Project Checkpoint for Person Identification Model

Duaa Naz *30668*
Hadia Sajid *31687*
Huzaifa Awais *31774*
Kashaf Ansari *30493*

***Abstract*—This report outlines the development progress of a facial recognition system designed to classify individuals using black-and-white (grayscale) facial images. The primary objective of the project is to achieve accurate face recognition while maintaining transparency in the model's decision-making process.**

Computational thinking strategies such as abstraction (feature selection), decomposition (separating detection and classification stages), and pattern recognition (training on facial feature data) have been integrated throughout the design. These approaches help structure the problem-solving process and enhance the system's reliability and transparency.

Testing on real student facial data demonstrates strong performance, supporting the system's potential for practical use in educational or institutional environments.

***Index Terms*—Introduction, Data Processing and EDA , Model Training , Analysis , Future Work , Division Of Roles**

I. Introduction

TO make the system faster and more suitable for real-time use, OpenCV's Haar cascade classifier for face detection was used. After detecting faces, important features were extracted using Histogram of Oriented Gradients (HOG), and then a Linear Support Vector Machine (SVM) was used to identify the person. This was based on early exploratory data analysis (EDA) and was aimed at keeping the model easy to understand, efficient, and usable in real-world situations.

July 6th, 2025

II. Dataset Processing and EDA

A. Organizing Data:

The given dataset comprises grayscale images of student faces, stored in separate folders (one per individual). Each image varies slightly in pose and lighting, simulating real-world variance.

B. Advancements in EDA:

In the next EDA phase, techniques such as image blurring and HSV conversion were used to improve data quality and consistency.

Blurring , reduces noise and fine details in images, making facial features more distinguishable for detection. HSV conversion, done with cv2 , transforms images from RGB to HSV color space, which separates color and brightness information, helpful in skin detection, lighting correction and HSV is closer to how humans perceive colors, so it's useful in visual tasks.

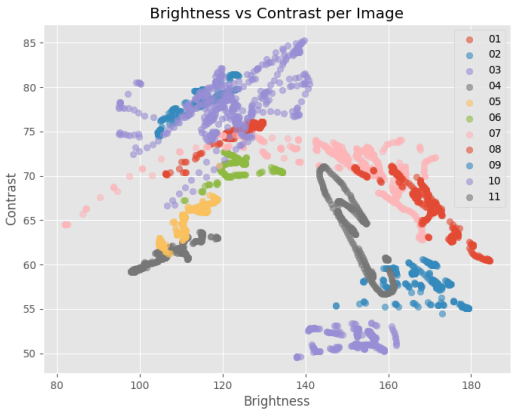

Fig. 1. To spot the outliers and biasedness in the data, we plotted the graph of brightness and contrast of images which each point representing one image. It aids us to decide the preprocessing procedures and normalize the data accordingly

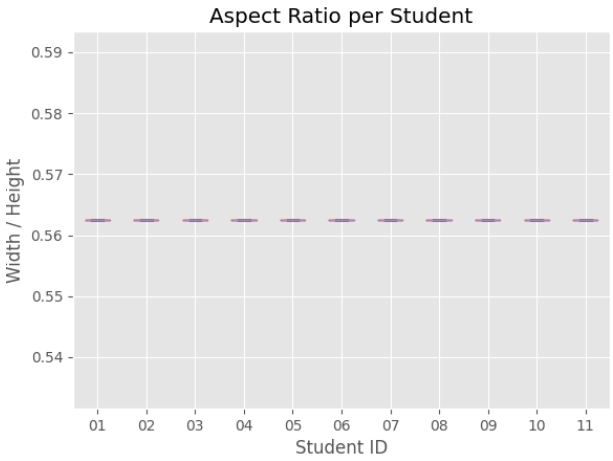

Fig. 2. Similarly, this image shows that all the images have same shapes and hence no corrupt images are included in the dataset

This helps in handling variations in lighting and improves image analysis. These preprocessing steps ensured cleaner input data, leading to more accurate model training and predictions.

