A black text on a white background

Description automatically generated

Bachelor of Engineering Honours in Robotics and Artificial Intelligence

**6FTC2057 Visual and Spoken Interfaces**

Report Title:

Visual Interfaces Report

|  |  |
| --- | --- |
| Name | Student ID |
| Jawad Bin Joha | UH Student ID: 23029411  PSB Student ID: 530BIYKP |

Contents

[Abstract 1](#_Toc183349538)

[Introduction 2](#_Toc183349539)

[Design 3](#_Toc183349540)

[Development 4](#_Toc183349541)

[Facial Analysis 4](#_Toc183349542)

[Gesture Recognition 4](#_Toc183349543)

[Real-Time Interaction 4](#_Toc183349544)

[Results 5](#_Toc183349545)

[Assessment 6](#_Toc183349546)

[Analysis 7](#_Toc183349547)

[Conclusion 8](#_Toc183349548)

[References A](#_Toc183349549)

[Appendix B](#_Toc183349550)

# Abstract

The following report details the development of an interactive visual interface for real-time facial and gesture analysis using MATLAB. The project combines facial recognition and finger gesture classification functionalities, using a fine-tuned AlexNet model for a more robust and efficient performance. Face detection is implemented using the Viola-Jones algorithm, and both tasks are integrated into a real-time interface capable of processing webcam input. The face recognition and the gesture recognition model achieved 97.30% test accuracy and 100% training accuracy respectively using a custom image dataset. This system serves as an example for the potential utilisation of deep learning and real-time image processing in interactive human-computer interfaces.

# Introduction

Facial and gesture recognition have emerged as important technologies in countless areas such as human-computer interaction, surveillance, and accessibility utilities. These systems empower interfaces to understand and respond to user actions and requests, thereby producing more intuitive and interactive experiences. An example would be the personalized authentication measures facial recognition can provide in security systems, while gesture recognition can enable contactless interfacing with devices. Recent advancements in deep learning have made it possible to develop accurate and efficient systems for these tasks [1][2].

In this project, an interactive visual interface capable of analysing facial features and hand gestures in real time was designed an implemented. The system leverages the deep learning capabilities, specifically a pre-trained AlexNet model, to perform classification. Utilising transfer learning enabled the system to customize pre-trained models for specific tasks which greatly reduced the need for large-scale datasets and extended training durations [6][8]. MATLAB was the platform chosen for this project due to its powerful deep learning and computer vision toolboxes, which facilitated the implementation of real-time processing pipelines [3][4].

The facial recognition component of the system revolved around on detecting and identifying faces in live video streams. It used the Viola-Jones algorithm for face detection, which has been widely used in computer vision due to its efficiency and simplicity [5]. After detection, the system crops and preprocesses the detected faces, passing them through a fine-tuned AlexNet model for classification. This modular design ensures flexibility, allowing the system to be easily extended for additional tasks such as facial expression analysis.

The gesture recognition component was developed to classify hand gestures representing numbers (fingers) from 1 to 5. This component uses a custom dataset of hand images, and like the facial recognition system, it relies on a fine-tuned AlexNet model. This functionality demonstrates how gesture recognition can be applied in scenarios like controlling devices or interpreting sign language. Together, the facial and gesture analysis components form an interactive system capable of processing webcam input and providing real-time feedback.

# Design

The system comprises of three main components: facial analysis, gesture classification, and real-time interaction. Each component was implemented as a solitary module, where the facial recognition model and gesture recognition model were generated using 2 separate scripts, titled SimpleFaceRecognition\_Model\_Trainer.m and Gesturing\_Model\_Trainer.m respectively, and called into a main script, titled Visual\_Interfaces\_Main\_Script.m, combining their separate functions, allowing for easy integration and future extensibility. This modularity also makes it possible to replace individual components with alternative methods or models, depending on the application requirements.

Facial analysis was designed to perform both detection and recognition of individuals and expressions. The detection stage identifies faces in video frames, while the recognition stage classifies these faces into predefined categories. By separating these stages, the system ensures that only relevant regions of the image are processed by the recognition model, reducing computational overhead. The Viola-Jones algorithm was chosen for face detection due to its ability to process video frames quickly, making it suitable for real-time applications [5].

