FACE LANDMARK PREDICTION

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE420 - COMPUTER VISION

Submitted by

PRITHIVI RAJ C
(125163005)

November 2024

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**FACE LANDMARK DETECTION**" submitted as a requirement for the course, **CSE420** for M.Tech. **ARTIFICIAL INTELLIGENCE AND ROBOTICS** programme, is a bonafide record of the work done by **PRITHIVI RAJ C (125163005)** during the academic year 2024-2025, in the SCHOOL of ELECTRICAL AND ELECTRONICS Engineering, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : N.R.Raajan,ASSOCIATE PROFESSOR,ECE,SEEE

Date :

Project Viva-voce held on _____

Examiner 1 **Examiner 2**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR – 613 401

Declaration

I declare that the report titled "**FACE LANDMARK PREDICTION**" submitted by me is an original work done by me under the guidance of **N.R.Raajan, ASSOCIATE PROFESSOR, ECE, SEEE, SASTRA Deemed to be University** during the odd semester of the final academic year 2024-2025, in the **School of ELECTRICAL AND ELECTRONICS ENGINEERING** . The work is original and wherever I have used materials from other sources, I have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate:

Name of the candidate: PRITHIVI RAJ C

Date :

Acknowledgements

I am grateful for what I am and have. My thanks giving are perpetual. Though only my name appears on the cover of this dissertation, many great people have backed me in my research career. I owe my gratitude to all those people who have made this dissertation possible.

My grateful and sincere thanks to Prof. R. Sethuraman, Chairman, SASTRA Deemed to be University and Dr. S. Vaidhyasubramaniam, Vice-Chancellor of SASTRA Deemed to be University, for providing me the opportunity to carry out my doctoral studies in this prestigious university with the needed infrastructure and the research environment.

I owe my sincere thanks to Dr. S. Swaminathan, Dean – Planning & Development for his motivation towards research. I would like to extend my sincere thanks for the support given by Dr. R. Chandramouli, Registrar, SASTRA Deemed to be University.

I express my sincere and grateful thanks to Dr. K. Sridhar, Dean, Research,
Dr. K.Thenmozhi, Dean, School of Electrical & Electronics Engineering, Dr. V. Muthubalan, Associate Dean – Research, School of Electrical & Electronics Engineering.

I would like to thank my project advisor Dr.N.R.Raajan,ASSOCIATE PROFESSOR,ECE, SEEE, SASTRA Deemed to be University, for their untiring encouragement, support, and patience during my course period. I feel highly privileged and fortunate to have an opportunity to carry out my project work under their supervision. Their sage advice, insightful criticisms, and patient encouragement aided the writing of this thesis in innumerable ways.

I extend my gratefulness to one and all who helped me directly or indirectly in the successful completion of this project work.

I express my emotional thanks to all my friends for their inspirational words, criticism, advice and suggestions which I received from day one of my project work. I am indebted to God for having gifted me the much-needed strength, spirit and willpower to pursue my project endeavor.

Thank you

PRITHIVI RAJ C

Contents

List of Figures

Abstract

This project presents a novel approach to continuous facial landmark prediction using Google's pretrained Mediapipe model without retraining. The primary objective is to enable real-time prediction of user-specified facial landmarks from video input, enhancing efficiency in applications such as gesture recognition and human-computer interaction. The methodology involves processing a video file where users define the number of landmarks to predict, resulting in a video output that overlays the predicted landmarks onto the original frames.

Results demonstrate the effectiveness of this method in accurately predicting and visualizing facial landmarks across various video inputs. This research has significant implications for fields like augmented reality and interactive media, showcasing the potential for improved user engagement through precise facial analysis without extensive computational overhead. Future work may focus on further optimizations and exploring applications in dynamic environments.

CHAPTER 1

INTRODUCTION

1.1 Facial Landmark

Facial landmarks are precise points on a face that indicate important features, like the eyes, nose, mouth, and jawline. Their detection is crucial for various technologies involving faces, such as face recognition, animated avatars, emotion detection, and augmented reality filters. By identifying these points, computers can comprehend and monitor the structure, position, and changes in a face.

Deep learning has significantly enhanced the detection of facial landmarks in recent years, resulting in faster and more accurate results. This progress has driven advancements in interactive applications and entertainment, where real-time tracking of facial expressions is vital. Tools like Google's MediaPipe have simplified the process for developers to accurately detect hundreds of facial landmarks, even on mobile devices, through a pre-trained model that tracks faces

1.1.1 Sparse Landmark Prediction:

Sparse landmark prediction involves detecting a small number of key points on the face, such as the corners of the eyes, nose, mouth, and chin. These predictions typically include between 5 and 68 points strategically placed to capture the essential facial structure. This type of prediction is commonly used in applications like face alignment, facial expression recognition, and low-resolution face tracking, where fast processing is required without using a lot of computational resources.

Sparse prediction is efficient because it simplifies the model, making it quicker and simpler to train, while still capturing the most important facial features. By concentrating only on key landmarks, sparse prediction enables effective tracking of faces in real time, making it suitable for applications on mobile devices and embedded systems with limited processing power.

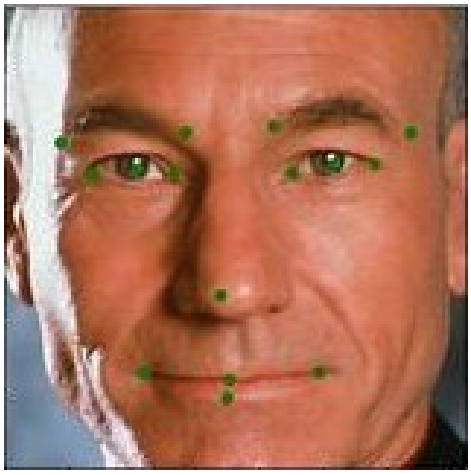

Figure 1.1: Sparse Landmark Prediction. Image courtesy of [Name/Source].

Sparse landmark prediction is typically done using a standardized layout, like the common 68-point layout utilized in face recognition systems. While not as detailed as dense prediction, sparse landmarks are often sufficiently accurate for many purposes and can serve as a foundation for alignment in more advanced applications such as 3D modeling or augmented reality. Due to its simplicity and speed, sparse landmark prediction is a practical option for systems needing a balance of efficiency and effectiveness in facial analysis.

