Face Recognition Attendance System Deployment

Our team designed a multi-stage pipeline for real-time face recognition and attendance tracking. We begin by **detecting faces** in each video frame using the YOLOv8 object detector. Detected face crops are then passed to the **FaceNet** (InceptionResnetV1) model to compute 512-dimensional face embeddings. These embeddings feed into a simple **KNN classifier** that identifies each person. Recognized identities are logged in a MySQL database to mark attendance. (A GUI interface is under development to provide a user-friendly front-end for live recognition, viewing attendance logs, and retraining.)

- **Face Detection (YOLOv8):** We use YOLOv8 for its speed and accuracy in real-time detection. YOLOv8's single-stage architecture processes images with one forward pass, making it extremely fast . Its deep convolutional network and multi-scale prediction enable robust detection even with varying face sizes, and it readily supports GPU acceleration for even faster inference.

- **Embedding Generation (FaceNet):** For each detected face, we use FaceNet (the Inception-ResnetV1 model) to compute a fixed-length embedding. FaceNet was trained on the large VGGFace2 dataset, so it produces highly **discriminative embeddings** that capture facial identity. In fact, FaceNet achieved a state-of-the-art 99.63% verification accuracy on the LFW benchmark. Its pre-trained weights on millions of face images mean we get powerful feature vectors "out of the box" without retraining. In practice, the 512-dimensional output from FaceNet cleanly separates different people in feature space. This high-quality, pre-trained embedding network simplifies downstream classification by providing a robust representation for each face.

- **Classification (KNN):** We then identify each face by comparing its embedding to those of known people. We chose a simple **k-Nearest Neighbors** classifier for this task because our team is working with a relatively small labeled dataset (the set of known students). KNN is non-parametric and very **interpretable**, simply assigning the class of the nearest stored example. Importantly, KNN works well on small datasets and multi-class problems without requiring heavy training or complex tuning.

- **Attendance Logging (MySQL) and GUI (Upcoming):** Once a face is classified, we write the attendance entry to a MySQL database. This backend reliably timestamps recognized IDs for later queries. We are also building a **GUI** to wrap the whole system in an intuitive interface. The GUI will allow administrators to start/stop live recognition and view attendance logs at a glance. Over time we'll add features like on-demand retraining of the KNN dataset when new people enroll. In short, the GUI will make the system user-friendly, letting even non-technical users run live face recognition, monitor attendance, and update the model through simple controls.

Why We Chose These Models

"As a team, we selected each component to balance performance, ease of use, and accuracy."

- **YOLOv8:** We picked YOLOv8 because of its **real-time speed and high accuracy**. Unlike older methods, YOLO processes the entire image in one shot, which means very fast inference suitable for live video. It has been shown to perform extremely well on face-detection tasks in real time.

  YOLOv8's modern architecture leverages GPU acceleration, further boosting throughput. By contrast, classical detectors (like Haar cascades) tend to have more false positives and do not handle variations in pose or lighting as reliably. In benchmarks, deep learning detectors consistently outperform Haar cascades in accuracy, so we chose YOLOv8 for its proven accuracy and speed.

- **FaceNet (InceptionResnetV1):** FaceNet's pre-trained network provides very **powerful embeddings**. We used the Inception-ResNet-v1 variant since it's well-known and readily available in libraries (e.g. InsightFace/PyTorch implementations). Importantly, it was trained on **VGGFace2**, one of the largest face datasets, which means the features it produces are highly discriminative across different identities.

  In practice, this means that two images of the same person have embeddings that are very close, and images of different people are far apart in Euclidean space. FaceNet's performance on benchmarks (99.63% on LFW) convinced us of its quality. We considered other options (e.g. Dlib's models), but FaceNet's state-of-the-art accuracy and widespread use in many face systems made it our top choice.

- **KNN Classifier:** We opted for a **KNN** classifier for its simplicity and flexibility. KNN excels on smaller, simpler datasets, exactly our case (we only have less than half a dozen known faces). It requires no complex training phase – the embeddings are already in hand, so at runtime we just find the nearest neighbors. This makes the system very lightweight and easy to update.