Gesture recognition was designed to classify static hand gestures representing the numbers 1 to 5. A library of images was collected, showcasing hands with varying positions and orientations. These images were organized into five categories corresponding to the number of fingers visible in each gesture. The AlexNet model was fine-tuned to classify these categories, providing a robust solution for gesture recognition tasks. The use of a pre-trained model significantly reduced the time required for training while maintaining high accuracy [6][7].

Real-time interaction was centre-stage to the design of the system. Processing video frames from a webcam in real time, the interface uses bounding boxes and text overlays to provide instant response to the user’s presence and gestures. This interactive approach enhances usability and demonstrates the practical applications of the system. MATLAB’s built-in visualization functions, such as insertText and insertShape, were used to annotate the video feed, ensuring that the interface was both informative and user-friendly.

# Development

## Facial Analysis

Facial analysis in this project begins with detection. The Viola-Jones algorithm, implemented using MATLAB's vision.CascadeObjectDetector, was selected for its computational efficiency and simplicity. This algorithm uses Haar-like features to detect faces in images, making it suitable for real-time applications where speed is critical [5]. Detected faces are cropped and resized to a consistent size of 227x227 pixels, which is required by the AlexNet model for recognition.

The recognition stage of facial analysis uses a fine-tuned AlexNet model. The final fully connected and classification layers of AlexNet were replaced to match the number of unique face classes in the dataset. This modification ensures that the model can accurately classify the detected faces. Training was conducted on a labeled dataset of cropped facial images with 4 differing expressions obtained from the same individual, Jawad\_Expressionless, Jawad\_Smiling, Jawad\_Scowling and Jawad\_Shocked, using transfer learning to leverage AlexNet's pre-trained feature extraction capabilities. This approach significantly reduced training time while achieving high accuracy [6][8].

## Gesture Recognition

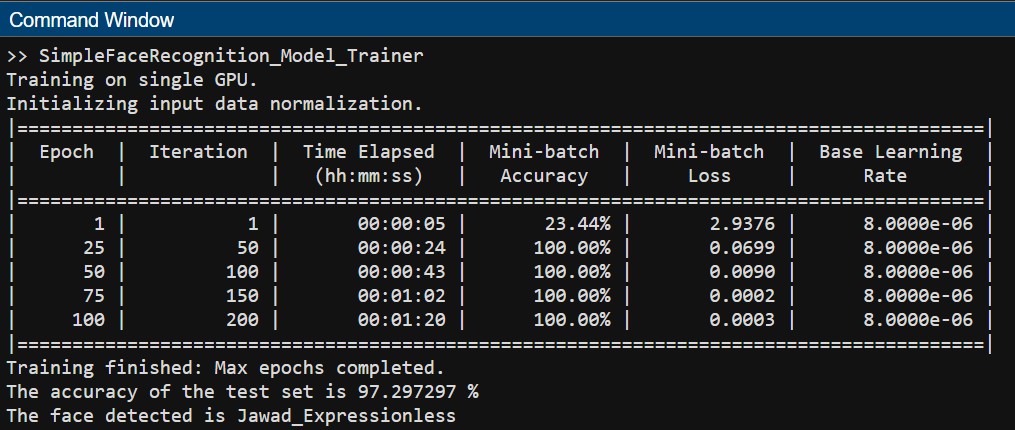
Gesture recognition was achieved by utilising a library of hand images that was categorised in 5 folders, 1, 2, 3, 4 and 5, to represent the number of fingers being held up. Data augmentation techniques were then applied to enhance the variability of the dataset, ensuring that the model generalized to different hand positions and orientations.

The AlexNet model was fine-tuned for this classification task by modifying its final layers to output five classes. The training process used the RMSprop optimizer, which is effective for image classification tasks due to its adaptive learning rate adjustments [7]. Training was conducted for 200 epochs, and the model achieved 100% training accuracy, indicating that it successfully learned the features of the hand gestures.

