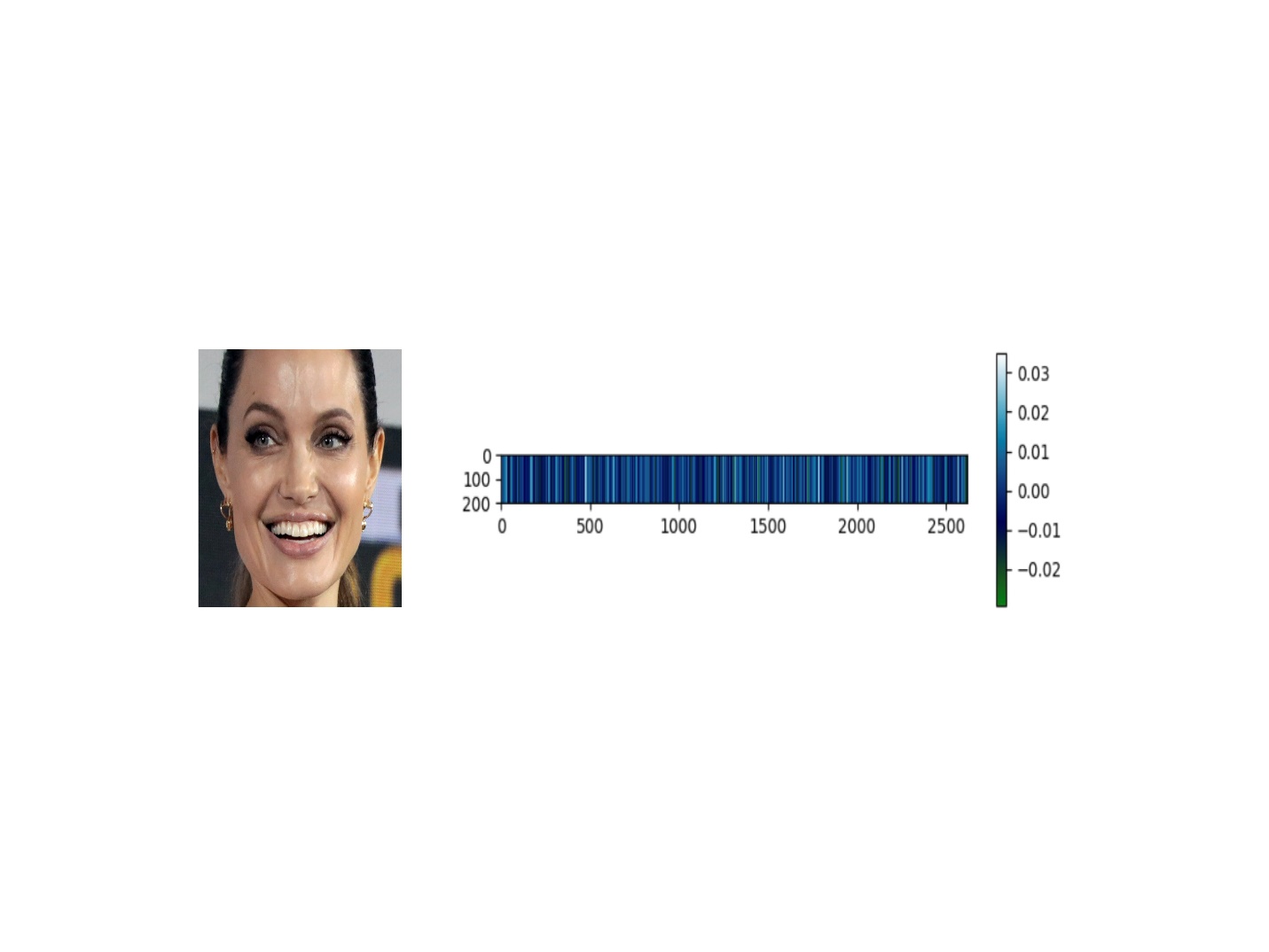
This function returns an array as embedding. The size of the embedding array would be different based on the model name. For instance, VGG-Face is the default model and it represents facial images as 2622 dimensional vectors.

embedding = embedding\_objs[0]["embedding"]

assert isinstance(embedding, list)

assert model\_name = "VGG-Face" and len(embedding) == 2622

Here, embedding is also [plotted](https://sefiks.com/2020/05/01/a-gentle-introduction-to-face-recognition-in-deep-learning/) with 2622 slots horizontally. Each slot is corresponding to a dimension value in the embedding vector and dimension value is explained in the colorbar on the right. Similar to 2D barcodes, vertical dimension stores no information in the illustration.



