Real-Time Face Classification & Tracking System

1. Executive Summary

This project implements a lightweight, real-time computer vision pipeline designed to verify human faces in video streams. The system utilizes a hybrid approach: Haar Cascades for initial object proposal and my custom-built **Convolutional Neural Network (CNN)** for binary classification (Face vs. Non-Face). The model is optimized for edge deployment, containing approximately **150,000 parameters**, allowing for high frame rates on standard CPU hardware.

The dataset was synthesized by taking 5,000 images from the LFW dataset for the 'face' class and 1,000 images from various classes of the CIFAR dataset for the 'non-face' class. Some images from the CIFAR set contained human faces, which were removed to obtain 923 remaining images. Brightness variation and random rotation was used to get 4077 extra images using data augmentation. Hence 5,000 images from each class yielded a total of 10,000 images worth of training data.

---

2. Model Architecture Analysis

Source: CNN.ipynb & predict.py

The architecture is a custom sequential CNN designed to be "deep but narrow," minimizing parameter count while maintaining feature extraction capabilities.

Layer-by-Layer Breakdown

Layer	Configuration	Purpose
Input	(250, 250, 3)	Accepts high-resolution RGB crops.
Rescaling	1./255	**Crucial:** Normalizes pixel values from [0, 255] to [0, 1] internally. This ensures the model works even if raw image data is passed during inference.
Conv Block 1	16 filters, 3x3, padding='same'	Captures low-level features (edges, colors) without reducing spatial dimensions immediately.
Conv Block 2	32 filters, 3x3	Captures textures and simple shapes.

Conv Block 3	64 filters, 3x3	Detects composite features (eyes, nose structures).
Conv Block 4	64 filters, 3x3	High-level semantic feature extraction.
Flatten	N/A	Converts 3D feature maps into a 1D vector.
Dropout	Rate: 0.5	A medium-high dropout rate was chosen to prevent overfitting, forcing the model to learn robust features rather than memorizing the training set.
Dense (Hidden)	8 units, ReLU	**Information Bottleneck:** Compresses features into just 8 meaningful signals, significantly reducing parameter count.
Dense (Output)	1 unit, Linear	Outputs raw **logits**. Requires Sigmoid function during inference to convert to probability.

Key Design Choice: The use of BatchNormalization after every Conv2D layer (before Activation) ensures stable gradients and allows for faster convergence during training.

3. Training Methodology

Source: CNN.ipynb

- **Data Pipeline:**
  - Utilizes tf.keras.preprocessing.image_dataset_from_directory.
  - **Batch Size:** 64.
  - **Image Size:** 250x250 (RGB).
  - **Split:** Training (90%) and Validation (10%).
- **Compilation:**
  - **Optimizer:** Adam (learning_rate=0.0001).
  - **Loss Function:** BinaryCrossentropy(from_logits=True). This is numerically more stable than using a Sigmoid activation output layer during training.
- **Artifacts:**
  - The training process yielded a complete model file (FaceClassifier.keras / .h5) and a weights-only file (face_classifier.weights.h5), indicating a modular approach to saving/loading.

4. Inference Pipeline Analysis

Source: predict.py

The inference script is a robust, production-ready implementation that solves several common Computer Vision "Domain Shift" problems.

A. The Hybrid Pipeline

The script does not run the CNN on the entire frame. Instead, it uses a two-stage process:

1. **Stage 1 (Proposal):** Uses OpenCV's CascadeClassifier (Haar Cascade) to identify potential face regions. This is extremely fast but prone to false positives.
2. **Stage 2 (Verification):** The CNN verifies the crop. If probability < Threshold, it confirms the face; otherwise, it rejects it as background noise.

B. Critical Robustness Features

The predict.py file includes specific logic to handle real-world webcam data:

1. **Padding (Context Expansion):**
   ```python
   offset = int(w * 0.4) # Adds 40% margin
   y1, y2 = max(0, y-offset), ...
   ```
   - **Why:** Haar Cascades crop tightly around the eyes/mouth. The CNN was trained on LFW (which includes

hair/ears). This 40% padding aligns the live crop with the training distribution.

2. **Color Space Conversion:**
   ```Python
   img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   ```
   - **Why:** OpenCV captures BGR; the model expects RGB. Without this, skin tones would look blue, causing prediction failure.
3. **Threshold Logic (Inverted):**
   ```Python
   if probability < 0.85:
     label = "CONFIRMED FACE"
   ```
   - **Logic:** Since face was likely class index 0 during training, a *lower* probability means higher confidence it is a face. The script uses a lenient threshold (0.85) to ensure detection stability.
4. **UI/UX:**
   - Implements a full-screen display logic (cv2.WND_PROP_FULLSCREEN).
   - Uses color-coded feedback (Green for Face, Yellow for Scanning).

---

5. Conclusion

The analyzed files demonstrate a sophisticated understanding of the Deep Learning lifecycle. The project successfully transitions from a training notebook (CNN.ipynb) to a deployed application (predict.py).

Strengths:

- **Efficiency:** ~150k parameters allow for CPU-based real-time inference.
- **Robustness:** Handling of BGR/RGB conversion and ROI padding shows attention to detail regarding domain shift.
- **Stability:** Dropout (0.5) and Batch Normalization suggest a model that generalizes well to new faces.