

# AI used on this project:

## 1. Feature Implementation: Olivetti-Style Webcam Integration

The initial phase involved determining the requisite image processing pipeline to transform a modern high-resolution image into a format statistically consistent with the classic Olivetti Faces dataset.

- **AI Model Used: GLM-4.7 (Reasoning)**

- **Prompt Given:**

"How can we transform a modern 1080p image to match the statistical characteristics of the Olivetti Faces dataset? What steps or processing pipeline should a machine learning system follow to achieve this, including resizing, alignment, normalization, and any other relevant transformations?"

- **Response:**

To transform a modern 1080p image to match the statistical characteristics of the Olivetti Faces dataset, you must bridge the gap between high-resolution color consumer photography and low-resolution grayscale scientific imaging.

The Olivetti AT&T dataset (Olivetti Research Laboratory, Cambridge, UK) has very distinct physical properties. It consists of 400 images (10 per subject) taken between 1992 and 1994.

Here is the processing pipeline a machine learning system should follow to achieve this transformation, categorized by the stage of processing.

### 1. Geometric Normalization (Alignment & Resizing)

The most obvious characteristic of the Olivetti dataset is the low resolution (

64×64 pixels).

The Pipeline:

1. Face Detection & Cropping: Use a robust face detector (like MTCNN, DLIB, or RetinaFace) on the 1080p image to identify the bounding box.
2. Tight Crop: Extract the face. Unlike modern datasets that include background and shoulders (context), Olivetti images are "tight" crops focusing mostly on the head.
3. Eye Alignment (Crucial): Olivetti faces are strictly frontal.
  - Detect landmarks for the left and right eyes.
  - Calculate the angle between the eyes.

- Rotate the image so the eyes are horizontally aligned. This corrects for head tilt (roll).
4. Resizing:
- Resize the aligned, cropped face to exactly 64×64 pixels.
  - *Note:* Do not use high-end "super-resolution" upscaling if you are coming from a smaller source, but since you are downscaling from 1080p, use Lanczos or Bicubic interpolation to preserve edges during the shrink process.

## 2. Photometric Transformation (Color & Contrast)

Modern 1080p images are typically sRGB color with high dynamic range. Olivetti faces are grayscale with uniform, "flat" lighting.

The Pipeline:

1. Grayscale Conversion:
  - Convert the RGB image to Grayscale. Standard weighted average ( $0.299R+0.587G+0.114B$ ) is usually sufficient.
  - *Statistical Nuance:* The variance of the dataset is entirely contained in luminance; removing chroma channels is the first step in matching the data covariance structure.
2. Background Normalization:
  - In the Olivetti dataset, the background is a uniform black (0 value).
  - Apply a mask to isolate the face. Set all pixels outside the facial region to 0 (Black). This removes the "modern" artifacts of walls, trees, or rooms found in 1080p photos.
3. Global Histogram Matching (The "Statistical" Step):
  - Simply converting to grayscale does not match the *contrast* or *brightness distribution*.
  - Calculate the Cumulative Distribution Function (CDF) of the standard Olivetti dataset.
  - Calculate the CDF of your downsampled 1080p image.
  - Apply a histogram matching transformation to map your image's pixel intensities to the Olivetti CDF. This ensures the shadows and highlights fall within the same statistical ranges as the target data.
4. Contrast Limited Adaptive Histogram Equalization (CLAHE):
  - The Olivetti images were taken under controlled lighting. Modern photos often have harsh shadows.
  - Apply CLAHE with a small grid size (e.g., 4×4) to normalize local contrast and smooth out lighting variations (like shadows on one side of the face).

## 3. Statistical Feature Smoothing (Simulating Sensor Noise)

Modern sensors are sharp. 1990s cameras had softer lenses and lower resolution sensors, resulting in a lack of high-frequency edge detail.

The Pipeline:

1. Gaussian Blur:
  - Apply a mild Gaussian blur (Kernel size ≈3×3 or 5×5) to the 64×64 image.
  - This simulates the Optical Transfer Function (OTF) of older camera lenses and prevents the "aliasing" artifacts that occur when sharp 1080p edges are aggressively downsampled.

## 4. Data Normalization (Numerical Scaling)

Before feeding the image into a machine learning model (like a classifier or PCA pipeline), the numerical values must match the scale typically used when training on Olivetti.

The Pipeline:

1. Flattening:
    - o The Olivetti dataset is often treated as vectors of dimension 4096 (64×64).
    - o Reshape the image from (64,64) to (4096,).
  2. Scaling:
    - o Olivetti faces are almost exclusively normalized to the [0,1] range (Unit Scaling).
    - o Divide all pixel values by 255
- 

**The information above was used to draft the implementation prompt for the coding agent.**

- **Agent Used: Gemini 3 Pro (High)**
- **Prompt Given:**

"As a lead software developer, you are tasked with implementing a webcam feature in a Streamlit Python application. The feature should: 1. Capture an image from the user's webcam. 2. Detect and extract any faces present in the captured image. 3. Preprocess the extracted face(s) to match the format of the Olivetti Faces dataset from `sklearn.datasets.fetch_olivetti_faces` (i.e., 64x64 grayscale images). Provide clean, modular Python code that integrates seamlessly into a Streamlit app, using libraries like OpenCV, haar, or any other necessary tools. Ensure the code handles edge cases, such as no faces detected or multiple faces, gracefully."