**Face recognition models** - [Demo](https://youtu.be/i_MOwvhbLdI)

Deepface is a **hybrid** face recognition package. It currently wraps many **state-of-the-art** face recognition models: [VGG-Face](https://sefiks.com/2018/08/06/deep-face-recognition-with-keras/) , [Google FaceNet](https://sefiks.com/2018/09/03/face-recognition-with-facenet-in-keras/), [OpenFace](https://sefiks.com/2019/07/21/face-recognition-with-openface-in-keras/), [Facebook DeepFace](https://sefiks.com/2020/02/17/face-recognition-with-facebook-deepface-in-keras/), [DeepID](https://sefiks.com/2020/06/16/face-recognition-with-deepid-in-keras/), [ArcFace](https://sefiks.com/2020/12/14/deep-face-recognition-with-arcface-in-keras-and-python/), [Dlib](https://sefiks.com/2020/07/11/face-recognition-with-dlib-in-python/) and SFace. The default configuration uses VGG-Face model.

models = [

"VGG-Face",

"Facenet",

"Facenet512",

"OpenFace",

"DeepFace",

"DeepID",

"ArcFace",

"Dlib",

"SFace",

]

#face verification

result = DeepFace.verify(img1\_path = "img1.jpg",

img2\_path = "img2.jpg",

model\_name = models[0]

)

#face recognition

dfs = DeepFace.find(img\_path = "img1.jpg",

db\_path = "C:/workspace/my\_db",

model\_name = models[1]

)

#embeddings

embedding\_objs = DeepFace.represent(img\_path = "img.jpg",

model\_name = models[2]

)



FaceNet, VGG-Face, ArcFace and Dlib are [overperforming](https://youtu.be/i_MOwvhbLdI) ones based on experiments. You can find out the scores of those models below on both [Labeled Faces in the Wild](https://sefiks.com/2020/08/27/labeled-faces-in-the-wild-for-face-recognition/) and YouTube Faces in the Wild data sets declared by its creators.

| **Model** | **LFW Score** | **YTF Score** |
| --- | --- | --- |
| Facenet512 | 99.65% | - |
| SFace | 99.60% | - |
| ArcFace | 99.41% | - |
| Dlib | 99.38 % | - |
| Facenet | 99.20% | - |
| VGG-Face | 98.78% | 97.40% |
| Human-beings | 97.53% | - |
| OpenFace | 93.80% | - |
| DeepID | - | 97.05% |

**Similarity**

Face recognition models are regular [convolutional neural networks](https://sefiks.com/2018/03/23/convolutional-autoencoder-clustering-images-with-neural-networks/) and they are responsible to represent faces as vectors. We expect that a face pair of same person should be [more similar](https://sefiks.com/2020/05/22/fine-tuning-the-threshold-in-face-recognition/) than a face pair of different persons.

Similarity could be calculated by different metrics such as [Cosine Similarity](https://sefiks.com/2018/08/13/cosine-similarity-in-machine-learning/), Euclidean Distance and L2 form. The default configuration uses cosine similarity.

metrics = ["cosine", "euclidean", "euclidean\_l2"]

#face verification

result = DeepFace.verify(img1\_path = "img1.jpg",

img2\_path = "img2.jpg",

distance\_metric = metrics[1]

)

#face recognition

dfs = DeepFace.find(img\_path = "img1.jpg",

db\_path = "C:/workspace/my\_db",

distance\_metric = metrics[2]

)

Euclidean L2 form [seems](https://youtu.be/i_MOwvhbLdI) to be more stable than cosine and regular Euclidean distance based on experiments.

**Facial Attribute Analysis** - [Demo](https://youtu.be/GT2UeN85BdA)

Deepface also comes with a strong facial attribute analysis module including [age](https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/), [gender](https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/), [facial expression](https://sefiks.com/2018/01/01/facial-expression-recognition-with-keras/) (including angry, fear, neutral, sad, disgust, happy and surprise) and [race](https://sefiks.com/2019/11/11/race-and-ethnicity-prediction-in-keras/) (including asian, white, middle eastern, indian, latino and black) predictions. Result is going to be the size of faces appearing in the source image.