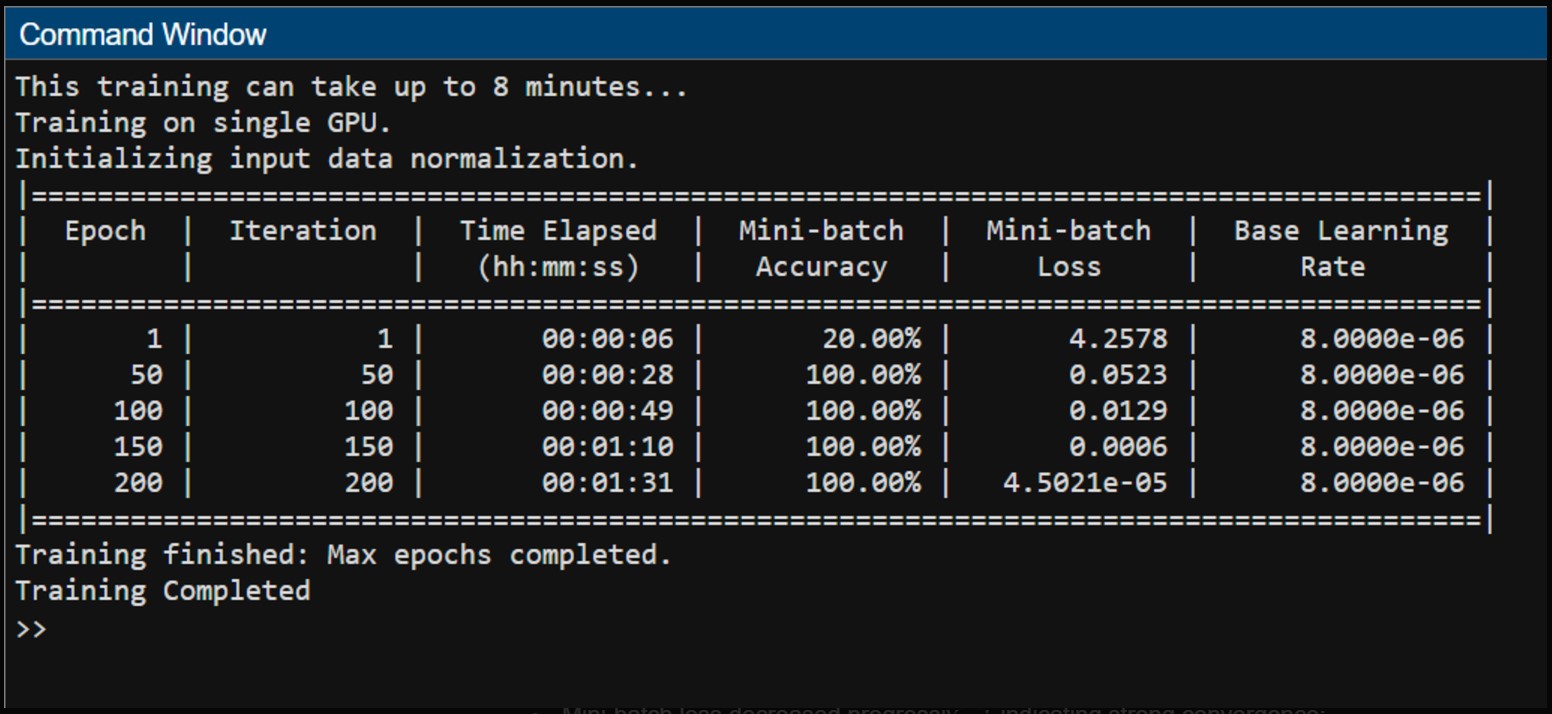
## Real-Time Interaction

The real-time interaction module integrates facial analysis and gesture recognition into a single interface in the main script. The system captures video frames from a webcam and processes them sequentially. Faces are detected, cropped, and passed to the facial recognition model, while gestures are identified within a predefined region of interest (ROI). The results are displayed on the video feed using bounding boxes and text annotations. MATLAB’s visualization functions were instrumental in creating the interactive interface. The insertShape function was used to draw bounding boxes around detected faces and gestures, while the insertText function displayed the classification results. This approach ensures that the system provides immediate feedback, making it intuitive and easy to use.

