Face Symmetry Analysis (Group 6)

Shivam Sharma(210984)
Chirag Garg(210288)
Paritosh Pankaj(210702)

November 9, 2024

Abstract

This project focuses on developing a metric to determine the symmetry of a human face from an image. Our approach includes background removal, key point assignment, and a combination of three distinct symmetry calculation methods: landmark distance comparison, midline distance normalization, and angular symmetry evaluation.

1 Introduction

Symmetry in human faces is a key factor in fields like aesthetic analysis, facial recognition, and psychology. This project aims to quantify facial symmetry by applying a series of mathematical computations to facial landmarks.

2 Methodology

2.1 Novelty

The novelty of our approach lies in the fact that instead of using a machine learning based landmark detection algorithm we use a keypoint detection algorithm which uses intensity values of the image to detect the important landmarks of the image and then use measures of linear and angular symmetry to get the final symmetry score. This approach adds new insights to the task of analyzig facial symmetry.

2.2 Data Preparation

- Capture or select an image of a face.
- Filtering the face to isolate the face in order to enable efficient (face specific) key point calculation to detect facial landmarks.
- Assign facial landmarks using standard keypoint detection model.

2.3 Symmetry Calculation Methods

2.3.1 Method 1: Landmark Distance Comparison

In this approach, we locate the essential keypoints capturing the important facial landmarks of the facial region like the chin, nose, lip corner, eye and the forehead, and try to calculate the distance between all of them for both left and right half of the face. To enable calculation for the other half, we just flip the image and apply the similar functions.

- **Mathematical Calculation**: Distance from chin landmark to each side landmark.
- **Symmetry Metric**: Average difference in distance between corresponding pairs of landmarks on each side.

2.3.2 Method 2: Midline Distance Normalization

In this method, a vertical midline(the distance between nose bridge and chin) is drawn through the center of the face.We select some of the specific facial landmarks like jawline, eyebrow and mouth points and locate their approximate keypoints (both on the left and right half of the face). Distances from each such landmark to this midline are calculated, normalized, using the width of the face(the leftmost and rightmost point of the face) and then compared across each side. We take mean of these normalized differences and construct a measure of the facial symmetry. (the higher value of this factor, less symmetrical are the faces).

- **Mathematical Calculation**: Distance of each landmark from the midline.
- **Symmetry Metric**: Normalized differences in distances across each side.

2.3.3 Method 3: Angular Symmetry Evaluation

This is one of the approaches which makes our entire project different and unique. Here, instead of going for the linear distances, we try to compute the angular distances of the key facial landmarks we mentioned in above two methods. The angle is computed by calculating the inner product between the coordinates of the landmark values assuming them as vectors. We do it for the left and the right half of the facial region and take their ratio to get symmetry ratios. Trivially, we average these different angular ratio to get the final angular symmetry ratio.

- **Mathematical Calculation**: Angle calculation between pairs of landmarks.
- **Symmetry Metric**: Average difference in angle for pairs on each side.

2.4 Final facial symmetry metric calculation

The final facial symmetry metric can be calculated as weighted linear combination of the symmetry scores obtained above, that is

$$r_f = \alpha r_1 + \beta r_2 + \gamma r_3$$

subject to

$$\alpha + \beta + \gamma = 1$$

where,
r_f is the final symmetry metric(the closer to one the better)
r_1 is the linear symmetry ratio
r_2 is the median symmetry ratio
r_3 is the angular symmetry ratio
α, β and γ are the corresponding weights which may depend on the type of problem being dealt with and decided upon by the user.

3 Implementation

3.1 Required Libraries

For this project, we used Python along with several libraries:

- `OpenCV` for image processing and image filteration.
- `medaipy, mediapipe` for facial landmark detection.
- `Numpy` for mathematical computations.
- `Matplotlib` for visualizing results.

4 Experimental Results

- Display processed images with detected landmarks.
- Present calculated symmetry scores for each method.

Figure 1: Sample facial landmark detection and symmetry visualization.

Figure 2: Facial region after filtering.