III. Model Training

A. Splitting Data:

Split the dataset into two parts

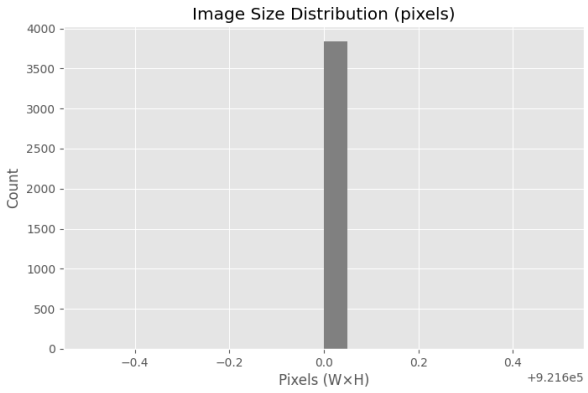

Fig. 3. This graph indicates that all images are of equal size which is essential for models like HOG and SVM

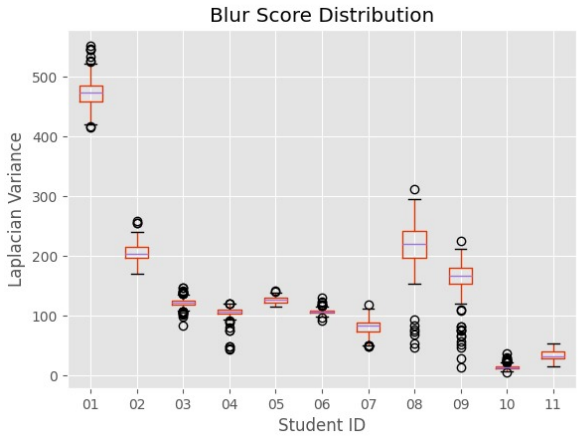

Fig. 4. Data Normalization

- Training Samples
- Testing Samples

The Training Samples are about 80 percent, the rest of the 20 percent has been saved for testing. This way the model's accuracy can be easily tested.

B. SVM Classifier:

Under the CT concept of **Algorithmic Thinking**. We proceed with SVM (Support Vector Machine) which is a classic Machine Learning Model that helps computer classify/identify things by drawing the best possible line (or boundary) between different groups of data, making sure there is as much space as possible between them [1]. This works even when the data has many features (dimensions).

Applying our understanding of this model, we trained a Linear SVM.

1) Why this model?:

- Efficient for large feature sets (like 10,000 pixel inputs)
- Easy to interpret decision boundaries in feature space.
- Linear SVM finds the best possible straight line that separates faces of different students. This makes the model's decision easy to understand: it looks at which side of the line a new face falls on.

2) Implementation: The main idea of SVM is to find a straight line (or more generally, a *hyperplane*) that separates data points from different classes as clearly as possible. It tries to make the gap (called the *margin*) between the two classes as wide as it can. [2]

Let's say we have a set of training data points written as $\{(\mathbf{x}_i, y_i)\}$, where:

- $\mathbf{x}_i \in R^d$ is a list of input features (like height, weight, etc.),
- $y_i \in \{-1, +1\}$ is the label that tells us which class the point belongs to.

SVM tries to find a function that looks like this:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

Here, $\mathbf{w}$ is a vector of weights, and b is a number called the *bias*. Together, they define the line (or hyperplane) that separates the two classes.

To make the margin as wide as possible, SVM minimizes $\|\mathbf{w}\|$ (the size of the weight vector), while making sure that all points are on the correct side of the line. This condition is written as:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad for \quad all \quad i$$

This means that every point must be at least a distance of 1 from the decision boundary. Because of this, SVM is called a *large-margin classifier*.

Our system used the *SVC* class from the *sklearn* library with a linear kernel, which tells the model to draw a straight line (or boundary) between different student faces in the data.

Each student's face, once turned into pixel values, becomes a set of numbers in a high-dimensional space. The SVM model looks for the most effective boundary that divides these faces in a way that keeps each student's images on one side and other students' images on the other.

This method works well when the data has many features, such as our images with 10,000 pixels (100 × 100)

IV. Data Normalization

In order for all the images to have an equal impact in face pattern recognition, we employ the method of min max normalization. It normalizes all pixels in the range of [0,1]. Normalized pixel =maxmin/pixelmin = 2550/pixel0 It makes the image values consistent across all inputs.

