Bangladesh University of Engineering and Technology

Department of Electrical and Electronics Engineering

Course No: EEE 212

Course Title: Numerical Technique Laboratory

Project Title: Attendance Marking System Using MATLAB

Submitted To:

Mr. Hamidur Rahman

Associate Professor, Department of Electrical and Electronic Engineering,

Bangladesh University of Engineering and Technology

Shahed Ahmed

Lecturer, Department of Electrical and Electronic Engineering,

Bangladesh University of Engineering and Technology

Shaimur Salehin Akash

Lecturer (PT), Department of Electrical and Electronic Engineering,

Bangladesh University of Engineering and Technology

Submitted By:

Ramisa Tahsin Shreya (ID: 1906072)

Ramim Hasan Shawn (ID: 1906082)

Table of Contents

List of Illustrations

Introduction:

After starting the app, webcam will be turned on and it will be the input for our app. The main problem is to process the inputs captured from webcam, compare it with the data storage, recognize the persons and make attendance sheet along with it. We will show some test cases in later part of the report. So, our main problems are:

- Capturing inputs using webcam and processing it.
- Comparing processed snapshots with data storage and recognize the students
- Create an attendance sheet of recognized students where recognized students will be categorized as 'present' and others as 'absent'.

Theory:

We first made a data storage with MATLAB file 'DataStore.m'. We input video files and the m file took snapshots and saved images in a file. After creating the data storage, we trained the data sets using machine learning. These two steps are needed to be done beforehand to run our project successfully. In the main code, we will take input from webcam, MATLAB file named TestingModelFinal will process the input, compare the input with data storage and recognize the students. After the recognition process, the code will create an attendance array which will be displayed as a table with appdesigner.

Algorithm:

1. Data Storage

- For the data storage part, we read the video of the person with the **videoreader** function and save it in the f variable.
- We save the number of frames in n variable.
- By using **vision.CascadeObjectDetector**, we detect human face and store it in the facedetector variable.
- We set the number of images. In our case, we took 200 snaps from single video.
- Now we create a loop so that we can take multiple images.
- Inside the loop we read the frames of the video, detect face and save the pixel information inside the bboxes variable. Bboxes is a 1*4 matrix that stores the x, y value of the upper left pixel of the detected face, the height and width of the detected portion.
- If a face is detected we maintain number of shots and then crop the detected faces using with **imcrop** function.
- We resize it to [227,227] pixel which is mandatory for the use of **alexnet.**
- Then we save the file in the current path in a serial order with **imwrite** function.
- we keep showing the cropped images with **imshow** function.

- if no face is detected we just show the full image.

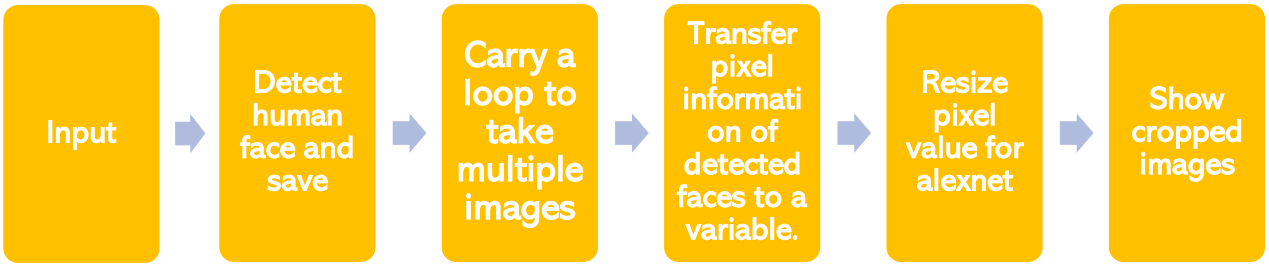

Fig 01: Flow Chart of Data Store

2. Training Model:

- We select the path of the data storage that we want to train and keep all the specific folder in a single folder. Alexnet will train these images and label the specific data with name of that folder. We also select the ratio of training image and validation image. Here we have trained 70% of the image and checked them with the rest 30% to avail accuracy.

  Code :

```
imds =
imageDatastore('datastorage','IncludeSubfolders',true,'LabelSource','foldernames' );
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

- Loaded the pretrained AlexNet neural network.
  net = alexnet;

- The first layer, the image input layer, requires input images of size 227-by-227-by-3, where 3 is the number of color channels.
  code:inputSize = net.Layers(1).InputSize

```
inputSize = 1×3

   227   227     3
```

- The last three layers of the pretrained network net are configured for 1000 classes. These three layers must be fine-tuned for the new classification problem. We extracted all layers, except the last three, from the pretrained network.

  Code: layersTransfer = net.Layers(1:end-3);

- Transferred the layers to the new classification task by replacing the last three layers with a fully connected layer, a softmax layer, and a classification output layer.

  Code: numClasses = numel(categories(imdsTrain.Labels))

- Specified the options of the new fully connected layer according to the new data. Set the fully connected layer to have the same size as the number of classes in the new data. To learn faster in the new layers than in the transferred layers, increase the WeightLearnRateFactor and BiasLearnRateFactor values of the fully connected layer.