# Results



The facial recognition model demonstrated exceptional performance during training and evaluation. During the training phase, the model achieved 100% accuracy after 25 epochs, with the loss steadily decreasing and converging to 0.0003 by the end of the 100th epoch. This indicates that the model successfully learned the features of the training dataset without overfitting. On the test set, the model achieved an accuracy of 97.30%, confirming its ability to generalize to unseen data. An example test case successfully identified a face as “Jawad\_Expressionless,” showcasing the model’s effectiveness in practical applications. The combination of the Viola-Jones algorithm for face detection and the fine-tuned AlexNet model for recognition proved to be an effective strategy. The real time application of the model proved that the libraries used allowed for the recognition of the subjects expressions as well.



The gesture recognition model, trained to classify hand gestures representing the numbers 1 to 5, also demonstrated excellent results. It achieved 100% accuracy on the training dataset after 200 epochs, with the loss converging to 4.5021e-05. This low loss value indicates that the model accurately learned to distinguish between the five gesture classes. The use of a custom library of images featuring different hand positions and orientations contributed to the model’s robustness. The classification accuracy and strong convergence suggest that the gesture recognition model can be effectively used in applications requiring touchless interaction, such as counting, device control, or interpreting sign language.

Real-time performance testing, as displayed in the demonstaration video revealed that the system could process webcam input with tolerable latency. The system successfully detected and recognized faces and expressions, providing accurate predictions displayed on the video feed. Similarly, the gesture recognition component accurately identified and counted hand gestures within the defined region of interest (ROI). The efficient real-time interaction demonstrates the system’s readiness for practical deployment in scenarios such as classroom settings, where finger counting or gesture-based communication might be required.

The performance of both models highlights the effectiveness of using transfer learning with pre-trained AlexNet architectures. Despite the relatively small size of the training datasets, the models achieved high accuracy and robust generalization. This suggests that the feature extraction capabilities of AlexNet, combined with fine-tuning for specific tasks, are well-suited for applications like facial and gesture analysis. However, further testing on larger and more diverse datasets would be necessary to confirm the system’s performance in more challenging environments, such as varying lighting conditions or diverse user demographics.

# Assessment

The facial recognition model’s evaluation was carried out based on its accuracy, usability, and real-time performance. The 97.30% test accuracy achieved by the model spotlights its ability to generalize unseen data, which is crucial for practical applications. Similarly, the gesture recognition model's 100% training accuracy showcases its efficiency in identifying hand symbols. Both models performed well in their respective domains, achieving results that align with the expectations of modern deep learning systems.

Real-time performance is a critical aspect of interactive systems, and this project met the requirements for low-latency operation. The system processed video frames from a webcam at a rate of approximately 1.5 milliseconds per frame, ensuring smooth and responsive interaction. This level of performance is sufficient for most practical applications, including educational tools, smart home interfaces, and surveillance systems. The ability to provide immediate feedback through visual annotations further enhances the user experience, making the system intuitive and easy to use.

Usability was another key factor in the assessment. The system’s interface provides clear visual feedback, with bounding boxes highlighting detected faces and gestures and text annotations displaying classification results. This design ensures that users can easily understand the system’s output, even without prior experience with similar technologies. The modularity of the system also contributes to its usability, as it allows for straightforward integration of additional features or modifications. For example, new gesture classes or additional facial recognition tasks could be incorporated with minimal changes to the existing framework.

Despite its strengths, the system has some limitations. The Viola-Jones algorithm, used for face detection, may not perform optimally under challenging conditions such as poor lighting or occlusion. Additionally, the gesture recognition component is limited to a static region of interest, which restricts its flexibility. Addressing these limitations through the integration of modern face detection algorithms, such as MTCNN or YOLO, and implementing dynamic ROIs for gesture recognition could further enhance the system’s performance and versatility.

# Analysis

The system’s success can be attributed to several key factors, including the use of transfer learning, efficient deep learning architectures, and MATLAB’s robust development environment. Transfer learning was especially impactful, allowing pre-trained AlexNet models to be adjusted for specific tasks with smaller datasets. This approach significantly reduced training time and computational requirements while achieving high accuracy. The results demonstrate the effectiveness of AlexNet’s feature extraction capabilities, which are well-suited for tasks like facial and gesture classification [6][8].

