**Assignment No: 2**

**Facial Recognition using OpenCV**

Mrunal Talikoti 22311431

1. **Problem Statement**

Facial recognition using OpenCV and deep learning for binary classification.

1. **Objective**

* Understand the fundamentals of face detection and recognition.
* Preprocess facial data and extract facial embeddings.
* Implement a deep learning-based model to classify faces.
* Evaluate model performance using accuracy and other metrics.
* Visualize training progress and model performance.

1. **Software and Hardware Requirements**

**Operating System:** Windows / Linux / MacOS  
**Programming Environment:** Python 3.x, Jupyter Notebook, Anaconda, or Google Colab  
**Hardware:** Minimum 4GB RAM (CPU); GPU optional for faster training

**Libraries:**

* TensorFlow / Keras
* OpenCV
* Dlib
* face\_recognition
* NumPy
* Pandas
* Matplotlib
* Scikit-Learn

1. **Theory**

**Definition:**  
A facial recognition system identifies or verifies a person from digital images or video frames by detecting facial features and matching them against a database. In binary classification, the model distinguishes between two classes, such as “face” and “no face.”

**System Structure:**

1. **Face Detection Module:** Detects faces in images using Haar Cascades or deep learning models (e.g., SSD, YOLO).
2. **Feature Extraction Module:** Extracts unique facial features using Convolutional Neural Networks (CNNs).
3. **Classification Module:** Binary classifier (CNN or SVM) determines if a detected region contains a face.

**Activation Functions:** ReLU, Sigmoid, or SoftMax introduce non-linearity to learn complex patterns.

**Backpropagation:** Updates CNN weights by propagating the error between predicted and true labels backward through the network.

1. **Methodology**
2. **Data Collection:** Acquire a dataset containing face and non-face images (or specific expressions like happy and sad).
3. **Preprocessing:**
   * Detect faces using OpenCV.
   * Resize images to uniform dimensions and normalize pixel values.
4. **Model Architecture:** Build a CNN using Keras/TensorFlow for binary classification.
5. **Training:** Train the model using binary cross-entropy loss and monitor accuracy.
6. **Evaluation:** Test the model on unseen data to assess performance metrics.
7. **Prediction:** Use the trained model to classify new images.
8. **Advantages**

* **High Accuracy:** Deep learning models achieve precise face recognition.
* **Real-Time Processing:** OpenCV enables fast face detection and recognition.
* **Automation:** Useful for authentication, access control, and security systems.

1. **Limitations**

* **Data Requirements:** Requires large, diverse, high-quality datasets for training.
* **Lighting and Pose Variability:** Accuracy may drop with variations in illumination, expressions, poses, or occlusions.
* **Privacy Concerns:** Unauthorized use of facial data can raise ethical and legal issues.
* **Computational Complexity:** Deep learning models need significant computational resources.
* **Overfitting:** Small or biased datasets may lead to overfitting.
* **Adversarial Vulnerabilities:** Small image perturbations can fool the system.

1. **Applications**

* **Security and Surveillance:** Access control, monitoring high-security areas, and identifying individuals.
* **Biometric Authentication:** Smartphones, laptops, and other devices for user verification.
* **Law Enforcement:** Identifying suspects, locating missing persons, and tracking criminals.
* **Healthcare:** Monitoring patient conditions, analyzing emotional states, and diagnosing genetic disorders.
* **Retail and Marketing:** Personalized marketing and customer analysis.
* **Time and Attendance:** Automating employee attendance tracking.
* **Smart Cities:** Integrating facial recognition for traffic monitoring, crowd control, and public safety.

1. **Working / Algorithm**

**Step 1:** Install Libraries

* OpenCV for image processing and TensorFlow/Keras for deep learning.

**Step 2:** Load and Preprocess Dataset

* Load face images from a local directory.
* Resize images to 128×128 pixels and normalize pixel values.
* Handle non-image files and errors appropriately.

**Step 3:** Label Encoding

* One-hot encode labels for binary classification.

**Step 4:** Split Data

* Divide dataset into training (80%) and testing (20%) sets with stratification to maintain class balance.

**Step 5:** Define CNN Architecture

* Three convolutional layers with 32, 64, and 128 filters, each followed by ReLU activation and max pooling.
* Flatten the output, add a dense layer with 256 neurons, and include dropout for regularization.
* Output layer uses Softmax activation for binary classification.

**Step 6:** Compile the Model

* Optimizer: Adam
* Loss: Binary cross-entropy
* Metric: Accuracy

**Step 7:** Train the Model

* Train for 15 epochs with validation on the test set.

**Step 8:** Evaluate the Model

* Assess performance on the test data and compute test accuracy.

**Step 9:** Save the Model

* Save the trained CNN for future use.

**Step 10:** Predict on New Images

* Preprocess new images and predict the class using the trained model.