1.1.2 Dense Landmark Prediction:

Dense landmark prediction involves the detection of numerous facial landmarks, often exceeding hundreds, distributed across the entire face. Unlike sparse prediction, which focuses on key facial points, dense prediction captures intricate details of the face's shape and structure, including contours, wrinkles, and subtle skin texture. This method is especially beneficial for applications requiring high precision and depth, such as 3D facial reconstruction, realistic facial animation, and augmented reality, where a more comprehensive facial map improves accuracy.

In dense prediction, landmarks are not restricted to surface points but may also encompass internal features like the jawline or cheekbones, providing a nearly complete representation of the face's 3D shape. This enables the model to monitor complex facial movements, making it well-suited for top

The finer level of detail required for dense landmark prediction results in higher computational requirements, necessitating more processing power and storage capacity. Nonetheless, advancements in deep learning techniques and dedicated platforms such as Google's MediaPipe have enabled the achievement of real-time dense prediction even on mobile devices. Despite the increased computational load, dense landmark prediction offers a superior level of intricacy, proving essential for applications that demand realistic accuracy in face

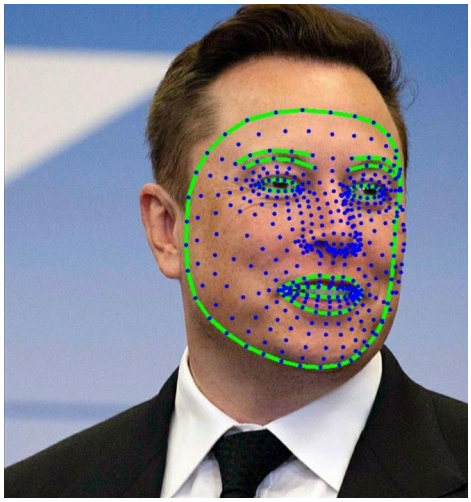

Figure 1.2: Dense Landmark Prediction
Credits: datamagic.webflow.io .

tracking and visualization.

The prediction of dense landmarks also offers increased versatility for adaptive uses, permitting ongoing customization and tracking of distinct facial characteristics, such as freckles, moles, or even temporary features like makeup or tattoos. This flexibility is particularly beneficial in personalized or medical contexts, where it is important to monitor specific traits over time. Additionally, dense landmarks allow for greater precision in animation, as each point can signify subtle expressions and delicate movements, capturing the individual's complete range of emotions and gestures.

In augmented and virtual reality, dense landmark data can be accurately applied to digital avatars or virtual faces, providing a more engaging and realistic experience for users. The capacity to precisely reproduce subtle movements in facial muscles or variations in skin texture introduces an enhanced degree of realism to applications necessitating engagement with digital characters, ranging from video games to social media filters. Although it presents challenges in terms of processing power and resource demands, dense landmark prediction remains a catalyst for advancements in areas that depend on realistic and responsive facial tracking, thereby expanding opportunities for more advanced and interactive digital experiences.

CHAPTER 2

LITERATURE REVIEW

2.1 High-resolution representations for facial landmark detection

NAME OF THE AUTHOR: Jingdong Wang and Ke Sun and Tianheng Cheng and Borui Jiang and Chaorui Deng and Yang Zhao and Dong Liu and Yadong Mu and Mingkui Tan and Xinggang Wang and Wenyu Liu and Bin Xiao
NAME OF THE JOURNAL: IEEE
YEAR OF PUBLISH: 2020
INFERENCE: Discusses high-resolution frameworks for real-time face landmark detection.

2.2 Continuous Landmark Detection with 3D Querie

NAME OF THE AUTHOR: Prashanth Chandran, Gaspard Zoss, Paulo Gotardo, Derek Bradley
NAME OF THE JOURNAL: IEEE
YEAR OF PUBLISH: 2023
INFERENCE: This paper introduces a novel framework for facial landmark detection that predicts continuous, unlimited landmarks based on 3D queries. Unlike traditional methods that rely on a fixed set of annotated landmarks, this approach allows arbitrary landmark prediction at runtime by combining an image feature extractor with a queried landmark predictor. By using 3D queries relative to a template face mesh, the method adapts to various tasks such as dense landmark detection, 3D face reconstruction, and tracking facial anatomy. This flexibility eliminates the need for retraining when switching configurations and supports non-surface features like bone or jaw landmarks.

2.3 Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs

NAME OF THE AUTHOR: Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, Matthias Grundmann
NAME OF THE JOURNAL: IEEE
YEAR OF PUBLISH: 2020
INFERENCE: Introduces the lightweight and real-time Face Mesh model by Google Mediapipe, which is capable of detecting 468 landmarks.

2.4 Deep Alignment Network: A Convolutional Neural Network for Robust Face Alignment

NAME OF THE AUTHOR: Marek Kowalski, Jacek Naruniec, Tomasz Trzciński
NAME OF THE JOURNAL: 2017
YEAR OF PUBLISH: CVF
INFERENCE: This paper discusses landmark detection using convolutional networks, which is conceptually similar to Mediapipe's landmark regression.

2.5 Infinite 3D Landmarks: Improving Continuous 2D Facial Landmark Detection

NAME OF THE AUTHOR: Prashanth Chandran, Gaspard Zoss, Paulo Gotardo, Derek Bradley
NAME OF THE JOURNAL: 2023
YEAR OF PUBLISH: IEEE
INFERENCE: This paper extends continuous 2D landmark prediction with modifications for better annotation consistency and landmark visibility estimation, leveraging a 3D landmark prediction head.

2.6 Robust Facial Landmark Detection via Occlusion-adaptive Deep Networks

NAME OF THE AUTHOR: Meilu Zhu, Daming Shi, Mingjie Zheng, and Muhammad Sadiq
NAME OF THE JOURNAL: IEEE
YEAR OF PUBLISH: 2019
INFERENCE: This paper discusses methods for landmark prediction under occlusion using adaptive deep networks, complementing the continuous detection methods for robust-

ness.

CHAPTER 3

SOFTWARE REQUIREMENTS

3.1 GOOGLE COLABROTARY

Google Colab is a complimentary, cloud-based service that offers a user-friendly environment for executing Python code, particularly in the realms of data science, machine learning, and deep learning. Based on the Jupyter Notebook framework, Colab allows users to write and run code in an interactive notebook format, catering to both novice and experienced users. A key feature of this platform is the provision of free GPU (Graphics Processing Unit) and TPU (Tensor Processing Unit) resources, which significantly enhance the performance of tasks that involve large datasets and intensive computations, such as the training of deep neural networks.