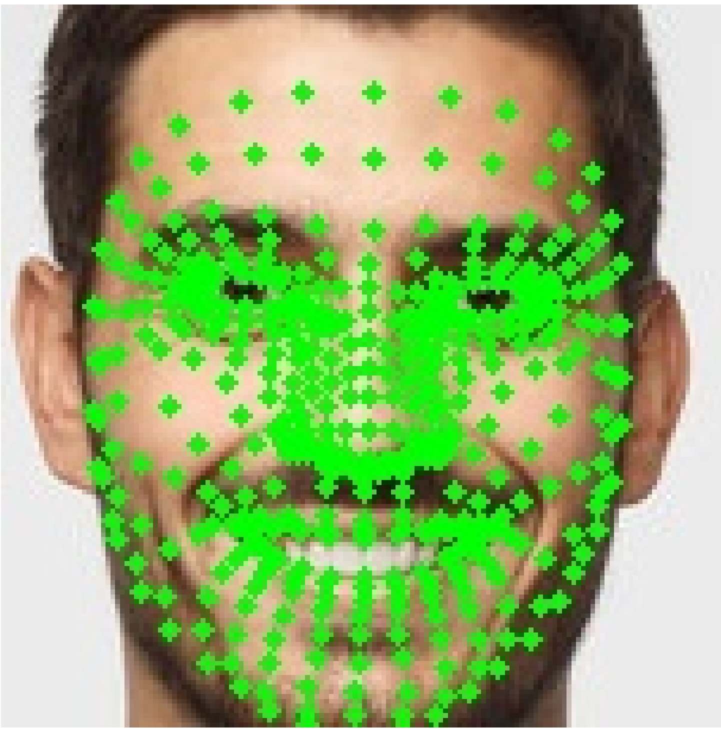

Figure 3: Face with identified keypoints.

We show keypoint calculation of a rather asymmetrical figure.

Figure 4: Sample facial landmark detection and symmetry visualization.

Figure 5: Facial region after filtering.

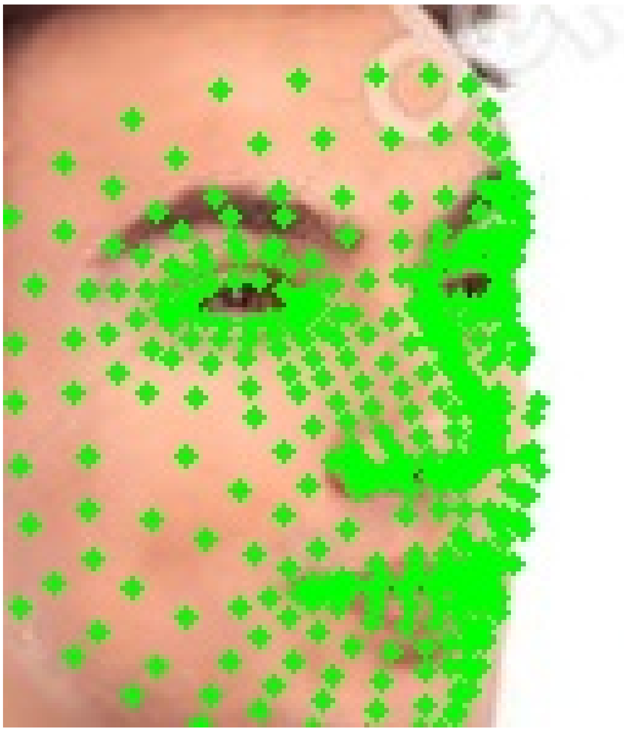

Figure 6: Face with identified keypoints.

The calculated symmetry scores for these figures figure are as follows:

Images	Linear ratio	Angular ratio	Median ratio	Median distance
Image 1	0.9725	1.0011	1.0538	2.14%
Image 2	0.8964	0.9926	3.144	28.36%
Image 3	1.0001	0.9983	1.1085	2.03%
Image 4	0.9698	1.0077	1.0291	2.67%
Image 5	0.977	1.0083	1.3217	10.54%
Image 6	0.9498	1.0099	1.2122	7.63%
Image 7	0.9626	1.0054	1.0803	5.23%
Image 8	1.0140	1.0074	0.85750	1.21%
Image 9	0.9783	0.9944	1.0575	3.65%
Image 10	1.0310	1.0013	0.7870	7.708%
Image 11	1.0101	0.9910	0.8853	3.53%
Image 12	1.1791	1.0130	0.5194	25.4%

Table 1: Comparison of symmetry ratios of above two figures

5 Conclusion

This project demonstrates a systematic approach to quantifying facial symmetry. The three methods provide varied insights into facial symmetry, useful for applications in medical diagnostics, facial recognition, and aesthetics. Although this approach is not full proofed and has some shortcomings as we have shown it through the two examples considered above. The user may be compelled to switch back and forth within the three methods by altering the values of the weight factors, that is α,β and γ.

6 APPENDIX

6.1 Link to dataset used

We have created a custom dataset for our analysis by handpicking images of different types and measuring the novelty of our approach.