The choice of RMSprop as the optimizer also played a crucial role in the system’s performance. RMSprop’s ability to adapt the learning rate for each parameter ensured steady convergence during training, as evidenced by the low loss values achieved by both models [7]. The consistent performance across epochs highlights the stability of the training process, which is essential for developing reliable deep learning models. However, experimenting with alternative optimizers, such as Adam or SGD with momentum, could provide insights into further performance improvements.

While the system demonstrates strong performance, its limitations highlight areas for future improvement. The Viola-Jones algorithm, though efficient, is less robust compared to modern deep-learning-based face detection methods like MTCNN or YOLO [9][10]. These methods utilise convolutional neural networks for more accurate detection, especially under conditions like low lighting or partial occlusions. Replacing the current face detection approach with one of these methods could significantly improve the system’s robustness and reliability.

The gesture recognition component is another area for potential enhancement. The current implementation relies on a static ROI, which restricts its ability to adapt to dynamic scenarios. Implementing dynamic ROIs or integrating body pose estimation techniques, such as OpenPose, could improve the system’s flexibility and applicability [9]. Increasing the gesture recognition dataset’s categories to include more diverse hand shapes, orientations, and backgrounds would also enhance the model’s generalization capabilities. Addressing these limitations could allow the system to be adapted for a greater range of applications not limited to multi-person interactions and complex gesture-based controls.

# Conclusion

This project successfully developed an interactive visual interface for facial and gesture analysis. The system achieved high accuracy for both tasks, with real-time performance being excellent and a tolerable amount of latency ensuring reliable responsive interaction. These results highlight the potential of combining deep learning and real-time image processing for human-computer interaction.

While the system demonstrates strong performance, future improvements could focus on addressing its limitations. Replacing the Viola-Jones algorithm with a modern face detection method and introducing dynamic gesture recognition capabilities would enhance its robustness and flexibility. Expanding the training datasets and incorporating more advanced visualization techniques could also improve the system’s usability and generalization.

Addressing these areas could allow the system to be applied in more areas such as educational tools and smart home interfaces as the results of this project highlight the feasibility of the mass utilization of deep learning to develop interactive visual interfaces, paving the way for further advancements in this field.

# References

[1] MathWorks, n.d. Deep Learning Toolbox - MATLAB. [online] Available at: https://www.mathworks.com/products/deep-learning.html [Accessed 27 November 2024].

[2] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. Communications of the ACM.

[3] MathWorks, n.d. Computer Vision Toolbox - MATLAB. [online] Available at: https://www.mathworks.com/products/computer-vision.html [Accessed 27 November 2024].

[4] MathWorks, n.d. Train Deep Learning Network to Classify New Images. [online] Available at: https://www.mathworks.com/help/deeplearning/ug/train-network-to-classify-new-images.html [Accessed 27 November 2024].

[5] Viola, P. and Jones, M.J., 2004. Robust real-time face detection. International Journal of Computer Vision. [Accessed 27 November 2024]

[6] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L., 2009. ImageNet: A large-scale hierarchical image database. CVPR. [Accessed 27 November 2024]

[7] Tieleman, T. and Hinton, G., 2012. RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. [Accessed 27 November 2024]

[8] Pan, S.J. and Yang, Q., 2010. A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering. [Accessed 27 November 2024]

[9] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E. and Sheikh, Y., 2021. OpenPose: Realtime multi-person 2D pose estimation using part affinity fields. IEEE Transactions on Pattern Analysis and Machine Intelligence. [Accessed 27 November 2024]

[10] Zhang, K., Zhang, Z., Li, Z. and Qiao, Y., 2016. Joint face detection and alignment using multi-task cascaded convolutional networks. IEEE Signal Processing Letters. [Accessed 27 November 2024]