Colab is compatible with widely-used machine learning libraries, including TensorFlow, PyTorch, and Keras, enabling users to construct, train, and evaluate models without the need for specialized hardware. Additionally, it seamlessly integrates with Google Drive, facilitating straightforward storage and retrieval of data directly from the notebook, and offers direct access to GitHub, allowing users to clone and execute code from repositories within the Colab environment.

For those requiring greater computational power, Google provides Colab Pro, which offers enhanced memory, extended runtimes, and superior GPU performance. Google Colab has become increasingly popular in educational and research settings, as it eliminates obstacles to high-performance computing and offers a versatile platform for experimenting with machine learning models. Given these features, Colab serves as an excellent option to fulfill the computational demands of this project, enabling effective model training and testing in a cloud-based setting.

CHAPTER 4

PACKAGES AND MODEL REQUIREMENTS

4.1 OpenCV (CV2)

The 'cv2' library in Python is an integral component of OpenCV (Open Source Computer Vision Library), which serves as a powerful framework for tasks related to computer vision and image processing. The library provides a variety of features for image and video processing, such as reading and writing images, displaying them, and performing transformations including resizing, rotating, and flipping. OpenCV facilitates color space conversion (e.g.from RGB to grayscale) and various filtering techniques such as Gaussian blur and edge detection.It also provides the ability to draw shapes and text on images, detect edges, and recognize objects, which are essential for applications such as face recognition and object tracking. In addition to image manipulation, 'cv2' also allows video processing by capturing footage from a file or camera and saving the processed results to a new file. It includes advanced features such as machine learning integration, object detection through pre-trained classifiers (such as the Haar cascade), and stereo vision for camera calibration.

4.2 TENSOR-FLOW

TensorFlow, an open-source machine learning framework developed by Google, has emerged as a cornerstone in the field of artificial intelligence. Renowned for its flexibility, scalability, and robustness, TensorFlow provides a comprehensive ecosystem for building and deploying machine learning models across a range of applications. Its intuitive high-level APIs simplify the development process, while its low-level operations offer fine-grained control for advanced users. TensorFlow's versatility extends from traditional neural networks to cutting-edge deep learning architectures, making it a go-to choice for researchers, developers, and enterprises seeking to harness the power of machine learning for diverse tasks such as image recognition, natural language processing, and more.

4.3 TIME

The ‘time‘ library in Python is a comprehensive module that offers a variety of functions for managing time-related operations, making it indispensable for numerous applications. It enables developers to work with both the current time and system time. The ‘time()‘ function provides the current time as a floating-point number, which represents the number of seconds elapsed since the epoch (January 1, 1970).

For converting time into a human-readable format, the ‘ctime()‘ function transforms time into a string, while ‘strftime()‘ and ‘strptime()‘ facilitate the formatting and parsing of date and time. The ‘sleep()‘ function is commonly utilized to introduce pauses in program execution, proving beneficial for time-sensitive applications such as animations or API rate-limiting. Additional functions like ‘localtime()‘ and ‘gmtime()‘ assist in retrieving and manipulating time in local or UTC formats.

Given its extensive capabilities, the ‘time‘ module is a crucial component of Python, allowing for efficient time management in applications such as scheduling, logging, and performance monitoring.

4.4 MEDIAPIPE

MediaPipe is an open-source framework developed by Google to support real-time perception in a wide range of applications, leveraging machine learning (ML) for functions such as pose detection, hand tracking, and facial landmark recognition. The framework has a modular pipeline architecture that simplifies the deployment of complex ML pipelines across multiple platforms, including mobile and web environments. The platform provides a range of predefined models and pipelines that make it easier to integrate ML solutions into applications, especially for users with limited machine learning experience. MediaPipe includes a full range of off-the-shelf solutions, such as face detection, hand tracking, pose estimation, and object detection, among others.

Each pipeline integrates ML models with customizable components, including image preprocessing, inference, and post-processing modules, improving both flexibility and scalability. One of the key advantages of MediaPipe is its cross-platform and low-latency capabilities, enabling real-time performance on both CPUs and GPUs. This efficiency is especially beneficial in resource-constrained environments, such as mobile devices.

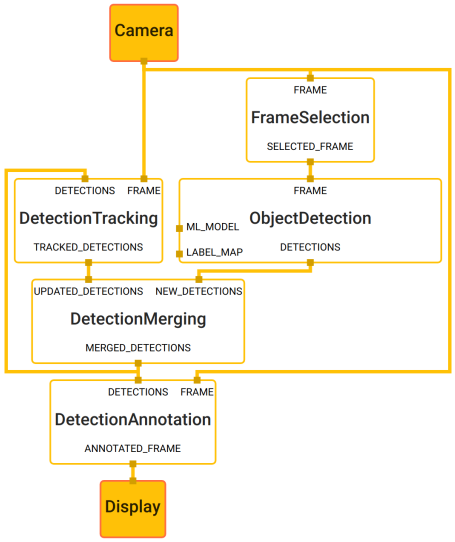

Figure 4.1: Mediapipe Architecture
Credit: MediaPipe- A Framework for Building Perception Pipelines.

Additionally, MediaPipe's graphing API provides a visual representation of the data flow in the pipeline, facilitating debugging and optimization. With TensorFlow Lite compatibility and GPU acceleration, MediaPipe provides a highly efficient solution for on-device inference, improving user privacy by eliminating the need for cloud processing. Developers can also build custom pipelines using a Python or C++ interface, making MediaPipe adaptable to both research and production environments. This versatility contributes to MediaPipe's popularity in areas such as computer vision, augmented reality, and human-computer interaction.

CHAPTER 5

METHODOLOGY

5.1 FACIAL LANDMARKS PREDICTIONS

In facial landmark prediction, the main idea is to identify the most prominent points of interest in a face, such as the eyes, the nose, the mouth, and the jaw line. These landmarks have been utilised in many applications and have aided the completion of various functions, among which include face alignment, emotion recognition, augmented reality effects, and facial identification.

Facial landmark prediction is essentially processing an image of a face toward the location of specific landmarks by estimating the coordinates of some predetermined key points. In most cases, it uses trained machine learning models trained on annotated datasets which enable facial features recognition across different poses, lighting conditions, and expressions.