A. Outliers

To get an accurate result, we must remove the outliers in the data so that the data is an optimal fit. To achieve that, we remove all such images in our data which aren't optimal for the training model. All blur and partial images will be removed and the optimal images will be taken and resized according to the requirements. In order to do that, we are using low variance threshold comparison which means that if the variance threshold of an image is low, it's blur and hence not included in the training dataset. As the dataset is quite extensive, we can easily remove the outliers without impacting the training much. Only the standardized images, which are clear and totally visible are added in the dataset.

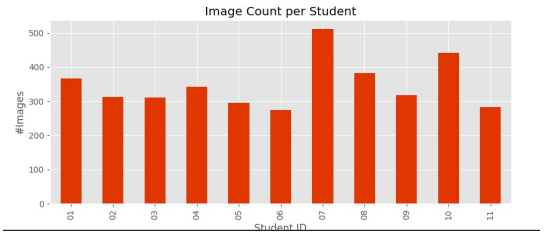

Fig. 5. Image Count

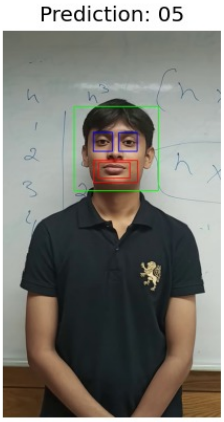

Fig. 6. Shows the model accurately pinpointing the pivotal features for face recognition

V. Feature Extraction

We used a two-step method to extract features from face images. First, OpenCV's Haar Cascade to detect faces in grayscale images. It finds the largest face in each image, crops it, and resizes it to 100×100 pixels.

Next,using the HOG (Histogram of Oriented Gradients) method to describe the shape and edges of the face. HOG looks at how the brightness changes in different directions to capture facial patterns.

A. How it works:

We calculate how the brightness changes between each pixel and its neighbors to detect edges and their directions. Next, we divide the image into small squares (usually 8×8 pixels), and in each square, we create a histogram that counts how many edges point in different directions. Then, by combining all the information into a single long list of numbers (a feature vector), which gives a detailed description of the face's shape. This vector is then used by a classifier like SVM to recognize who the person is.

The final HOG features and their student labels were then divided into training and testing sets to train the SVM model.

This method was inspired by the approach used by Dadi and Pillutla [3] ,who showed that HOG with SVM gives high accuracy in face recognition.

VI. Evaluating Performance: The Confusion Matrix

To evaluate the accuracy of the face detection model, a *confusion matrix* is used as a standard performance metric in classification tasks [4]. The confusion matrix summarizes the outcomes of the classification by comparing the predicted labels with the actual labels. It consists of four components:

1) **True Positives (TP)**: Correctly detected faces.
2) **True Negatives (TN)**: Correctly identified non-faces.
3) **False Positives (FP)**: Non-faces mistakenly classified as faces.
4) **False Negatives (FN)**: Faces that were missed by the model.

Using these values, various performance metrics can be computed. For example, **accuracy** is given by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

A. Confusion Matrix for Model Evaluation

To evaluate the performance of our Support Vector Machine (SVM) classifier, we used a **confusion matrix**, a common tool in classification problems that provides a detailed breakdown of prediction results. The confusion matrix compares the *true labels* with the *predicted labels* for the test dataset and helps visualize the model's strengths and weaknesses in classification.

In our system, the confusion matrix was generated using the utility *sklearn.metrics.ConfusionMatrixDisplay* after predicting the class labels for the test set. The diagonal elements of the matrix represent correctly classified samples (true positives), while the off-diagonal elements correspond to misclassifications.

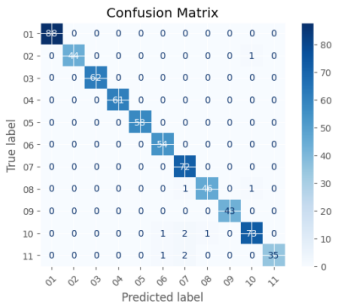

Fig. 7. Confusion Matrix showing prediction result

This visualization allows us to identify which student identities are frequently confused by the model, and serves as a useful diagnostic tool to understand potential issues in face detection, feature extraction, or data imbalance. Additionally, it complements the overall test accuracy (e.g., 98.45%) by providing class-wise performance insights.