Link for the google drive containing dataset in zipped format: Datasets

6.2 Code

We present the code for our method and evaluations :

```
import cv2
import math
import mediapy as media
import mediapipe as mp
import numpy as np

def crop_face(image_path, output_path):
    # Load the pre-trained Haar Cascade classifier for face
        detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
        "haarcascade_frontalface_default.xml")

```

```python
    # Read the image
    image = cv2.imread(image_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray_image, scaleFactor
        =1.1, minNeighbors=5, minSize=(30, 30))

    # Crop and save the first detected face
    for (x, y, w, h) in faces:
        face_crop = image[y:y+h, x:x+w]
        cv2.imwrite(output_path, face_crop)
        break  # Process only the first detected face

# Example usage
crop_face("/content/image_2_.jpeg", "output_face_2.jpg")


## plot face with keypoint detection
import cv2
import mediapipe as mp
import mediapy as media
import matplotlib.pyplot as plt

# Initialize MediaPipe Face Mesh
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=True,
    max_num_faces=1, min_detection_confidence=0.5)

def detect_landmarks(image):
    """
    Detects facial landmarks using MediaPipe Face Mesh.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        list: List of (x, y) tuples for facial landmarks.
    """
    results = face_mesh.process(cv2.cvtColor(image, cv2.
        COLOR_BGR2RGB))
    if not results.multi_face_landmarks:
        return None

    landmarks = []
    for landmark in results.multi_face_landmarks[0].landmark:
        x = int(landmark.x * image.shape[1])
        y = int(landmark.y * image.shape[0])
        landmarks.append((x, y))
    return landmarks

```

```
def plot_landmarks(image, landmarks):
    """
    Plots the facial landmarks on the image.

    Parameters:
        image (numpy.ndarray): The input image.
        landmarks (list): List of (x, y) coordinates of landmarks.
    """
    if landmarks is None:
        print("Error: No landmarks detected.")
        return

    # Draw each landmark point
    for (x, y) in landmarks:
        cv2.circle(image, (x, y), 2, (0, 255, 0), -1)

    # Display the image with landmarks using Mediapy
    media.show_image(image, title="Facial Landmarks")

    cv2.imwrite("face1_landmarks_2.jpg", image)

# Example usage:
image_path = "/content/output_face_2.jpg"
image = cv2.imread(image_path)

if image is None:
    print("Error: Unable to read the image.")
else:
    # Detect and plot landmarks
    landmarks = detect_landmarks(image)
    plot_landmarks(image, landmarks)


### calculate linear and angular symmetry ratios


# Initialize MediaPipe Face Mesh
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=True, max_num_faces=1, min_detection_confidence=0.5)

def detect_landmarks(image):
    """Detects facial landmarks using MediaPipe's Face Mesh."""
    results = face_mesh.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    if not results.multi_face_landmarks:
        return None

    landmarks = []
    for landmark in results.multi_face_landmarks[0].landmark:
```

```python
        x = int(landmark.x * image.shape[1])
        y = int(landmark.y * image.shape[0])
        landmarks.append((x, y))
    return landmarks

def calculate_angle(p1, p2, p3):
    """Calculates the angle formed by the points p1 -> p2 -> p3."""
    v1 = np.array(p1) - np.array(p2)
    v2 = np.array(p3) - np.array(p2)
    dot_prod = np.dot(v1, v2)
    mag_v1 = np.linalg.norm(v1)
    mag_v2 = np.linalg.norm(v2)
    if mag_v1 == 0 or mag_v2 == 0:
        return 0
    angle = math.acos(dot_prod / (mag_v1 * mag_v2))
    return np.degrees(angle)

def calculate_angular_distances(landmarks):
    """Calculate angular distances between specified facial features."""
    if landmarks is None:
        return None
    angles = {
        'eye_nose_eye': calculate_angle(landmarks[33], landmarks[168], landmarks[263]),  # Left eye -> Nose -> Right eye
        'nose_mouth_chin': calculate_angle(landmarks[168], landmarks[13], landmarks[152]),  # Nose -> Mouth -> Chin
        'left_eye_brow_eye': calculate_angle(landmarks[105], landmarks[168], landmarks[334])  # Left brow -> Nose -> Right brow
    }
    return angles

def calculate_distance(landmarks, indices):
    """Calculates the distance for the specified indices in the landmarks."""
    points = [landmarks[i] for i in indices]
    distances = [np.linalg.norm(np.array(points[i]) - np.array(points[i + 1])) for i in range(len(points) - 1)]
    return sum(distances)

def calculate_symmetry_ratios(original_landmarks, mirrored_landmarks):
    """Calculate symmetry ratios using both linear and angular measurements."""
    if original_landmarks is None or mirrored_landmarks is None:
        return None