For instance, Face Mesh in MediaPipe can predict facial feature landmarks in real time at 468 landpoints. This is made with deep learning algorithms that have very accurate results as well as GPU acceleration, cutting latency significantly, thereby suitable for mobile and web applications alike. Facial landmark prediction has vast implications into various application fields, like animation, health diagnostics, and authentication of the user, which leads to enhanced user interactions with technology via accurate facial analysis and real-time processing.

Figure 5.1: Basic Facial Landmarks Prediction
Credits: A Detailed Look At CNN-based Approaches In Facial Landmark Detection .

1. **Data Collection:**

   Gather a diverse and representative dataset containing annotated images or videos with labeled keypoint locations. This dataset should cover a wide range of landmark annotations, variations in lighting, and diverse backgrounds.

2. **Preprocessing:**

   Preprocess the dataset by normalizing images, resizing them to a consistent resolution, and augmenting data to enhance the model's generalization capabilities. Additionally, perform any necessary data cleaning and filtering to ensure the quality of annotations.

3. **Model Selection:**

   Choose a suitable facial Landmark prediction model architecture based on the specific requirements of the task. Popular choices include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more recently, transformer-based models. Consider factors such as model complexity, accuracy, and real-time performance.

4. **Model Training:**

   Train the selected model on the preprocessed dataset. This involves optimizing the model's parameters using a loss function that measures the disparity between predicted keypoints and ground truth annotations. Employ techniques like transfer learning if applicable, leveraging pre-trained models on large datasets.

5. **Post-Processing:**

   Implement post-processing techniques to refine the output of the model. This may include smoothing techniques, filtering out outlier keypoints, or considering temporal information in video sequences for improved accuracy.

6. **Evaluation Metrics:**

   Define appropriate evaluation metrics to assess the performance of the model. Common metrics include accuracy, precision, recall, and the mean average precision (mAP) for multi-person pose estimation. Utilize a validation set to fine-tune model hyper parameters and ensure robust generalization.

7. **Deployment:**

   Integrate the trained model into the desired application or system. Ensure compatibility with input sources (images or videos) and optimize for real-time performance if necessary. Consider deploying the model on edge devices or in cloud environments based on the application requirements.

8. **Continuous Improvement:**

   Continuously monitor the model's performance in real-world scenarios and collect additional annotated data if possible. Implement ongoing model updates and improvements to adapt to evolving challenges and enhance overall accuracy.

5.1.1 SPARSE FACIAL LANDMARK PREDICTION

Sparse facial landmark prediction is one of the directed approach within the facial landmark detection field. This technique relies on identifying a sparse number of critical points on the face, which frequently includes regions like the eyes, nose, and corners of the mouth. While the dense landmark prediction method maintains the location of hundreds of points, a general sparse landmark prediction can identify roughly 5-15 landmarks, meaning less computational resource usage and shorter processing periods.

This approach is particularly beneficial in applications that require real-time performance as well as minimizing the usage of resources, including mobile face detection or even gaze tracking. Sparse facial landmark prediction, actually, involves the use of learned machine learning models trained on sets of labeled landmarks data. With such models, it is possible to rapidly estimate key points under such conditions as variations in illumination and occlusions.

Sparse facial landmark prediction is particularly well-suited for face alignment, where a few landmarks are sufficient to normalize the orientation of the face. In emotion detection applications, sparse predictions are also highly basic because a couple of points will suffice to express what people are trying to convey and achieve a balance between efficiency and accuracy.

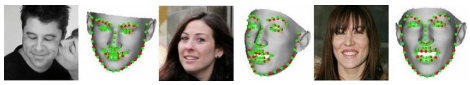

Figure 5.2: Sparse Facial Landmarks
Credits: Deep, Landmark-Free FAME: Face Alignment, Modeling, and Expression Estimation.

1. **Library Imports:**
   Import essential libraries, including OpenCV (`cv2`), Mediapipe, and Time. `IPython.display` is used for video embedding.

2. **Sparse Landmark Detector Class:**
   Define the `SparseLandmarkDetector` class to initialize Mediapipe's Face Detection or Face Mesh solution, configured for sparse landmark detection. Set parameters for:

   - `staticMode` for processing static images or real-time video,
   - `maxFaces` to set the maximum number of faces to detect,
   - `minDetectionCon` and `minTrackCon` for controlling detection and tracking confidence.

Within the class, initialize a Mediapipe detector to retrieve sparse landmarks.

3. **Video Capture Initialization:**
   Use `cv2.VideoCapture` to open a video file or camera stream. Retrieve properties like frame width, height, and frames per second (FPS).

4. **Output Video Writer:**
   Set up an output video writer (`cv2.VideoWriter`) to save the processed video with sparse landmarks drawn.

5. **Main Processing Loop:**
   Process each frame within a loop:

   (a) Convert the frame to RGB for Mediapipe compatibility.

   (b) Call the `findSparseLandmarks` method to detect sparse landmarks.

   (c) If `draw=True`, render the sparse landmarks on the frame.

   (d) Extract the sparse landmark coordinates for each detected face for reference or analysis.

6. **FPS Calculation:**
   Dynamically calculate frames per second (FPS) to monitor the performance in real time.

7. **Rendering and Saving:**
   Display the calculated FPS on each frame and save the modified frames to the output video file.

8. **Release Resources:**
   After processing, release the video capture and output writer resources and display a confirmation message.

9. **Display Processed Video:**
   Use `IPython.display.Video` to embed and preview the processed video.

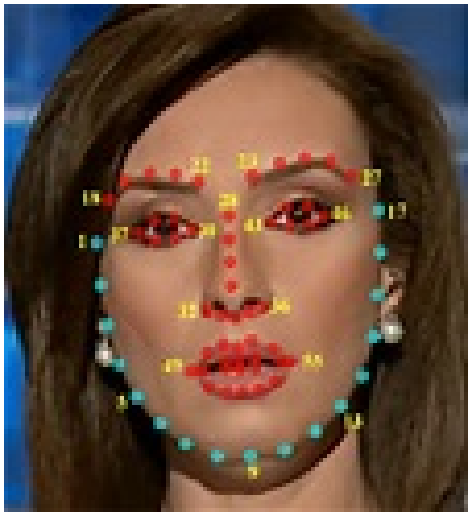

Figure 5.3: 68 Points Facial Landmark Prediction
Credits: Deepfake Detection Using Robust Spatial and Temporal Features from Facial Landmarks.

