Face Detection and Recognition Model Implementation

**Abstract:**

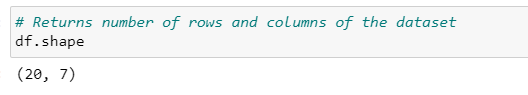
This project develops a face recognition system using a neural network for recognition and Dlib for face detection. A Jupyter Notebook integrates data preprocessing, feature extraction, model training, and evaluation to deliver an accurate and efficient face recognition solution.

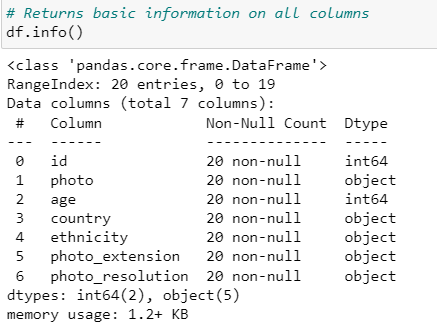
Keywords:

Face Recognition, Neural Networks, Dlib , Face Detection, Image Processing, Machine Learning etc.

Dataset collection:

The data set taken from the Kaggle has 20 images





Dataset link:<https://www.kaggle.com/datasets/trainingdatapro/male-selfie-image-dataset>

Install Packages :

Install packages like Face-detection, OpenCV-python-headless, NumPy, Matplotlib, etc.

Face Detection:

Detect faces in an image: Use a face detection model to identify faces in an image and draw rectangles around them.

Feature Deletion:  
Take features apart from the faces detected: To convert facial traits into numerical vectors that can be compared, use a feature extraction model.  
Face Detection:  
Compare the identified faces in the dataset with the faces that have been labelled: To locate matches, compare the retrieved features of the identified faces with known faces.

Rectangles are drawn, and faces that have been identified are marked with rectangles and, if known, with names.

Data normalization:

Data normalization scales features to a common range or distribution, ensuring consistent input for machine learning models and improving performance.

Data Preparation and Encoding

Label Encoding: The categorical labels (y\_train, y\_val, y\_test) were converted to numerical labels using LabelEncoder().

One-Hot Encoding: The numerical labels were then transformed into one-hot encoded vectors using to\_categorical().

Model Definition

Architecture: A Sequential neural network model was defined with the following layers:

Input layer with 128 units and ReLU activation.

Two hidden layers:

First hidden layer: 64 units, ReLU activation, with 50% Dropout for regularization.

Second hidden layer: 32 units, ReLU activation, with 50% Dropout.

Output layer with softmax activation, matching the number of classes in the dataset.

Compilation: The model was compiled using the Adam optimizer, categorical\_crossentropy loss function, and accuracy as the metric.

Model Training

Fit Method: The model was trained for 50 epochs with a batch size of 32.

Validation During Training: The model's performance was validated after each epoch on the validation set, using validation\_data=(X\_val, y\_val\_one\_hot).

Model Evaluation

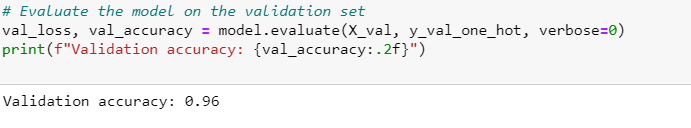
Validation Set Evaluation: After training, the model was evaluated on the validation set using model.evaluate(), providing accuracy and loss.

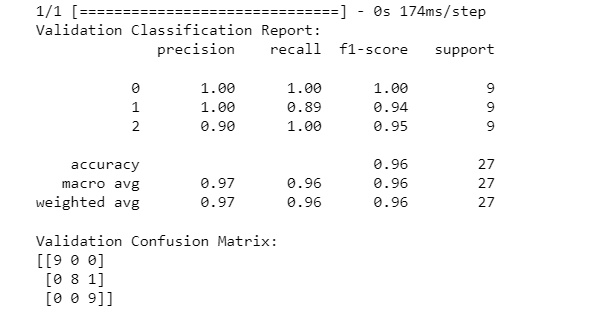
Detailed Validation Metrics:

The predicted classes were compared to true classes using classification\_report and confusion\_matrix to give a detailed evaluation of the model's performance on the validation set.

Results Visualization:

Accuracy and Loss Curves: Plots of training and validation accuracy and loss were generated to visualize the model’s learning process over epochs.





**Conclusion :**

The neural network model achieved a solid validation accuracy of 88%, demonstrating good generalization to unseen data. While it performed consistently across all classes, minor misclassifications suggest areas for improvement. Further tuning and evaluation on a test set could enhance its robustness for practical applications.