*After some refactoring the code to suit my codebase, code looks like this :*

```
camera_photo = st.camera_input("Take a photo of your face")

if camera_photo is not None:
    # Process the captured image
    st.subheader("Processing Pipeline")

    # Load image from camera
    pil_image = Image.open(camera_photo)
    img_array = np.array(pil_image)

    # === PREPROCESSING PIPELINE ===
    # Matching Olivetti: detect -> crop -> resize -> grayscale -> histogram
    equalization -> normalize

    # Step 1: Convert to grayscale
    if len(img_array.shape) == 3:
```

```

        gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
    else:
        gray = img_array

    # Step 2: Detect face using Haar Cascade
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    faces = face_cascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
    )

    haar_face_detected = len(faces) > 0

    if haar_face_detected:
        # Use the largest detected face
        faces = sorted(faces, key=lambda f: f[2] * f[3], reverse=True)
        x, y, w, h = faces[0]

        # Add padding (20%)
        padding = int(0.2 * max(w, h))
        x_start = max(0, x - padding)
        y_start = max(0, y - padding)
        x_end = min(gray.shape[1], x + w + padding)
        y_end = min(gray.shape[0], y + h + padding)

        # Crop the face region
        face_crop = gray[y_start:y_end, x_start:x_end]

        # Make it square
        fh, fw = face_crop.shape
        if fh > fw:
            diff = (fh - fw) // 2
            face_crop = face_crop[diff:diff+fw, :]
        elif fw > fh:
            diff = (fw - fh) // 2
            face_crop = face_crop[:, diff:diff+fh]

        cropped = face_crop
    else:
        # Fallback: center crop
        h_img, w_img = gray.shape

```

```

min_dim = min(h_img, w_img)
start_h = (h_img - min_dim) // 2
start_w = (w_img - min_dim) // 2
cropped = gray[start_h:start_h+min_dim, start_w:start_w+min_dim]

# Step 3: Resize to 64x64 (Olivetti face size)
resized = cv2.resize(cropped, (64, 64), interpolation=cv2.INTER_AREA)

# Step 4: CLAHE (Contrast Limited Adaptive Histogram Equalization)
# Better than regular histogram equalization for matching Olivetti
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_applied = clahe.apply(resized)

# Step 5: Normalize to [0, 1] range
normalized_raw = clahe_applied.astype(np.float64) / 255.0

# Step 6: Match Olivetti statistics
# Olivetti faces have specific mean/std characteristics
olivetti_mean = np.mean(X_train) # ~0.55 typically
olivetti_std = np.std(X_train) # ~0.23 typically

webcam_mean = np.mean(normalized_raw)
webcam_std = np.std(normalized_raw)

# Standardize to match Olivetti distribution
if webcam_std > 0:
    normalized = (normalized_raw - webcam_mean) / webcam_std * olivetti_std + olivetti_mean
else:
    normalized = normalized_raw

# Clip to valid range [0, 1]
normalized = np.clip(normalized, 0, 1)

# Flatten to vector
webcam_face = normalized.flatten()

# Show preprocessing pipeline
with st.expander("View Preprocessing Pipeline", expanded=False):
    cols_pipe = st.columns(6)

    with cols_pipe[0]:
        st.caption("1. Original")
        st.image(pil_image, use_container_width=True)

```

```

        with cols_pipe[1]:
            st.caption("2. Grayscale")
            st.image(gray, use_container_width=True)

        with cols_pipe[2]:
            st.caption("3. Face Crop")
            st.image(cropped, use_container_width=True)

        with cols_pipe[3]:
            st.caption("4. Resize 64x64")
            st.image(resized, use_container_width=True)

        with cols_pipe[4]:
            st.caption("5. CLAHE")
            st.image(clahe_applied, use_container_width=True)

        with cols_pipe[5]:
            st.caption("6. Normalized")
            # Show as grayscale image
            fig_norm, ax_norm = plt.subplots(figsize=(2, 2))
            ax_norm.imshow(normalized.reshape(64, 64), cmap='gray', vmin=0,
                           vmax=1)
            ax_norm.axis('off')
            st.pyplot(fig_norm)

        if haar_face_detected:
            st.success(f"Haar Cascade detected a face: {w}x{h} pixels")
        else:
            st.warning("No face detected by Haar Cascade. Using center
crop.")

        st.info(f"**Stats:** Olivetti mean={olivetti_mean:.3f},
std={olivetti_std:.3f} | Your image after matching: mean={np.mean(normalized):.3f},
std={np.std(normalized):.3f}")
    
```

## 2. Content Refinement and Restructuring

The website content, specifically the information presented on the frontend, was refined and restructured using the Gemini 3 Pro AI model to enhance readability and maintain a professional tone.