5.1.2 DENSE FACIAL LANDMARK PREDICTION

Dense facial landmark prediction, or face mesh generation, applies deep learning algorithms to detect many key points in a face that can be used for facial recognition, 3D reconstruction, and augmented reality. The preliminary step involves data preparation which is the gathering of annotated images of faces with dense landmark points or face meshes obtained from different angles to cover more of expressions, lighting conditions and poses. These pictures are then normalized and preprocessed for feature detection from various aspects by aligning, cropping, and scaling the images. Features are extracted using the CNN or rather a more efficient architecture, such as ConvNet or MobileNet, resulting in the spatial hierarchies of facial characteristics in the form of feature maps. A landmark prediction network takes the feature maps as input and predicts dense 2D or 3D landmark points by dense regression or heatmap-based methods that give a probability distribution for each point. The final step is post-processing in which the above predictions are smoothed to remove any inconsistencies, and registered against an image using a 3D face model or mesh to maintain coherence. In summary, dense landmark points are used in reconstructing or fitting a 3D face mesh, which further allows for high-precision applications, including face segmentation, performance capture, and animated avatars.

Figure 5.4: Face Mesh Prediction
Credits: ml-kit/vision/face-mesh-detection.

1. **Library Imports:**
   Import necessary libraries, including OpenCV (`cv2`), Mediapipe, Time, and `IPython.display` for video embedding.

2. **Face Mesh Detector Class:**
   Define the `FaceMeshDetector` class for initializing Mediapipe's Face Mesh solution. Set parameters for static or real-time processing, maximum faces, detection confidence, and tracking confidence. Include methods to process frames and draw face mesh landmarks.

3. **Video Capture Initialization:**
   Open a video file using OpenCV (`cv2.VideoCapture`) and retrieve properties like frame width, height, and frames per second (FPS).

4. **Output Video Writer:**
   Define an output video writer (`cv2.VideoWriter`) to save the processed video with rendered face mesh.

5. **Main Processing Loop:**
   Loop through video frames, checking if each frame is successfully captured. For each frame:

   (a) Convert the image to RGB for Mediapipe processing.

   (b) Use the `findFaceMesh` method to detect and draw face mesh landmarks.

   (c) Retrieve face landmark coordinates for further use or analysis.

6. **FPS Calculation:**
   Compute the frames per second (FPS) dynamically using timestamps to monitor performance.

7. **Rendering and Saving:**
   Display FPS on the video frame and save it to the output file.

8. **Release Resources:**
   Once processing is complete, release video capture and writer resources. Output a confirmation message.

9. **Display Processed Video:**
   Use `IPython.display.Video` to embed and preview the processed video.

5.1.3 FACIAL LANDMARK PREDICTION ON MULTIPLE FACES

The Facial Landmark prediction on multiple faces starts by instantiating the ‘FaceMeshDetector‘ class, which invokes MediaPipe’s FaceMesh for facial landmark detection and visualization. This class accepts various parameters, such as ‘staticMode‘, ‘maxFaces‘, ‘minDetectionCon,‘ and ‘minTrackCon,‘ controlling real-time tracking with a maximum number of faces to be detected and its confidence thresholds for both detection and tracking. Following that, the application reads in a video file with OpenCV’s ‘VideoCapture‘ and creates a ‘VideoWriter‘ to store the output with facial landmarks.

Then for each processed frame, it fetches the image, converts it from BGR to RGB format, and then applies the FaceMesh model from MediaPipe to detect facial landmarks. The code captures all landmarks for every detected face and draws them on the video using contours with line width and radius parameters that can be modified. It then calculates and shows frames per second for each frame in order to measure performance. Ultimately, the program writes out the processed frames, with the rendered mesh already included, into an output video file and releases the resources at the end of processing. The outcome will be a video that shows exactly overlaid face meshes for multiple detected faces in real time, guaranteeing solid multi-face landmark tracking even for dynamic or visually intricate scenarios.

Figure 5.5: Face Mesh Prediction on Multiple Faces
Credits: medium.com.

1. **Library Imports:**

Import necessary libraries, including `cv2` for video processing, Mediapipe for facial landmark detection, `time` for FPS calculation, and `IPython.display` for embedding the output video in Google Colab.

2. **Google Drive Integration:**
   Mount Google Drive using `drive.mount()` to load input video files and save the processed output video directly to the cloud storage.

3. **Face Mesh Detector Class:**
   Define the `FaceMeshDetector` class to initialize Mediapipe's Face Mesh solution with customizable parameters:

   - `staticMode` for real-time or static image processing,
   - `maxFaces` to set the maximum number of faces to detect,
   - `minDetectionCon` and `minTrackCon` to control detection and tracking confidence thresholds.

   The class includes a `findFaceMesh` method to process each frame and optionally highlight a specified number of landmarks.

4. **Input for Landmark Highlighting:**
   Prompt the user to specify the number of landmarks to highlight, defaulting to 68 if no valid input is provided. Ensure the input does not exceed the maximum number of landmarks supported (468 for MediaPipe Face Mesh).

5. **Video Capture Initialization:**
   Load a video file from Google Drive using `cv2.VideoCapture`, and retrieve properties such as frame width, height, and frames per second (FPS).

6. **Output Video Writer:**
   Set up a `cv2.VideoWriter` to save the processed video with highlighted landmarks to a specified location in Google Drive.

7. **Main Processing Loop:**
   Process each video frame within a loop:

   (a) Convert the frame to RGB for Mediapipe compatibility.

   (b) Call the `findFaceMesh` method to detect and highlight the specified number of landmarks.

   (c) Optionally, print the coordinates of the first detected face for debugging or reference.

8. **FPS Calculation:**
   Compute the frames per second dynamically using timestamps to monitor real-time performance and display the FPS on each frame.

9. **Rendering and Saving:**
   Draw the landmarks and FPS on the video frames, then write the modified frames to the output video file.

10. **Release Resources:**
    After processing, release the video capture and output writer resources. Print a confirmation message indicating the output file location in Google Drive.

11. **Display Processed Video:**
    Embed and preview the processed video directly in Google Colab using `IPython.display.Video`.