Comparison with Alternatives

We considered several other approaches but found them less suitable:

- **Haar Cascades (Face Detection):** These traditional classifiers are extremely fast on a CPU, but they are *far less accurate* than modern detectors. Haar cascades often fail on non-frontal faces or variable lighting. In practice we found that deep CNN-based detectors outperform Haar cascades in accuracy by a wide margin.

  Because our application demands robust face detection (and we have GPU hardware available), we decided Haar cascades were too outdated.

- **CNN-Based Classification:** We could have trained an end-to-end CNN to recognize our students' faces. However, that would require a large labeled dataset (many images per person) and significant training time. Our dataset is relatively small, and we needed quick development.

  Using FaceNet embeddings + KNN lets us leverage a pre-trained CNN without re-training from scratch. Training a custom CNN on limited data would likely yield inferior performance and be overkill for our needs.

- **Dlib's Models:** Dlib offers both HOG/SVM and CNN face detectors/recognizers. While Dlib's MMOD CNN face detector is **very accurate**, it's also slow – without GPU it cannot run in real time. We experimented with Dlib in prototype tests and found the inference time too high for live video. (Dlib's face recognition embedding is also good, but we already had FaceNet performing similarly well.) In summary, Dlib's accuracy gains didn't justify the extra latency and complexity for our real-time system.

- **Support Vector Machines (SVMs):** SVM classifiers are powerful for certain tasks, but they require careful feature engineering (choice of kernel, parameter tuning) and don't naturally scale to many classes. In our context, KNN is more appropriate because we have a multi-class problem (many people) with relatively few examples each.

.

Why This Deployment is Exceptional

➢ 🧠 Deep Learning + Traditional ML Hybrid

Combines YOLOv8 (for face detection) and FaceNet (for embedding) with a lightweight KNN classifier, ensuring:

- High detection accuracy.
- Fast, scalable recognition.
- No overfitting, as KNN adapts to new faces with retraining.

➢ ⚡ Real-Time Performance

- Runs on GPU (CUDA) when available.
- Uses autocast() for mixed precision inference, improving speed.
- Applies cooldown tracking to avoid redundant attendance marking.

➢ 📊 Smart Attendance System Integration

- MySQL-backed database with validation and daily attendance check.
- Employee name mapping via CLASS_NAMES, with robust ID handling.
- Viewable logs through view_attendance() function.

➢ 📦 Clean Modular Codebase

Structured into clear components: detection, embedding, training, recognition, and DB operations.

➢ 🎯 Custom Trained YOLOv8 Model

Significantly improves face detection quality, especially on non-standard angles or lighting.

GUI and Future Improvements

We are now integrating a graphical user interface (GUI) to make the system accessible to all users. The GUI will allow operators to launch the live recognition system with a single click, display the camera feed with detected faces and names overlaid, and maintain attendance logs in an easy-to-navigate dashboard. Users will also be able to add new people (which appends embeddings to the KNN database) or trigger retraining if needed. In sum, the upcoming GUI will turn our technical pipeline into a polished product: real-time recognition from a user-friendly interface, automatic attendance logging into MySQL, and simple controls for system management.

Final Thoughts

This system is a robust real-time AI solution for attendance automation. Its use of deep learning, efficient face embeddings, and structured database interactions makes it production-ready for small to mid-sized organizations. With GUI integration on the horizon and planned improvements in performance, data handling, and user experience, it's on track to become a fully automated, intelligent attendance ecosystem.