objs = DeepFace.analyze(img\_path = "img4.jpg",

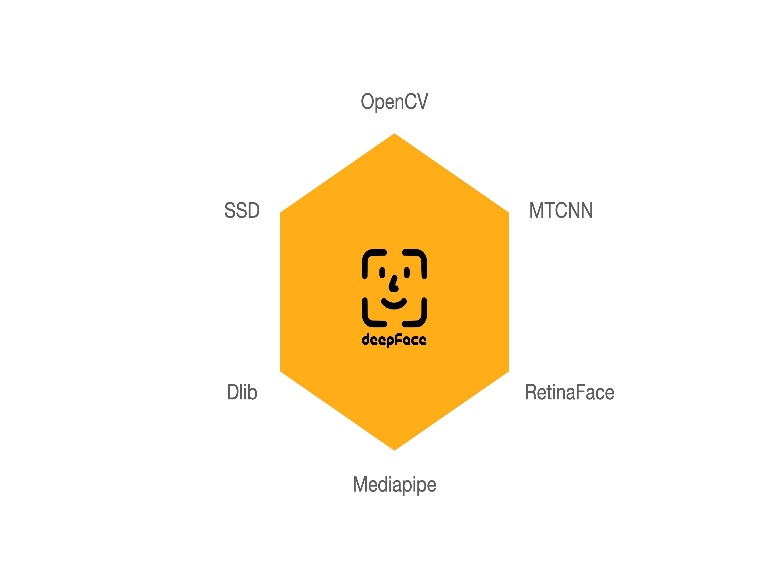
actions = ['age', 'gender', 'race', 'emotion']

)



Age model got ± 4.65 MAE; gender model got 97.44% accuracy, 96.29% precision and 95.05% recall as mentioned in its [tutorial](https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/).

**Face Detectors** - [Demo](https://youtu.be/GZ2p2hj2H5k)

Face detection and alignment are important early stages of a modern face recognition pipeline. Experiments show that just alignment increases the face recognition accuracy almost 1%. [OpenCV](https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/), [SSD](https://sefiks.com/2020/08/25/deep-face-detection-with-opencv-in-python/), [Dlib](https://sefiks.com/2020/07/11/face-recognition-with-dlib-in-python/), [MTCNN](https://sefiks.com/2020/09/09/deep-face-detection-with-mtcnn-in-python/), [RetinaFace](https://sefiks.com/2021/04/27/deep-face-detection-with-retinaface-in-python/) and [MediaPipe](https://sefiks.com/2022/01/14/deep-face-detection-with-mediapipe/) detectors are wrapped in deepface.

All deepface functions accept an optional detector backend input argument. You can switch among those detectors with this argument. OpenCV is the default detector.

backends = [

'opencv',

'ssd',

'dlib',

'mtcnn',

'retinaface',

'mediapipe'

]

#face verification

obj = DeepFace.verify(img1\_path = "img1.jpg",

img2\_path = "img2.jpg",

detector\_backend = backends[0]

)

#face recognition

dfs = DeepFace.find(img\_path = "img.jpg",

db\_path = "my\_db",

detector\_backend = backends[1]

)

#embeddings

embedding\_objs = DeepFace.represent(img\_path = "img.jpg",

detector\_backend = backends[2]

)

#facial analysis

demographies = DeepFace.analyze(img\_path = "img4.jpg",

detector\_backend = backends[3]

)

#face detection and alignment

face\_objs = DeepFace.extract\_faces(img\_path = "img.jpg",

target\_size = (224, 224),

detector\_backend = backends[4]

)

Face recognition models are actually CNN models and they expect standard sized inputs. So, resizing is required before representation. To avoid deformation, deepface adds black padding pixels according to the target size argument after detection and alignment.



[RetinaFace](https://sefiks.com/2021/04/27/deep-face-detection-with-retinaface-in-python/) and [MTCNN](https://sefiks.com/2020/09/09/deep-face-detection-with-mtcnn-in-python/) seem to overperform in detection and alignment stages but they are much slower. If the speed of your pipeline is more important, then you should use opencv or ssd. On the other hand, if you consider the accuracy, then you should use retinaface or mtcnn.

The performance of RetinaFace is very satisfactory even in the crowd as seen in the following illustration. Besides, it comes with an incredible facial landmark detection performance. Highlighted red points show some facial landmarks such as eyes, nose and mouth. That's why, alignment score of RetinaFace is high as well.

  
The Yellow Angels - Fenerbahce Women's Volleyball Team

You can find out more about RetinaFace on this [repo](https://github.com/serengil/retinaface).