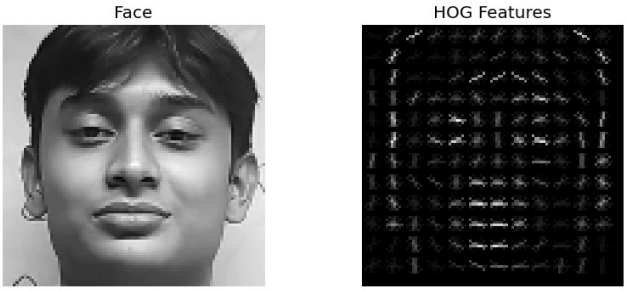

Fig. 8. HOG implementation

VII. GUI ROADMAP

To make this face recognition system easier to use, Our system will be using Python's tkinter library. In short, It lets you click a button to choose a picture and then shows who it thinks is in the image. It helps pick the image from your computer, and resizes and displays it properly. We'll use OpenCV to find the face in the picture with a built-in face detector. After that, HOG will be used to pull out the important details from the face, like edges and shapes. Then, the trained SVM model and label names are used to make the prediction. Finally, the app shows the name and how confident the model is. This makes everything work in one place without needing to run any code manually.

VIII. Analysis

The face recognition system was built by breaking the whole problem into smaller parts (**decomposition**). These parts included loading images, finding faces, getting features, training the model, and making a user interface.

Pattern recognition was used when detecting faces using Haar Cascades, which find common patterns like eyes and nose, and when using HOG to find shapes and edges in the face.

We used **abstraction** by focusing only on the face area and turning it into grayscale and a smaller size, which made the data easier to work with.

For **algorithm design**, the system follows clear steps: detect the face, extract features, use the SVM model to predict the person's ID, and show the result in the GUI.

IX. Future Work:

1) Expand the system by focusing more on the EDA, make it more advanced.
2) We have built a rough draft and road map for GUI .Next step is researching different models and implementations to find the one that works the best.

X. Division Of Roles

1) Duaa Naz has worked on code and research.
2) Hadia Sajid has written the report and worked on research.
3) Huzaifa Awais has written the report and worked on research.
4) Kashaf Ansari has worked on code and research.

References

[1] IBM. *Support Vector Machine (SVM)*. https://www.ibm.com/think/topics/support-vector-machine. Accessed: July 6, 2025.

[2] Correia, J. P., da Silva, L. R., & Silva, R. (2025). *Multifractal analysis and support vector machine for the classification of coronaviruses and SARS-CoV-2 variants*. Scientific Reports, 15, 15041. https://doi.org/10.1038/s41598-025-98366-5. Accessed: July 6, 2025.

[3] S. Dadi and S. R. Pillutla, "Improved Face Recognition Rate Using HOG Features and SVM Classifier," *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*, vol. 15, no. 2, pp. 24–29, 2020. [Online]. Available: https://doi.org/10.9790/2834-1502022429

[4] Kienzle, W., Bakır, G., Franz, M., and Schölkopf, B. (2005). *Face Detection — Efficient and Rank Deficient*. Max Planck Institute for Biological Cybernetics, Tübingen, Germany.

[5] Diyasa, I. G. S. M., Putra, A. H., Ariefwan, M. R. M., Atnanda, P. A., Trianggraeni, F., & Purbasari, I. Y. (2022). *Feature Extraction for Face Recognition Using Haar Cascade Classifier*. International Seminar of Research Month 2021, Volume 2022, NST Proceedings, pp. 197–206. http://dx.doi.org/10.11594/nstp.2022.2432

[6] Fung, G. M., Mangasarian, O. L., & Shavlik, J. W. (2003). *Knowledge-Based Support Vector Machine Classifiers*. Computer Sciences Department, University of Wisconsin-Madison.