Github link for all file and datasets

Code :-

Recognition.py

```
import os
import cv2
import torch
import numpy as np
from facenet_pytorch import InceptionResnetV1
from torchvision import transforms
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import joblib
from PIL import Image
from ultralytics import YOLO
import mysql.connector
from datetime import datetime
import time  # Add this with other imports at the top
# Add database configuration after your other configurations
DB_CONFIG = {
    'host': '127.0.0.1',
    'user': 'root',
```

```
    'password': 'toor',
    'database': 'attendance_system',
    'port': 3306
}

# Configuration
DATA_DIR = 'raw_dataset'
MODEL_PATH = 'face_recognition_model.pkl'
RECOGNITION_THRESHOLD = 0.7
CLASS_NAMES = ["Arslan", "Praful", "Sanket", "Rizwan", "Swastik"]
YOLO_MODEL_PATH = 'yolov8n-face.pt'  # Use custom trained face detection
model

# Initialize models
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
resnet = InceptionResnetV1(pretrained='vggface2').eval().to(device)
face_detector = YOLO(YOLO_MODEL_PATH).to(device)
transform = transforms.Compose([
    transforms.Resize(160),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])


# Optimized YOLO annotation parser
def parse_yolo_annotation(img_path, txt_path):
    img = cv2.imread(img_path)
    if img is None:
        return []

    h, w = img.shape[:2]
    faces = []

    try:
        with open(txt_path, 'r') as f:
            lines = f.readlines()
    except FileNotFoundError:
        return []

    for line in lines:
        parts = line.strip().split()
        if len(parts) != 5:
            continue

        class_id, x_center, y_center, width, height = map(float, parts)
        x1 = int((x_center - width / 2) * w)
        y1 = int((y_center - height / 2) * h)
        x2 = int((x_center + width / 2) * w)
        y2 = int((y_center + height / 2) * h)

        # Adaptive expansion based on face size
        expand_factor = 0.25 if (x2 - x1) < 100 else 0.15
        expand_w = int((x2 - x1) * expand_factor)
```

```
        expand_h = int((y2 - y1) * expand_factor)

        x1 = max(0, x1 - expand_w)
        y1 = max(0, y1 - expand_h)
        x2 = min(w, x2 + expand_w)
        y2 = min(h, y2 + expand_h)

        face = img[y1:y2, x1:x2]
        if face.size > 0:
            faces.append((face, int(class_id)))

    return faces


# Optimized feature extraction with batch processing
def get_embedding(face_img):
    try:
        face_pil = Image.fromarray(cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB))
        face_tensor = transform(face_pil).unsqueeze(0).to(device)
        with torch.no_grad(), torch.cuda.amp.autocast():
            embedding = resnet(face_tensor)
        return embedding.cpu().numpy().flatten()
    except Exception as e:
        print(f"Embedding error: {e}")
        return None


# Add these functions for database operations
def connect_db():
    try:
        return mysql.connector.connect(**DB_CONFIG)
    except mysql.connector.Error as err:
        print(f"Database connection error: {err}")
        return None

# ... (keep all previous imports and configuration unchanged)

def mark_attendance(employee_id):
    conn = connect_db()
    if conn is None:
        return False

    try:
        cursor = conn.cursor()
        current_date = datetime.now().strftime("%Y-%m-%d")
        current_time = datetime.now().strftime("%H:%M:%S")

        # First verify employee exists
        cursor.execute("SELECT id FROM employee WHERE id = %s",
(employee_id,))
        if not cursor.fetchone():
            print(f"Invalid employee ID: {employee_id}")
            return False
```