**Real Time Analysis** - [Demo](https://youtu.be/-c9sSJcx6wI)

You can run deepface for real time videos as well. Stream function will access your webcam and apply both face recognition and facial attribute analysis. The function starts to analyze a frame if it can focus a face sequentially 5 frames. Then, it shows results 5 seconds.

DeepFace.stream(db\_path = "C:/User/Sefik/Desktop/database")

Even though face recognition is based on one-shot learning, you can use multiple face pictures of a person as well. You should rearrange your directory structure as illustrated below.

**SOURCE CODE :**

HOME.HTML

<html>

  <head>

    <title>Video Live Stream</title>

    <style>

      #main-title{

        color: rgb(6, 192, 124);

      }

    </style>

  </head>

  <body>

    <h1 id="main-title">Video Live Stream</h1>

    <img width="500px" height="500px" src="{% url 'video\_feed' %}" >

  </body>

</html>

APPS.PY

from django.apps import AppConfig

class StreamappConfig(AppConfig):

    name = 'streamapp'

CAMERA.PY

from tensorflow.keras.models import load\_model

from imutils.video import VideoStream

from deepface import DeepFace

import imutils

import cv2

import os

import urllib.request

import numpy as np

from django.conf import settings

face\_detection\_videocam = cv2.CascadeClassifier(os.path.join(

    settings.BASE\_DIR, 'opencv\_haarcascade\_data/haarcascade\_frontalface\_default.xml'))

face\_detection\_webcam = cv2.CascadeClassifier(os.path.join(

    settings.BASE\_DIR, 'opencv\_haarcascade\_data/haarcascade\_frontalface\_default.xml'))

# load our serialized face detector model from disk

prototxtPath = os.path.sep.join(

    [settings.BASE\_DIR, "face\_detector/deploy.prototxt"])

weightsPath = os.path.sep.join(

    [settings.BASE\_DIR, "face\_detector/res10\_300x300\_ssd\_iter\_140000.caffemodel"])

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

maskNet = load\_model(os.path.join(

    settings.BASE\_DIR, 'face\_detector/mask\_detector.model'))

class VideoCamera(object):

    def \_\_init\_\_(self):

        self.video = cv2.VideoCapture(0)

    def \_\_del\_\_(self):

        self.video.release()

    def get\_frame(self):

        success, image = self.video.read()

        # We are using Motion JPEG, but OpenCV defaults to capture raw images,

        # so we must encode it into JPEG in order to correctly display the

        # video stream.

        gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

        faces\_detected = face\_detection\_videocam.detectMultiScale(

            gray, scaleFactor=1.3, minNeighbors=5)

        for (x, y, w, h) in faces\_detected:

            cv2.rectangle(image, pt1=(x, y), pt2=(x + w, y + h),

                          color=(255, 0, 0), thickness=2)

        result = DeepFace.analyze(

            image, actions=['emotion'], enforce\_detection=False)

        gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

        font = cv2. FONT\_HERSHEY\_SIMPLEX

         # Use putText() method for

        # inserting text on video

        cv2.putText(image, result[0]['dominant\_emotion'],

                    (50, 50), font, 3, (0, 0, 255), 2, cv2.LINE\_4)

        # frame\_flip = cv2.flip(image, 1)

        ret, jpeg = (cv2.imencode('.jpg', image))

        jpeg = np.array(jpeg)

        return jpeg.tobytes()

URLS.py

from django.urls import path, include

from streamapp import views

urlpatterns = [

    path('', views.index, name='index'),

    path('video\_feed', views.video\_feed, name='video\_feed'),

VIEWS.py

from django.shortcuts import render

from django.http.response import StreamingHttpResponse

from streamapp.camera import VideoCamera

# Create your views here.

def index(request):

    return render(request, 'streamapp/home.html')

def gen(camera):

    while True:

        frame = camera.get\_frame()

        yield (b'--frame\r\n'

               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

def video\_feed(request):

    return StreamingHttpResponse(gen(VideoCamera()),

                                 content\_type='multipart/x-mixed-replace; boundary=frame')

ASGI.PY

"""

ASGI config for video\_stream project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/

"""

import os

from django.core.asgi import get\_asgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'video\_stream.settings')

application = get\_asgi\_application()

SETTINGS.PY

"""

Django settings for video\_stream project.

