EAI 6010 Module 5 : Facial Landmark Detection Microservice: A Deployment Case Study

|  |  |  |  |
| --- | --- | --- | --- |
| **Course** | EAI6010 | **Module** | 5 |
| **Submission By** | Arjun Shrivatsan | | |
| **Included** | Assignment Report  Link to Deployed Service  Source Code Link | | |

Overview

In this project, I have developed and deployed a microservice that performs facial landmark detection using MediaPipe, OpenCV, and Flask. The service processes a user-uploaded image and returns a detailed JSON response of facial landmarks and derived facial measurements like forehead width, eye gap, and chin width. Facial landmark detection has wide-ranging applications in healthcare, augmented reality, and safety systems. The idea behind building this service stems from a real-world requirement in helmet manufacturing, where accurate facial dimensions can improve helmet fit and thereby enhance safety.

## Use Case

I have chosen as part of my work that I have done in sports safety. The primary use case is for personalized helmet fitting, safety equipment design, or biometric analysis. By retrieving accurate facial measurements from a 2D image, manufacturers or developers can build safer and more customized wearables without requiring expensive 3D scanning hardware.

## Why this Use Case

By using a lightweight and web-deployable model like MediaPipe's FaceMesh, I can offer scalable and accurate detection without intensive server-side computation. The use case also presents an intersection of AI, healthcare, and manufacturing—domains where I have practical experience. This deployment serves as a starting point to build future pipelines that include photogrammetry, 3D modeling, and reinforcement learning-based fit optimization.

This microservice can be integrated into systems that require accurate facial geometry measurements, such as custom safety gear fitting, biometric systems, and facial analysis for behavioral studies. I chose this use case as I have previously worked with helmet manufacturers and observed the gap in custom-fit solutions. Real-time and mobile-based facial detection solutions can reduce dependency on expensive scanning equipment.

# Approach and Technology Stack

The core approach was to create a RESTful API using Flask that accepts an image, processes it using MediaPipe’s FaceMesh solution, and returns landmark coordinates and derived facial dimensions in JSON format. MediaPipe is developed by Google Research and is well-documented and efficient for real-time applications (Lugaresi et al., 2019). The service is deployed on Render.com, ensuring public accessibility with minimal server maintenance.

### ****Tech Stack****

| **Component** | **Details** |
| --- | --- |
| Programming Language | Python 3.10 |
| Web Framework | Flask |
| Face Detection | MediaPipe (FaceMesh) |
| Image Processing | OpenCV |
| Image Transformation | PIL (Python Imaging Library) |
| Deployment Platform | [Render.com](https://render.com) – for ease of use and Docker support |
| Containerization | Docker |
| API Testing Tool | cURL |

# Dataset

The dataset used for testing and demonstration in this microservice is derived from the **CelebA (CelebFaces Attributes Dataset)**, a widely used large-scale face attributes dataset with more than 200,000 celebrity images, each annotated with 40 attribute labels and 5 landmark locations. Although our model (MediaPipe FaceMesh) does not use the attribute annotations, the diversity and facial variation in CelebA make it an excellent dataset for testing face landmark detection tasks. It covers a wide range of pose variations, facial expressions, occlusions, and lighting conditions, providing robustness in evaluation.

The dataset used for testing the service includes high-resolution face images stored locally. These are personal test datasets and not publicly available. For actual deployment, any real face image with frontal orientation can be used to test the API.

We use a few sample images from the CelebA dataset as test inputs to validate the endpoint's ability to detect facial landmarks and measure facial features effectively.

# Implementation

Development began with local development using Jupyter Notebooks and MediaPipe's visualization tools to test the FaceMesh model. I then built the Flask microservice and tested it locally with different image sizes and angles to ensure robustness. Unit tests were written to validate JSON responses, and integration testing was done using `curl` commands. The application was then Dockerized for consistent deployment across environments.

### ****Output****

The output from the /face-coordinates endpoint is returned in **JSON** format and includes two major components:

1. **Landmarks** – A list of facial keypoints with their id, and x, y pixel positions. These are generated by MediaPipe's FaceMesh model, which detects 468 facial landmarks per face. Each coordinate pair corresponds to a unique anatomical facial feature.
2. **Measurements** – Three key facial metrics computed using landmark positions:
   * forehead\_width\_px
   * eye\_gap\_px
   * chin\_width\_px

These values are helpful for various applications, including custom safety helmet design, avatar modeling, and biometric validation. The output is lightweight and easy to parse, making it ideal for further processing or integration with frontend visualization tools.