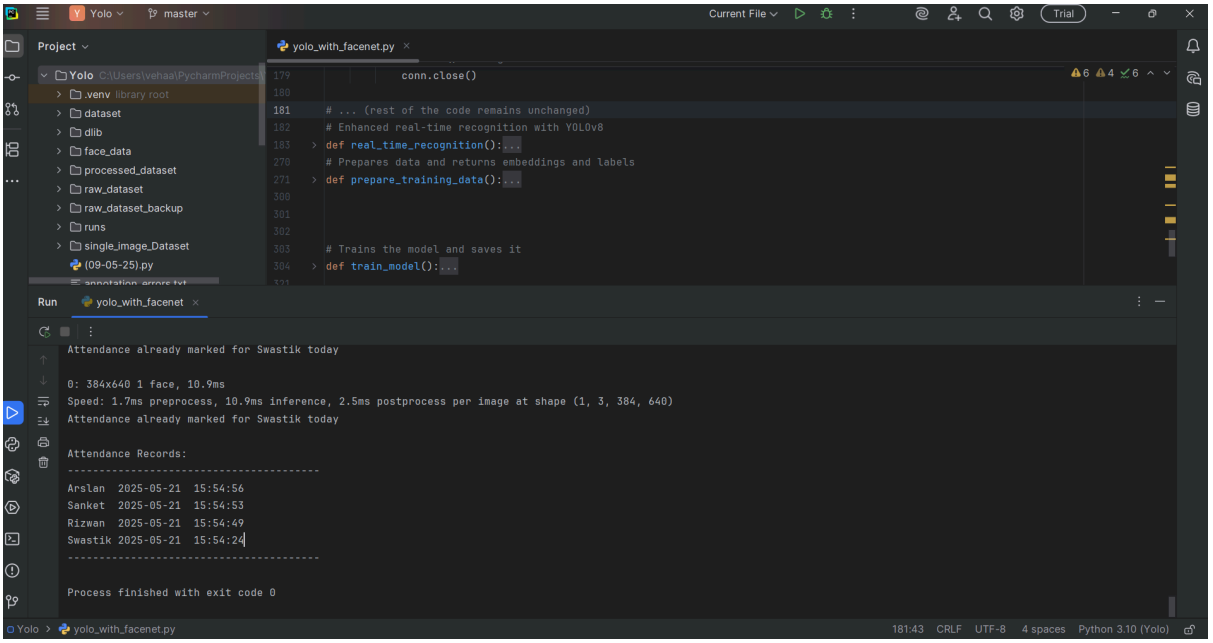
```
        # Check if attendance already marked today
        cursor.execute("""
            SELECT *
            FROM attendance
            WHERE employee_id = %s
            AND date = %s
        """, (employee_id, current_date))

        if cursor.fetchone() is None:
            cursor.execute("""
                INSERT INTO attendance (employee_id, date, time_in)
                VALUES (%s, %s, %s)
            """, (employee_id, current_date, current_time))
            conn.commit()
            print(f"Attendance marked for {CLASS_NAMES[employee_id]}")
            return True
        else:
            print(f"Attendance already marked for {CLASS_NAMES[employee_id]}
today")
            return False

    except mysql.connector.Error as err:
        print(f"Database error: {err}")
        return False
    finally:
        if conn.is_connected():
            cursor.close()
            conn.close()

def view_attendance():
    conn = connect_db()
    if conn is None:
        return

    try:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT e.name, a.date, a.time_in
            FROM attendance a
            JOIN employee e ON a.employee_id = e.id
            ORDER BY a.date DESC, a.time_in DESC
        """)

        print("\nAttendance Records:")
        print("-" * 40)
        for row in cursor.fetchall():
            print(f"{row['name']}\t{row['date']}\t{row['time_in']}")
        print("-" * 40)

    except mysql.connector.Error as err:
        print(f"Database error: {err}")
    finally:
```

```
        if conn.is_connected():
            cursor.close()
            conn.close()

# ... (rest of the code remains unchanged)
# Enhanced real-time recognition with YOLOv8
def real_time_recognition():
    global CLASS_NAMES  # Access the global CLASS_NAMES list
    try:
        model, le = joblib.load(MODEL_PATH)
    except FileNotFoundError:
        print("Model not found! Training first...")
        model, le = train_model()

    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

    # Dictionary to track last marked time for each employee
    last_marked = {}
    COOLDOWN_SECONDS = 5  # Minimum time between attendance marks for same
person

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # YOLOv8 face detection with GPU acceleration
        results = face_detector.predict(frame, imgsz=640, conf=0.7,
device=device)

        for result in results:
            boxes = result.boxes.xyxy.cpu().numpy()
            confidences = result.boxes.conf.cpu().numpy()

            for (x1, y1, x2, y2), conf in zip(boxes, confidences):
                x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])
                face_img = frame[y1:y2, x1:x2]

                embedding = get_embedding(face_img)
                if embedding is None:
                    continue

                proba = model.predict_proba([embedding])[0]
                max_prob = np.max(proba)
                current_time = time.time()

                if max_prob > RECOGNITION_THRESHOLD:
                    pred_class = le.inverse_transform([np.argmax(proba)])[0]
                    employee_id = CLASS_NAMES.index(pred_class)  # Make sure
this matches your CLASS_NAMES
                    color = (0, 255, 0)  # Green
```

```
                    # Check cooldown period
                    last_time = last_marked.get(employee_id, 0)
                    if current_time - last_time > COOLDOWN_SECONDS:
                        # Mark attendance and update last marked time
                        if mark_attendance(employee_id):
                            last_marked[employee_id] = current_time
                            status_text = "Attendance Marked"