```

```python
    # Define indices for different facial features
    left_indices = list(range(0, 234))
    right_indices = list(range(234, 468))
    forehead_indices = [10, 338, 297, 332, 284, 251, 389, 356, 168, 8, 107, 336]

    chin_to_ear_indices = left_indices + right_indices  # Chin to ear
    lip_corner_to_eye_and_ear_indices = left_indices + right_indices  # Lip corner to eye and ear
    nose_to_ear_indices = left_indices + right_indices  # Nose to ear

    # Calculate distances for each facial feature
    chin_to_ear_distance_original = calculate_distance(original_landmarks, chin_to_ear_indices)
    chin_to_ear_distance_mirrored = calculate_distance(mirrored_landmarks, chin_to_ear_indices)

    lip_corner_to_eye_and_ear_distance_original = calculate_distance(original_landmarks, lip_corner_to_eye_and_ear_indices)
    lip_corner_to_eye_and_ear_distance_mirrored = calculate_distance(mirrored_landmarks, lip_corner_to_eye_and_ear_indices)

    nose_to_ear_distance_original = calculate_distance(original_landmarks, nose_to_ear_indices)
    nose_to_ear_distance_mirrored = calculate_distance(mirrored_landmarks, nose_to_ear_indices)

    forehead_distance_original = calculate_distance(original_landmarks, forehead_indices)
    forehead_distance_mirrored = calculate_distance(mirrored_landmarks, forehead_indices)

    # chin_to_ear_distance_original = calculate_distance(original_landmarks, left_indices + right_indices)
    # chin_to_ear_distance_mirrored = calculate_distance(mirrored_landmarks, left_indices + right_indices)

    # Linear symmetry ratio for Chin to Ear distance
    # chin_to_ear_symmetry_ratio = chin_to_ear_distance_original / chin_to_ear_distance_mirrored
    chin_to_ear_symmetry_ratio = chin_to_ear_distance_original / chin_to_ear_distance_mirrored
    lip_corner_to_eye_and_ear_symmetry_ratio = lip_corner_to_eye_and_ear_distance_original / lip_corner_to_eye_and_ear_distance_mirrored
    nose_to_ear_symmetry_ratio = nose_to_ear_distance_original / nose_to_ear_distance_mirrored
```

```python
    forehead_symmetry_ratio = forehead_distance_original / forehead_distance_mirrored

    linear_symmetry_ratios = {}

    linear_symmetry_ratios["chin_to_ear_ratio"] = chin_to_ear_symmetry_ratio
    linear_symmetry_ratios["lip_corner_to_eye_ratio"] = lip_corner_to_eye_and_ear_symmetry_ratio
    linear_symmetry_ratios["nose_to_ear_ratio"] = nose_to_ear_symmetry_ratio
    linear_symmetry_ratios["forehead_symmetry_ratio"] = forehead_symmetry_ratio

    # Angular symmetry ratios
    original_angles = calculate_angular_distances(original_landmarks)
    mirrored_angles = calculate_angular_distances(mirrored_landmarks)
    angular_symmetry_ratios = {}
    for angle_name in original_angles:
        original_angle = original_angles[angle_name]
        mirrored_angle = mirrored_angles[angle_name]
        angular_symmetry_ratios[angle_name] = original_angle / mirrored_angle

    return linear_symmetry_ratios, angular_symmetry_ratios

    # return chin_to_ear_symmetry_ratio, angular_symmetry_ratios

def test_symmetry():
    original_image_path = "/content/output_face_2.jpg"
    original_image = cv2.imread(original_image_path)

    if original_image is None:
        print("Error: Unable to read the image.")
        return

    mirrored_image = cv2.flip(original_image, 1)

    original_landmarks = detect_landmarks(original_image)
    mirrored_landmarks = detect_landmarks(mirrored_image)

    symmetry_ratios = calculate_symmetry_ratios(original_landmarks, mirrored_landmarks)

    return original_image, mirrored_image, symmetry_ratios

# Run the symmetry test and display results
original_image, mirrored_image, symmetry_ratios = test_symmetry()
```

```python
if original_image is not None and mirrored_image is not None:
    linear_symmetry_ratios, angular_symmetry_ratios = symmetry_ratios

    # print("Linear Symmetry Percentages:")
    # print("Chin to Ear Symmetry Percentage: {:.2f}%".format((1 - linear_symmetry_ratio) * 100))

    print("\n Linear Symmetry Ratios:")
    for ratio_name, ratio in linear_symmetry_ratios.items():