5.1.4 FACIAL LANDMARK PREDICTION FOR GIVEN NUMBER OF POINTS

Facial landmark prediction for given number of points enables flexible and scalable facial landmark detection because a model can detect an arbitrary number of landmarks across varied configurations without any need to retrain. Methods require a fixed set of predefined landmarks and, therefore cannot be adaptable to any application. However, whereas continuous prediction models such as the one presented above must use a 3D query-based approach with landmark positions that are designated at runtime based on the task requirements, the model makes use of an image feature extractor and a queried landmark predictor by which users can input points within 3D space on a face template in order to obtain the corresponding coordinates of images appearing in 2D space. This allows the model to account for sparse and dense configurations of landmarks, further predicting landmark locations beyond facial surfaces such as bone or teeth positions. Flexibility in this opens up applications in face reconstruction, segmentation, and performance capture, thereby offering a completely and adaptively comprehensive solution to detection across different contexts for facial landmarks.

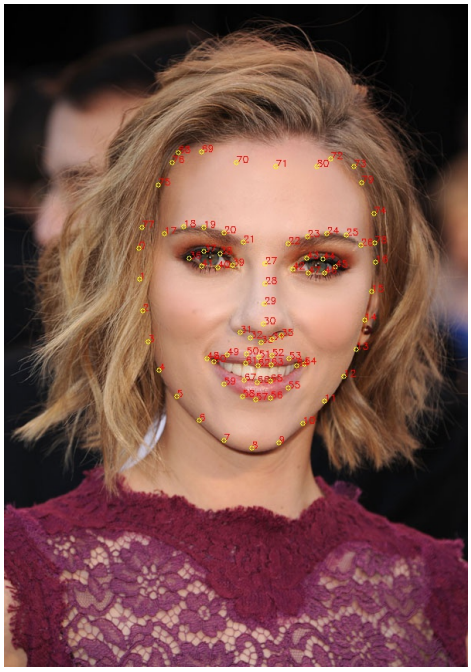

Figure 5.6: Facial Landmark Prediction for given number of points as Input
Credits: codeniko.

1. **Library Imports:**
   Import essential libraries, including OpenCV (`cv2`) for video processing, Mediapipe for facial landmark detection, `time` for frame rate calculations, and `IPython.display` for embedding the output video in interactive environments.

2. **Google Drive Integration:**
   Mount Google Drive to facilitate input video loading and saving the processed output directly to the cloud storage for easy access.

3. **Face Mesh Detector Class:**
   Design the `FaceMeshDetector` class, which initializes Mediapipe's Face Mesh solution. Include the following parameters:

   - `staticMode` to enable either static image or real-time video processing.
   - `maxFaces` to specify the maximum number of faces to detect.
   - `minDetectionCon` and `minTrackCon` for controlling detection and tracking confidence levels.

   Add methods to process video frames, detect landmarks, and highlight specific ones as per user input.

4. **Video Capture Initialization:**

Open the video file using `cv2.VideoCapture` and extract properties like frame width, height, and FPS for processing and saving output.

5. **User Input for Landmark Highlighting:**
   Accept user input for the number of landmarks to highlight. Validate the input to ensure it does not exceed the maximum supported landmarks (468 for MediaPipe Face Mesh).

6. **Main Processing Loop:**
   Process each video frame in a loop:

   (a) Convert each frame to RGB for Mediapipe compatibility.

   (b) Detect landmarks using the `findFaceMesh` method and highlight user-specified landmarks with a distinct color.

   (c) Overlay the frames with FPS information for real-time performance monitoring.

7. **Output Video Writer:**
   Define an output writer using `cv2.VideoWriter` to save the processed video with highlighted landmarks to a specified path on Google Drive.

8. **Rendering and Saving:**
   Render detected landmarks and FPS information on each frame, then write the modified frames to the output video file.

9. **Release Resources:**
   Once processing is complete, release all video-related resources and confirm the saved output file's location in Google Drive.

10. **Embedding Processed Video:**
    Use `IPython.display.Video` to embed and preview the processed video file directly within the Colab notebook environment.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 SPARSE LANDMARKS PREDICTION:

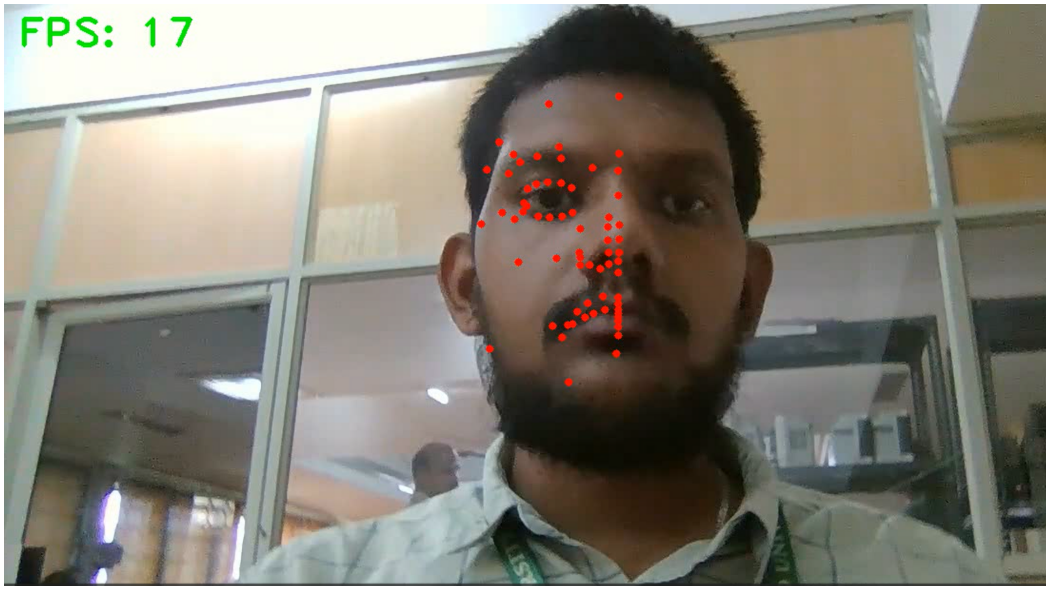

Figure 6.1: SPARSE FACIAL LANDMARK PREDICTION

This code will read through a video and look for every face, then spot a 68-point facial landmark mesh on all of the faces. It uses the ‘mediapipe‘ Face Mesh model, which positions rectifications and maps 68 points of interest onto the face, to set up the settings so that you will view only the standard landmark points drawing out the features of the face. It reads frames from a video file, executes landmark detection, overlays mesh onto it and saves all the annotated frames to an output video file. In addition, it measures and displays the FPS for performance monitoring. This approach yields a real-time annotated video depicting the accurate facial landmarks of all the faces in every frame.