Here is a brief example snippet of the output:

json

Copy code

{

"landmarks": [{"id": 0, "x": 91, "y": 147}, ...],

"measurements": {

"forehead\_width\_px": 10,

"eye\_gap\_px": 21,

"chin\_width\_px": 8

}

}

This structured response helps users, including the instructor, clearly verify that the model is functioning as intended and delivering valid facial geometry data.

### ****Implementation Details****

| **Step / Component** | **Description** |
| --- | --- |
| Endpoint | /face-coordinates (POST method) |
| Input | Image uploaded as multipart/form-data |
| Image Handling | Converted to NumPy array and BGR format using OpenCV |
| Landmark Detection | MediaPipe FaceMesh model |
| Landmark Mapping | Facial feature indices mapped (e.g., eyes, forehead, chin) |
| Measurement Calculation | Pixel distance between key facial points |
| Output | JSON with: |
|  | – landmarks: list of facial coordinates (id, x, y) |
|  | – measurements: forehead width, eye gap, and chin width (in pixels) |
|  |  |

## Deployment

The deployment used Render.com with a connected GitHub repository. A Dockerfile was created to install dependencies from `requirements.txt` and expose the Flask server. After successful builds and logs verification, I tested the public endpoint using image uploads. The Render deployment remains live, and the endpoint can be invoked reliably as long as the Render service does not auto-suspend.

A screenshot of a computer

AI-generated content may be incorrect.

## Annexure B: Key Information

|  |  |
| --- | --- |
| Item | Details |
| Service URL | https://facial-coordinates.onrender.com |
| Git URL | [Private/Local] |
| Dataset Reference | Custom local test images |
| MediaPipe Reference | Lugaresi et al., 2019 - https://arxiv.org/abs/1906.08172 |
| Tech Stack Required | Python 3.10, Flask, OpenCV, MediaPipe, Docker |
| Credentials Required | None for API access |

## Annexure C: Installation Instructions

1. Clone the repository.  
2. Install dependencies: `pip install -r requirements.txt`  
3. Run using Flask: `python app.py`  
4. To deploy, use Docker and push to Render.com.  
5. Test API with `curl` or Postman.

## Additional Notes

Keep the Render app active by making a request at intervals to prevent it from sleeping. If it goes to sleep, the first request might take a few seconds to wake it up. Ensure that large image files are resized for faster processing.

### ****Running Instructions (Using CURL)****

To invoke the facial landmark detection service, a curl command can be used to post an image to the /face-coordinates endpoint. The command follows the format:

**bash**

**Copy code**

**curl -X POST -F "image=@/path/to/image.jpg"** [**https://facial-coordinates.onrender.com/face-coordinates**](https://facial-coordinates.onrender.com/face-coordinates)

**e.g Shown image below :**

**A blue and black pattern

AI-generated content may be incorrect.**

This allows easy testing from the command line, especially for developers or instructors evaluating the microservice. The key part of the command is -F "image=@/path/to/image.jpg", which uploads the image as form data under the key image.

This method is simple, doesn't require a frontend interface, and works well with most Linux, macOS, and Windows environments that have cURL installed. No additional headers or tokens are required, making the service frictionless to test.

## Conclusion

This project demonstrates how AI can be meaningfully applied through a lightweight and accessible microservice. By leveraging MediaPipe for facial landmark detection (Lugaresi et al., 2019; Google Developers, 2023), we enabled efficient face geometry analysis without heavy computation. The CelebA dataset (Liu et al., 2015) served as a robust testbed, providing facial diversity and alignment quality. Deployment via Render.com ensured public accessibility and reproducibility (Render, 2024). The use of curl allowed straightforward command-line testing and integration. This implementation proves the potential of real-time facial geometry analysis in applications such as custom-fit helmet design and safety gear innovation.

## References

Google Developers. (2023). MediaPipe face mesh. <https://developers.google.com/mediapipe/solutions/vision/face_mesh>

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 3730–3738).

Lugaresi, C., Tang, J., Nash, H., McGuire, M., Tseng, Y., Chang, C., ... & Kwatra, V. (2019). MediaPipe: A framework for building perception pipelines (arXiv:1906.08172). arXiv. <https://arxiv.org/abs/1906.08172>

OpenCV Team. (2023). OpenCV: Open source computer vision library. <https://opencv.org/>

Render. (2024). Render.com documentation: Web services. <https://render.com/docs/web-services>

Zafeiriou, S., Trigeorgis, G., Chrysos, G. G., Deng, J., & Tzimiropoulos, G. (2017). The Menpo facial landmark localisation challenge: A step towards the solution. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 2116–2125). <https://openaccess.thecvf.com/content_cvpr_2017_workshops/w5/html/Zafeiriou_The_Menpo_Facial_CVPR_2017_paper.html>