# Appendix

|  |
| --- |
| **Visual\_Interfaces\_Main\_Script** |
| %% Initialization  clc;  clear all;  close all;  warning off;  % Load models  load faceRecognitionModel; % Pre-trained face recognition model  load FingerGestureModel; % Pre-trained gesture classification model  % Create the face detector object  faceDetector = vision.CascadeObjectDetector('FrontalFaceCART', 'MinSize', [150, 150]);  % Create the point tracker object for face tracking  pointTracker = vision.PointTracker('MaxBidirectionalError', 2);  % Initialize webcam and video player  cam = webcam();  videoFrame = snapshot(cam);  frameSize = size(videoFrame);  videoPlayer = vision.VideoPlayer('Position', [100 100 [frameSize(2), frameSize(1)] + 30]);  % Define the ROI for hand gestures (top-left corner)  x = 10; y = 10; width = 300; height = 300; % Adjust ROI size as needed  gestureROI = [x, y, width, height];  % Variables for face detection/tracking  runLoop = true;  numPts = 0;  bboxPoints = [];  oldPoints = [];  frameCount = 0;  %% Main Loop  while runLoop  % Capture the current frame  videoFrame = snapshot(cam);  videoFrameGray = rgb2gray(videoFrame);  % Face Detection/Tracking  if numPts < 10  % Detection mode  bbox = faceDetector.step(videoFrameGray);  if ~isempty(bbox)  % Detect points within the face region  points = detectMinEigenFeatures(videoFrameGray, 'ROI', bbox(1, :));  xyPoints = points.Location;  numPts = size(xyPoints, 1);  % Reinitialize the tracker  release(pointTracker);  initialize(pointTracker, xyPoints, videoFrameGray);  % Save points and bounding box  oldPoints = xyPoints;  bboxPoints = bbox2points(bbox(1, :));  end  else  % Tracking mode  [xyPoints, isFound] = step(pointTracker, videoFrameGray);  visiblePoints = xyPoints(isFound, :);  oldInliers = oldPoints(isFound, :);  numPts = size(visiblePoints, 1);  if numPts >= 10  % Geometric transformation for bounding box  [xform, oldInliers, visiblePoints] = estimateGeometricTransform(...  oldInliers, visiblePoints, 'similarity', 'MaxDistance', 4);  bboxPoints = transformPointsForward(xform, bboxPoints);  oldPoints = visiblePoints;  setPoints(pointTracker, oldPoints);  % Face classification  [img, faceDetected] = cropface(videoFrame);  if faceDetected == 1  img = imresize(img, [227, 227]);  facePrediction = classify(faceRecognitionModel, img);  feedbackFace = char(facePrediction);  else  feedbackFace = 'No Face Detected';  end  % Draw face bounding box  bboxPolygon = reshape(bboxPoints', 1, []);  videoFrame = insertShape(videoFrame, 'Polygon', bboxPolygon, 'LineWidth', 3);  videoFrame = insertText(videoFrame, bboxPoints(1, :), feedbackFace, ...  'FontSize', 20, 'BoxColor', 'green', 'BoxOpacity', 0.6, 'TextColor', 'white');  end  end  % Hand Gesture Classification  % Crop the ROI for gestures  handFrame = imcrop(videoFrame, gestureROI);  handFrame = imresize(handFrame, [227, 227]);  gesturePrediction = classify(FingerGestureModel, handFrame);  feedbackGesture = char(gesturePrediction);  % Annotate gesture ROI and prediction  videoFrame = insertShape(videoFrame, 'Rectangle', gestureROI, 'LineWidth', 3, 'Color', 'blue');  videoFrame = insertText(videoFrame, [gestureROI(1), gestureROI(2) + gestureROI(4) + 10], feedbackGesture, ...  'FontSize', 20, 'BoxColor', 'blue', 'BoxOpacity', 0.6, 'TextColor', 'white');  % Display the annotated video frame  step(videoPlayer, videoFrame);  % Check if the video player window is still open  runLoop = isOpen(videoPlayer);  end  %% Clean Up  clear cam;  release(videoPlayer);  release(faceDetector);  release(pointTracker); |
| **SimpleFaceRecognition\_Model\_Trainer** |