6.2 DENSE LANDMARKS PREDICTION:

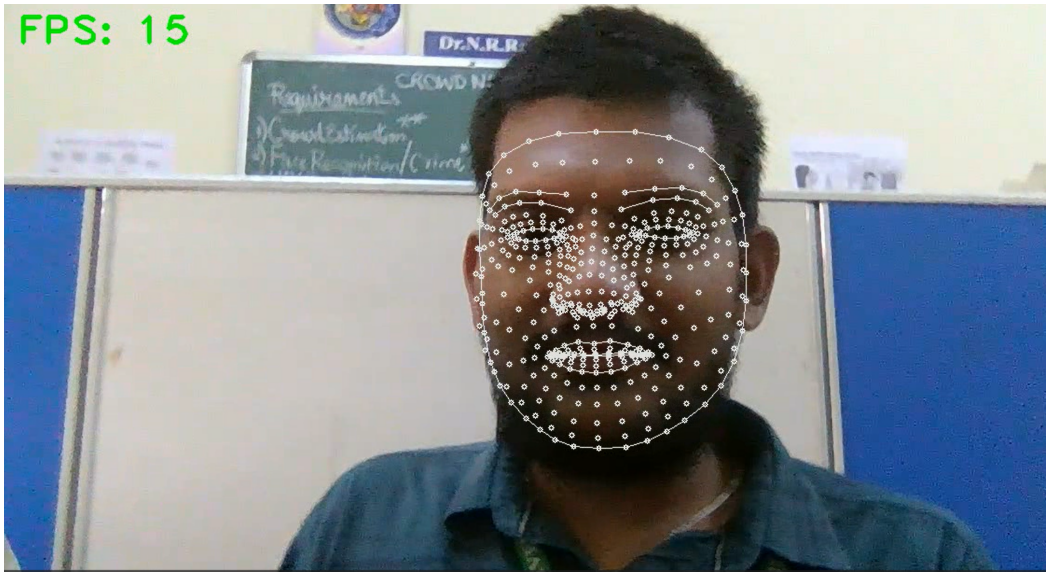

Figure 6.2: DENSE FACIAL LANDMARK PREDICTION

This code processes a video that detects and overlays facial landmarks using MediaPipe's Face Mesh model. It initializes the 'FaceMeshDetector' class to indicate detection and tracking confidence levels. It then goes through the detected frames of a video, identifies facial landmarks, overlays the mesh for every face detected, and computes FPS for reference. After processing, it saves the annotated video as 'output with mesh.mp4'. The resultant output video will show a real-time facial mesh overlay, marking key facial landmarks, and a dynamic FPS counter, thus providing a very smooth facial tracking experience.

6.3 FACE LANDMARK PREDICTION ON MULTIPLE FACES

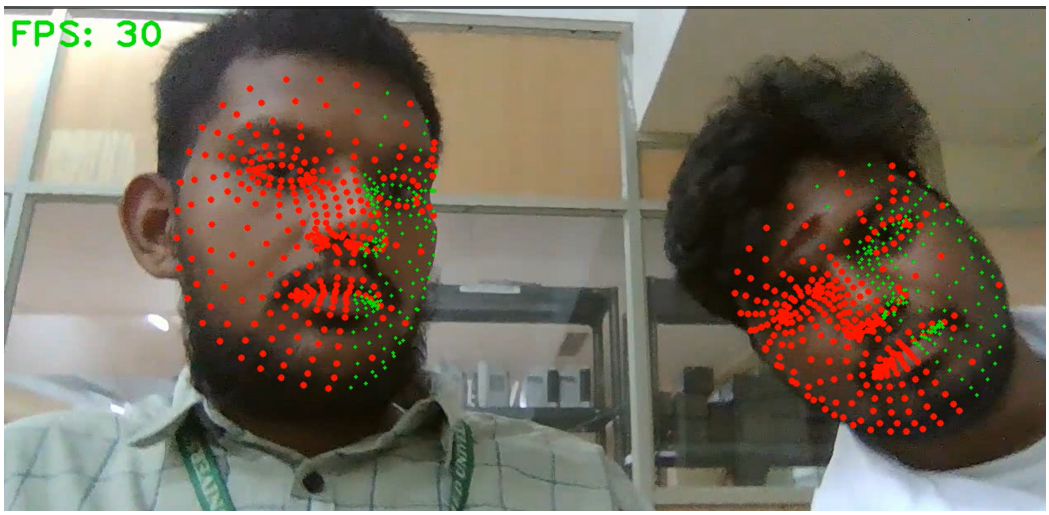

Figure 6.3: FACIAL LANDMARK PREDICTION ON MULTIPLE FACES

The FaceMeshDetector code employs MediaPipe to identify and overlay facial landmarks from several faces within a video, achieving efficient landmark prediction even in intricate scenes. The code initializes the FaceMeshDetector with selected parameters, including the maximum number of detectable faces (‘maxFaces = ”faces in the scene”‘). During video processing, each frame is processed through the ‘findFaceMesh‘ method, which converts the image to RGB, captures the facial landmarks, and overlays them onto the video with contours for enhanced clarity. The resulting output, ‘output with mesh.mp4‘, will feature the landmarked mesh of each face in real-time video, along with the frames per second (FPS) information. This script is capable of performing detailed facial tracking, even in scenarios involving multiple faces.