        print(f"{ratio_name} : {ratio:.2f}")

    print("\nAngular Symmetry Ratios:")
    for angle_name, ratio in angular_symmetry_ratios.items():
        print(f"{angle_name} : {ratio:.2f}")

    # Display the images with Mediapy
    media.show_image(original_image, title="Original Image")
    media.show_image(mirrored_image, title="Mirrored Image")
else:
    print("Face not detected in one or both images.")

linear_symmetry_ratio = []
for key in linear_symmetry_ratios.keys():
  linear_symmetry_ratio.append(linear_symmetry_ratios[key])
print(f"Linear symmetry ratio: {np.mean(linear_symmetry_ratio)}")


angular_symmetry_ratio = []
for key in angular_symmetry_ratios.keys():
  angular_symmetry_ratio.append(angular_symmetry_ratios[key])
print(f"Angular symmetry ratio: {np.mean(angular_symmetry_ratio)}")
## calculation of median symmetry ratio and median symmetry distance
# Initialize MediaPipe Face Mesh
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=True, max_num_faces=1, min_detection_confidence=0.5)

def detect_landmarks(image):
    """
    Detects facial landmarks using MediaPipe Face Mesh.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        list: List of (x, y) tuples for facial landmarks.
    """
    results = face_mesh.process(cv2.cvtColor(image, cv2.
```

```
        COLOR_BGR2RGB))
    if not results.multi_face_landmarks:
        return None

    landmarks = []
    for landmark in results.multi_face_landmarks[0].landmark:
        x = int(landmark.x * image.shape[1])
        y = int(landmark.y * image.shape[0])
        landmarks.append((x, y))
    return landmarks

def calculate_facial_symmetry(image_path):
    """
    Calculates the overall facial symmetry of a face in an image.

    Parameters:
        image_path (str): The path to the image file.

    Returns:
        float: The facial symmetry score as a percentage (0% to 100%).
    """
    # Load the image
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Unable to read the image.")
        return None

    # Detect facial landmarks
    landmarks = detect_landmarks(image)
    if landmarks is None:
        print("Error: No face detected in the image.")
        return None

    # Calculate the facial midline using nose bridge and chin points
    midline_x = (landmarks[168][0] + landmarks[8][0]) / 2  # Nose bridge (168) and chin (8)
    face_width = abs(landmarks[234][0] - landmarks[454][0])  # Width between leftmost (234) and rightmost (454) points

    # Define symmetric point pairs using MediaPipe Face Mesh indices
    symmetry_pairs = [
        (234, 454), (93, 323), (132, 361),  # Jawline points
        (105, 334), (107, 336), (46, 276),  # Eyebrow points
        (33, 263), (159, 386), (145, 374),  # Eye points
        (78, 308), (191, 415), (13, 14)     # Mouth points
    ]

    symmetry_diffs_normalized = []
```

```
    symmetry_ratio_normalized = []

    for left_idx, right_idx in symmetry_pairs:
        left_point = landmarks[left_idx]
        right_point = landmarks[right_idx]

        # Calculate distances from midline
        left_dist = abs(left_point[0] - midline_x)
        right_dist = abs(right_point[0] - midline_x)

        # Absolute difference in distances
        symmetry_diff = abs(left_dist-right_dist)
        if right_dist == 0:
                right_dist = 0.001
        symmetry_ratio = left_dist/right_dist
        # Normalize the difference
        symmetry_diff_normalized = symmetry_diff / face_width
        symmetry_diffs_normalized.append(symmetry_diff_normalized
            )
        symmetry_ratio_normalized.append(symmetry_ratio)

    # Calculate overall symmetry score
    overall_symmetry_diff =  np.mean(symmetry_diffs_normalized)
    overall_symmetry_ratio = np.mean(symmetry_ratio_normalized)
    # symmetry_percentage = overall_symmetry * 100  # Convert to
        percentage

    return overall_symmetry_diff,overall_symmetry_ratio
image_path = "/content/output_face_2.jpg"
symmetry_diff,symmetry_ratio = calculate_facial_symmetry(
    image_path)

print(f"Median symmetry ratio: {symmetry_ratio}")
print(f"Median symmetry difference: {symmetry_diff}")
```