Generated by 'django-admin startproject' using Django 3.0.5.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/3.0/ref/settings/

"""

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = '1lxk3a6m+6c-@mr5i3k6(4sv73-f@s=o\*j2@#^)e^^i-px4!kj'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'streamapp'

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'video\_stream.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'video\_stream.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

}

# Password validation

# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_URL = '/static/'

URLS.pY

"""video\_stream URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

    https://docs.djangoproject.com/en/3.0/topics/http/urls/

Examples:

Function views

    1. Add an import:  from my\_app import views

    2. Add a URL to urlpatterns:  path('', views.home, name='home')

Class-based views

    1. Add an import:  from other\_app.views import Home

    2. Add a URL to urlpatterns:  path('', Home.as\_view(), name='home')

Including another URLconf

    1. Import the include() function: from django.urls import include, path

    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('streamapp.urls'))

]

WSGI.PY

"""

WSGI config for video\_stream project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/

"""

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'video\_stream.settings')

application = get\_wsgi\_application()

MANAGE,PY

#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

    os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'video\_stream.settings')

    try:

        from django.core.management import execute\_from\_command\_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

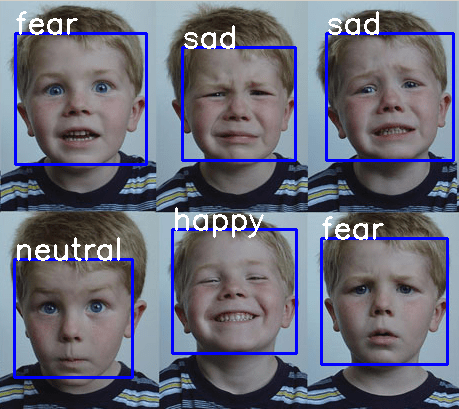
            "forget to activate a virtual environment?"

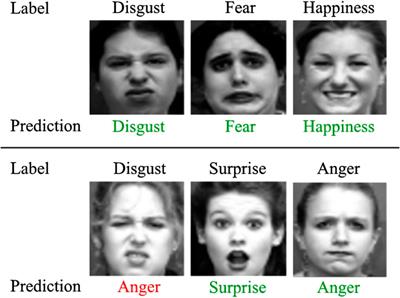
        ) from exc

    execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

    main()





**TESTING-UNIT AND INTEGRATION**

**1. Introduction:**

Testing is an essential phase in the development process of the Django-based Emotion Detection project using the DeepFace library. It ensures that the application functions as intended, detects emotions accurately, and provides a reliable user experience. Testing is performed at different levels, including unit testing and integration testing, to validate the individual components and their interactions within the system.

**2. Objectives of Testing:**

The objectives of testing in the Django-based Emotion Detection project are as follows:

a) Verify the accuracy of emotion detection algorithms implemented using the DeepFace library.

b) Ensure the proper functioning of each component and module within the application.

c) Validate the integration and interactions between different modules and external libraries.

d) Identify and resolve any defects, bugs, or errors in the application.

e) Assess the application's performance, security, and usability aspects.

f) Provide confidence in the reliability and quality of the application.

**3. Unit Testing:**

Unit testing focuses on testing individual components or units of code to ensure they function correctly. In the Django-based Emotion Detection project, unit testing can involve:

a) Testing the DeepFace library's emotion detection algorithms with sample input images and verifying the accuracy of the predicted emotions.

b) Testing Django models, views, and templates to ensure their functionality and data integrity.

c) Writing test cases to cover edge cases, boundary conditions, and expected input/output scenarios.

d) Using testing frameworks, such as Django's built-in testing framework or third-party libraries like PyTest, to automate and streamline the unit testing process.

**4. Integration Testing:**

Integration testing evaluates the interactions between different modules and components within the application. In the Django-based Emotion Detection project, integration testing can include:

a) Testing the integration of the DeepFace library with Django, ensuring proper communication and data flow between the two.

b) Verifying the integration of user registration and authentication with the emotion detection functionality.

c) Testing the interaction between the image upload functionality and the emotion detection algorithms.

d) Validating the integration of the result display module with the emotion detection module to ensure accurate presentation of predicted emotions.

**5. Test Coverage and Test Automation:**

To ensure thorough testing, it is important to achieve high test coverage, which means that a significant portion of the application's code is exercised by the tests. This can be achieved through a combination of unit tests and integration tests that cover different functionalities and scenarios.

Test automation plays a crucial role in improving the efficiency and effectiveness of testing. It allows for the automation of repetitive and time-consuming test cases, reducing the overall testing effort. Automation frameworks like Selenium or Django's built-in testing tools can be utilized to automate test cases.