6.4 FACE LANDMARK PREDICTION FOR GIVEN NUMBER OF INPUTS:

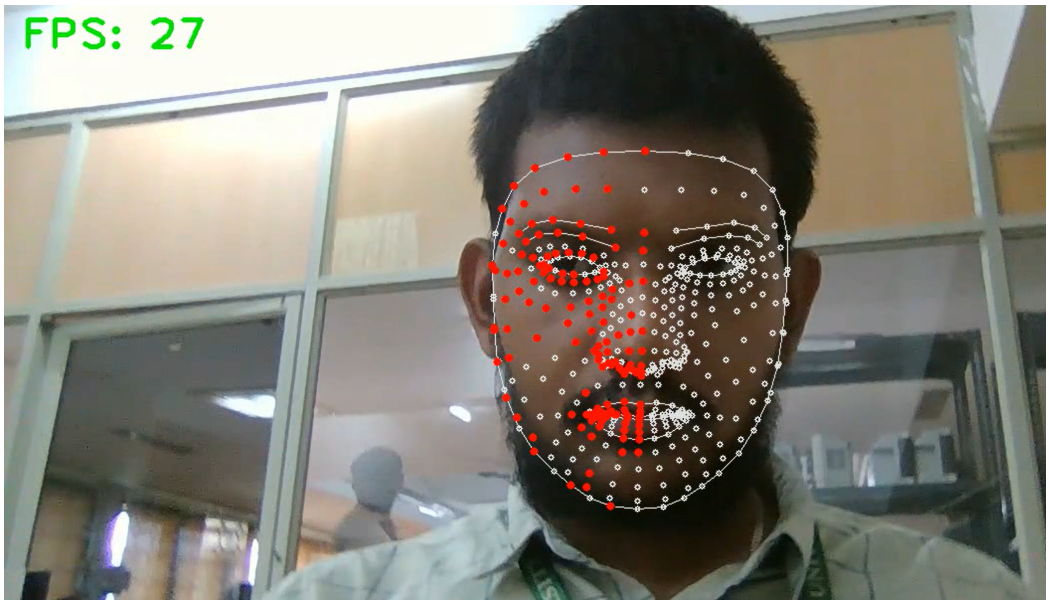

Figure 6.4: FACIAL LANDMARK PREDICTION FOR GIVEN INPUT HIGHLIGHTED IN FACE MESH

Figure 6.5: FACIAL LANDMARK PREDICTION FOR GIVEN INPUT IN MULTIPLE FACES

This code detects and superimposes facial landmarks on a video that can also emphasize an actual count of points. It will import the Google Drive for accessing and storing the file before initializing a 'FaceMeshDetector' class for frame analysis in using MediaPipe's Face Mesh technology. The User can specify the number of facial landmarks to emphasize-it has a maximum limit of 468 and is visualized as red circles for better visualization. It also calculates and prints the frame per second (FPS) for each frame, thus saving the marked

video as ‘output with given points landmarks.mp4‘ on Google Drive and playing it back with marked facial points along with live FPS values.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1 CONCLUSION

The execution of facial landmark detection using Google MediaPipe is an effective way of showing the potentiality of current machine learning frameworks in their applications to face tracking tasks with the objective of real-time high-precision face tracking. The pre-trained model was effectively used for facial landmark detection to monitor applications requiring detailed facial structure analysis. It utilized the computational capabilities of Google Colab. This execution did not depend on local hardware and, hence, was made easy, open, and scalable.

The results underscore the effectiveness of the system in identifying crucial points on the face with low misclassifications, especially with variable illumination and pose changes. This study shows that cloud-based platforms can be combined with state-of-the-art neural network models toward realizing efficient computer vision applications that are dependable. The possible future enhancements involve some extensions of the application that may include 3D reconstruction of faces or flexibility improvement on dense landmark prediction in complex scenarios. With regard to this, the project has a solid framework for the development of advanced facial recognition and tracking technologies.

7.2 FUTURE WORKS

An important area for the future will be further development of the facial landmark detection system for further improvements and widespread applications. In particular, dense landmark detection should be improved to capture even more detailed features. It will be easier to detect 3D landmarks to enable augmented reality and 3D model applications. Another important area is refining the system so that it can run in real time, especially in live applications. Enhanced functionalities of cross-platform deployment will make the system compatible with mobile and IoT devices. The customizable landmark detection allows tracking distinctive features such as moles or scars. Multimodal inputs like voice and gaze improve human-computer interaction. Mapping landmarks onto virtual assets leads to in-

novative AR/VR applications, such as face filters or gesture controls.

In other words, all these gains are achieved due to training on large datasets, usage of architectures and tools such as TensorFlow Lite for mobile optimization, and so on. Adding all these improvements into the system allows it to support vast ranges of applications in the domains of medical imaging, interactive technologies, and more, leading to more robust and versatile end products.

REFERENCES

[1] P. Chandran, G. Zoss, P. Gotardo, and D. Bradley. Infinite 3d landmarks: Improving continuous 2d facial landmark detection. *Computer Graphics Forum*, 43(6), June 2024.

[2] Prashanth Chandran, Gaspard Zoss, Paulo Gotardo, and Derek Bradley. Continuous landmark detection with 3d queries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16858–16867, June 2023.

[3] Google. Mediapipe: Face mesh. https://google.github.io/mediapipe/solutions/face_mesh.html, 2023. Accessed: 2024-10-28.

[4] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. In *CVPR Workshop on Computer Vision for Augmented and Virtual Reality 2019*, Long Beach, CA, 2019.

[5] Marek Kowalski, Jacek Naruniec, and Tomasz Trzcinski. Deep alignment network: A convolutional neural network for robust face alignment. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2034–2043, 2017.

[6] Ritesh Prajapati. Facial landmarks and face detection in python with opencv. https://medium.com/analytics-vidhya/facial-landmarks-and-face-detection-in-python-with-opencv-73979391f30e, 2020. Accessed: 2024-09-25.

[7] Adrian Rosebrock. Real-time facial landmark detection with opencv, python, and dlib. https://pyimagesearch.com/2017/04/17/real-time-facial-landmark-detection-opencv-python-dlib/, 2017. Accessed: 2024-10-15.

[8] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin

Xiao. Deep high-resolution representation learning for visual recognition. *CoRR*, abs/1908.07919, 2019.

[9] Libing Zeng, Lele Chen, Wentao Bao, Zhong Li, Yi Xu, Junsong Yuan, and Nima K. Kalantari. 3d-aware facial landmark detection via multi-view consistent training on synthetic data. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12747–12758, 2023.

[10] Meilu Zhu, Daming Shi, Mingjie Zheng, and Muhammad Sadiq. Robust facial landmark detection via occlusion-adaptive deep networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3481–3491, 2019.

[11] Computer Vision Zone. Facial landmarks course. https://www.computervision.zone/courses/facial-landmarks/, 2024. Accessed: 2024-09-15.