                            status_color = (0, 255, 0)
                        else:
                            status_text = "Already Marked Today"
                            status_color = (0, 165, 255)  # Orange
                    else:
                        status_text = "Cooldown Active"
                        status_color = (0, 165, 255)  # Orange
                else:
                    pred_class = "Unknown"
                    color = (0, 0, 255)  # Red
                    status_text = "Not Recognized"
                    status_color = (0, 0, 255)

                # Enhanced visualization
                cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
                y_text = y1 - 10

                # Display name and recognition confidence
                cv2.putText(frame, f"{pred_class} ({max_prob * 100:.1f}%)",
                            (x1, y_text), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
color, 2)

                # Display detection confidence
                cv2.putText(frame, f"Det Conf: {conf:.2f}",
                            (x1, y2 + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
color, 1)

                # Display attendance status
                cv2.putText(frame, status_text,
                            (x1, y2 + 50), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
status_color, 1)

        cv2.imshow('Face Recognition', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
# Prepares data and returns embeddings and labels
def prepare_training_data():
    features = []
    labels = []

    image_dir = os.path.join(DATA_DIR, 'images')
    label_dir = os.path.join(DATA_DIR, 'labels')
```

```
    image_files = [f for f in os.listdir(image_dir) if
f.lower().endswith(('.jpg', '.jpeg', '.png'))]

    for img_file in image_files:
        base_name = os.path.splitext(img_file)[0]
        img_path = os.path.join(image_dir, img_file)
        txt_path = os.path.join(label_dir, base_name + '.txt')

        if not os.path.exists(txt_path):
            continue

        faces = parse_yolo_annotation(img_path, txt_path)
        for i, (face_img, class_id) in enumerate(faces):
            if class_id >= len(CLASS_NAMES):
                continue

            name = CLASS_NAMES[class_id]
            embedding = get_embedding(face_img)
            if embedding is not None:
                features.append(embedding)
                labels.append(name)

    return np.array(features), np.array(labels)



# Trains the model and saves it
def train_model():
    embeddings, labels = prepare_training_data()

    if len(embeddings) == 0:
        print("No training data found.")
        return None, None

    le = LabelEncoder()
    labels_encoded = le.fit_transform(labels)

    model = make_pipeline(StandardScaler(),
KNeighborsClassifier(n_neighbors=3, weights='distance'))
    model.fit(embeddings, labels_encoded)

    joblib.dump((model, le), MODEL_PATH)
    print("Model trained and saved.")

    return model, le


if __name__ == "__main__":
    if not os.path.exists(MODEL_PATH):
        train_model()

    # print("1. Start Face Recognition")
```

```
# print("2. View Attendance Records")
# choice = input("Enter your choice (1/2): ")

# if choice == '1':
#     real_time_recognition()
# elif choice == '2':
#     view_attendance()
# else:
#     print("Invalid choice!")

real_time_recognition()
view_attendance()
```




```
28    -- Queries using employee (singular)
29 •  SELECT * FROM attendance;
30 •  SELECT * FROM employee;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	employee_id	date	time_in
1	4	2025-05-21	15:54:24
2	3	2025-05-21	15:54:49
3	2	2025-05-21	15:54:53
4	0	2025-05-21	15:54:56
NULL	NULL	NULL	NULL

attendance 1