**6. Performance Testing:**

Performance testing evaluates the application's responsiveness and stability under various workloads. In the Django-based Emotion Detection project, performance testing can involve:

a) Simulating multiple concurrent user requests to assess the application's ability to handle peak loads.

b) Measuring response times for image upload, emotion detection, and result display functionalities.

c) Analyzing resource utilization, such as CPU and memory usage, during peak loads.

d) Identifying and optimizing any performance bottlenecks to ensure smooth and efficient operation of the application.

**7. Security Testing:**

Security testing is important to ensure the application's protection against potential vulnerabilities and attacks. In the Django-based Emotion Detection project, security testing can include:

a) Validating input sanitization and handling to prevent injection attacks.

b) Assessing user authentication and authorization mechanisms to ensure secure access control.

c) Testing for secure transmission of data using encryption protocols (e.g., HTTPS).

d) Conducting vulnerability scanning and penetration testing to identify and address potential security weaknesses.

**CONCLUSION**

In conclusion, the project "Django-based Emotion Detection using DeepFace Library" aims to develop an application that allows users to upload facial images for emotion detection using the DeepFace library. The objectives of the project were to provide accurate emotion detection, user-friendly interfaces, image analysis history, and an admin dashboard for system management.

Throughout the project, various aspects were considered to ensure its success. A feasibility study was conducted, which examined technical, economic, operational, legal, ethical, and schedule feasibility. The study determined that the project is viable and aligns with the organization's goals.

Testing played a crucial role in ensuring the quality and reliability of the application. Unit testing was performed to verify the functionality of individual components, including the DeepFace library, Django models, views, and templates. Integration testing was conducted to validate the interactions between different modules and the integration of the DeepFace library with Django.

Test coverage and automation were utilized to achieve thorough testing and improve testing efficiency. Performance testing was performed to assess the application's responsiveness and stability under various workloads, while security testing ensured the application's protection against potential vulnerabilities and attacks.

In conclusion, the Django-based Emotion Detection project using the DeepFace library is a feasible and promising initiative. By accurately detecting emotions, providing user-friendly interfaces, maintaining image analysis history, and offering an admin dashboard, the project aims to meet user needs and provide a valuable tool for emotion analysis. Continued monitoring, user feedback, and updates will be essential to enhance the application's long-term viability and relevance in the field of emotion detection.

**REFRENCES**

1. DeepFace: A library for facial analysis, recognition, and emotion detection. GitHub repository. Available at: [https://github.com/serengil/deepface](https://github.com/serengil/deepface)

2. Django: The web framework for perfectionists with deadlines. Official documentation. Available at: [https://www.djangoproject.com/](https://www.djangoproject.com/)

3. Python Programming Language. Official website. Available at: [https://www.python.org/](https://www.python.org/)

4. OpenCV: Open Source Computer Vision Library. Official documentation. Available at: [https://docs.opencv.org/](https://docs.opencv.org/)

5. MySQL Database Management System. Official website. Available at: [https://www.mysql.com/](https://www.mysql.com/)

6. Django Documentation. Official documentation. Available at: [https://docs.djangoproject.com/](https://docs.djangoproject.com/)

7. The Django Testing Tools Documentation. Official documentation. Available at: [https://docs.djangoproject.com/en/3.2/topics/testing/](https://docs.djangoproject.com/en/3.2/topics/testing/)

8. Selenium: Web Browser Automation. Official website. Available at: [https://www.selenium.dev/](https://www.selenium.dev/)

9. PyTest: Python Testing Framework. Official documentation. Available at: [https://docs.pytest.org/](https://docs.pytest.org/)

10. HTTPS: Hypertext Transfer Protocol Secure. Wikipedia. Available at: [https://en.wikipedia.org/wiki/HTTPS](https://en.wikipedia.org/wiki/HTTPS)

11. OWASP: Open Web Application Security Project. Official website. Available at: [https://owasp.org/](https://owasp.org/)

12. Scrum: Agile Software Development Framework. Official website. Available at: [https://www.scrum.org/](https://www.scrum.org/)

13. IEEE Standard for Software and System Test Documentation (IEEE 829). Available at: [https://standards.ieee.org/standard/829-2008.html](<https://standards.ieee.org/standard/829-2008.html>)

14. <https://medium.com/analytics-vidhya/introduction-to-numpy-82321478e788#:~:text=NumPy%2C%20which%20stands%20for%20Numerical,as%20its%20architecture%20and%20environment>.

15.

Note: Additional references specific to deep learning, image analysis, and emotion detection algorithms should be consulted during the project development, as per the specific requirements and implementation choices.