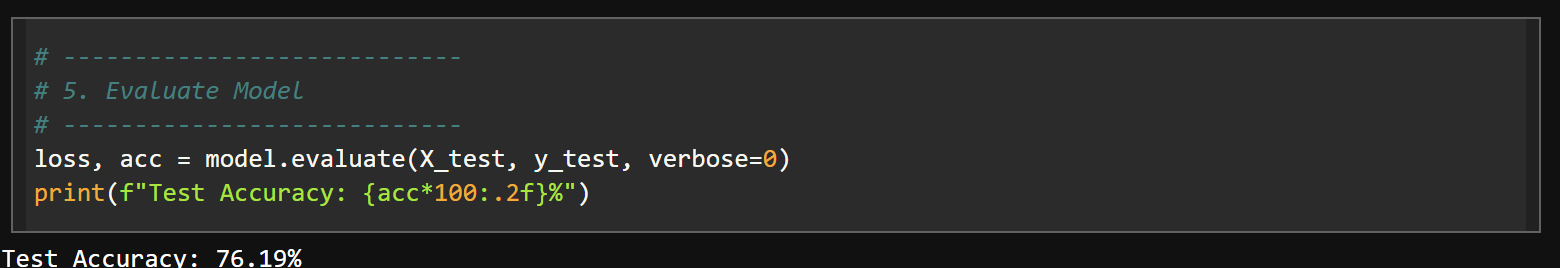
**Step 11:** Display Results

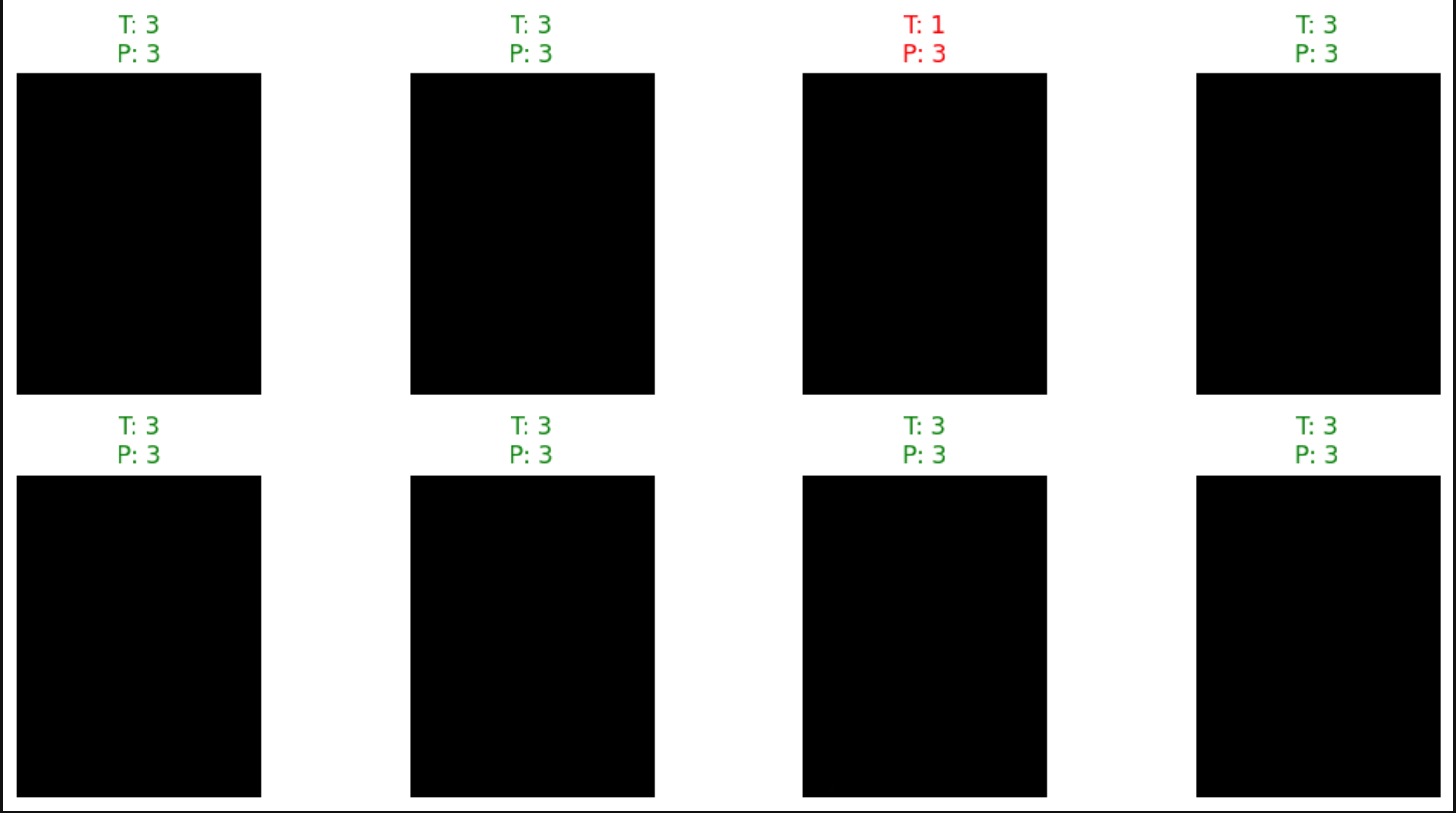
* Print test accuracy and display images with predicted classes.

1. **Conclusion**

OpenCV-based facial recognition provides a fast and effective approach for real-time face detection and classification. By leveraging classical computer vision techniques or deep learning models, it delivers high accuracy and scalability for applications such as security, authentication, and surveillance. Challenges like varying lighting, occlusions, and dataset quality must be addressed for optimal performance. With proper optimization and training, OpenCV offers a reliable solution for facial recognition tasks across industries.

1. **Output**

****

****