| % Create a datastore for the original training images  im\_original = imageDatastore('Faces\train\', 'IncludeSubfolders', true, 'LabelSource', 'foldernames');  % Ensure the 'Faces\cropped\' folder exists  croppedFolder = 'Faces\cropped\';  if ~exist(croppedFolder, 'dir')  mkdir(croppedFolder);  end  % Dynamically determine the unique labels in the dataset  uniqueLabels = unique(im\_original.Labels);  people = cellstr(uniqueLabels); % Ensure it matches the dataset labels  n = numel(people); % Number of unique labels  % Loop through each label, crop faces, and save them in respective folders  for i = 1:n  str = people{i}; % Use the correct label dynamically  ds1 = imageDatastore(fullfile('Faces\train\', str), 'IncludeSubfolders', true, 'LabelSource', 'foldernames');  dirName = fullfile(croppedFolder, str); % Specify the path for each label's folder  if ~exist(dirName, 'dir')  mkdir(dirName);  end  cropandsave(ds1, char(str)); % Pass char type for str  end  % Create a datastore for the cropped faces  im = imageDatastore(croppedFolder, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');  % Resize the images to the input size of AlexNet  im.ReadFcn = @(loc) imresize(imread(loc), [227, 227]);  % Split the dataset into training and testing sets  [Train, Test] = splitEachLabel(im, 0.8, 'randomized');  % Dynamically determine the number of unique labels in the training set  uniqueLabels = unique(Train.Labels);  n = numel(uniqueLabels); % Number of unique labels  % Load the AlexNet model  net = alexnet;  ly = net.Layers;  % Replace the fully connected and classification layers  fc = fullyConnectedLayer(n, 'Name', 'fc'); % Output size matches 'n'  ly(23) = fc;  cl = classificationLayer('Name', 'classification');  ly(25) = cl;  % Set training options  learning\_rate = 0.000008; % Original value was 0.00001  opts = trainingOptions('rmsprop', ...  'InitialLearnRate', learning\_rate, ...  'MaxEpochs', 100, ...  'MiniBatchSize', 64, ...  'Plots', 'training-progress');  % Train the network  [faceRecognitionModel, info] = trainNetwork(Train, ly, opts);  % Evaluate the model  [predict, scores] = classify(faceRecognitionModel, Test);  names = Test.Labels;  pred = (predict == names);  s = size(pred);  acc = sum(pred) / s(1);  fprintf('The accuracy of the test set is %f %% \n', acc \* 100);  % Save the trained model  save faceRecognitionModel;  %% Prediction Section - Test a New Image  % Use the code below with a path to new image  img = imread('Faces\cropped\Jawad\_Expressionless\1.jpg');  [img, face] = cropface(img);  figure;  imshow(img);  % Check if a face is detected  if face == 1  img = imresize(img, [227, 227]);  predict = classify(faceRecognitionModel, img);    % Dynamically match prediction to the people array  for i = 1:n  if predict == uniqueLabels(i)  fprintf('The face detected is %s\n', char(uniqueLabels(i)));  break;  end  end  else  disp("No Faces detected");  end |
| **Gesturing\_Model\_Trainer** |
| %% - Training and Save Section  % Clear the workspace, close all figures, and suppress warnings  clc  clear all  close all  warning off  % Display message indicating the expected training duration  disp("This training can take up to 8 minutes...")  % Define the learning rate for training the network  learning\_rate = 0.000008;  % Load the pre-trained AlexNet model  g = alexnet;  layers = g.Layers;  % Modify the AlexNet architecture to suit the gesture classification task  layers(23) = fullyConnectedLayer(5); % Replace the fully connected layer for 5 gesture classes  layers(25) = classificationLayer; % Replace the final classification layer  % Load and preprocess the gesture dataset  allImages = imageDatastore('gesturing\', ... % Load images from the 'gesturing' folder  'IncludeSubfolders', true, ... % Include images from subfolders  'LabelSource', 'foldernames'); % Use folder names as labels  allImages.ReadFcn = @(loc) imresize(imread(loc), [227, 227]); % Resize images to 227x227 (AlexNet input size)  % Define training options for the network  opts = trainingOptions("rmsprop", ... % Use RMSprop optimizer  "InitialLearnRate", learning\_rate, ... % Set the initial learning rate  'MaxEpochs', 200, ... % Train for 200 epochs  'MiniBatchSize', 64); % Use a mini-batch size of 64  % Train the network using the specified layers and options  FingerGestureModel = trainNetwork(allImages, layers, opts);  % Save the trained model to a file  save FingerGestureModel;  % Display message indicating that training is complete  disp("Training Completed") |