```
layers = [
    layersTransfer

fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',2
0)
    softmaxLayer
    classificationLayer];
```

- The network requires input images of size 227-by-227-by-3, but the images in the image datastores have different sizes. We used an augmented image datastore to automatically resize the training images, specified additional augmentation operations to perform on the training images: randomly flipped the training images along the vertical axis, and randomly translated them up to 30 pixels horizontally and vertically. Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...

'RandXReflection',true,'RandXTranslation',pixelRange,'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);
```

- To automatically resize the validation images without performing further data augmentation, we used an augmented image datastore without specifying any additional preprocessing operations.

```
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

- Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network. In the previous step, you increased the learning rate factors for the fully connected layer to speed up learning in the new final layers. This combination of learning rate settings results in fast learning only in the new layers and slower learning in the other layers. An epoch is a full training cycle on the entire training data set. The software validates the network every ValidationFrequency iterations during training.

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',10, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',1e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',3, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

- Train the network that consists of the transferred and new layers .

```
netTransfer = trainNetwork(augimdsTrain,layers,options);
```

- Classify the validation images using the fine-tuned network.

```
[YPred,scores] = classify(netTransfer,augimdsValidation);
```

- Calculate the classification accuracy on the validation set. Accuracy is the fraction of labels that the network predicts correctly.

```
YValidation = imdsValidation.Labels;
accuracy = mean(YPred == YValidation)
```

- We train the dataset to use for face recognition and save it in the name recog.

```
recog = trainNetwork(imds,layers,options)

save recog;
```

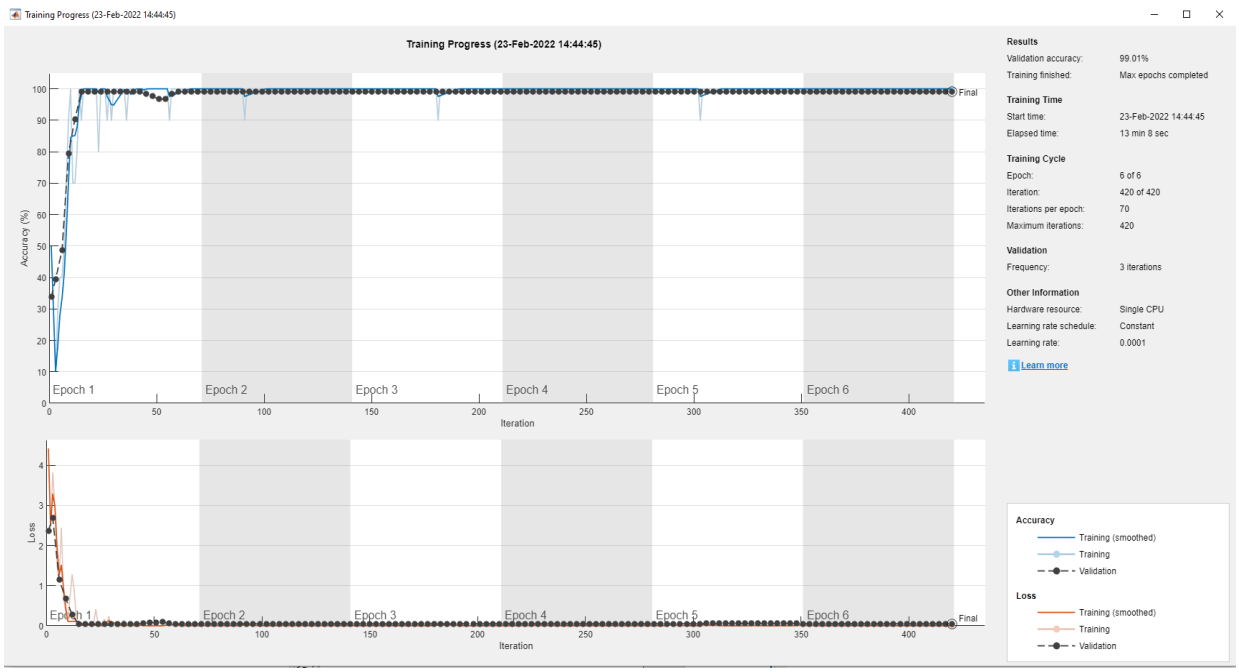

Fig 02: Progress of Training Dataset

```
Command Window

inputSize =

   227   227     3


numClasses =

     6


accuracy =

    0.9901


recog =

  SeriesNetwork with properties:

         Layers: [25×1 nnet.cnn.layer.Layer]
     InputNames: {'data'}
    OutputNames: {'classoutput'}
```

Fig 03: Class and Accuracy of Dataset

3. Testing Model :

This m file is basically a function named TestingModelFinal. The output we'll get after calling this function is att1, which is string array consisting data present or absent students. The parameter used is c where we pass webcam input later on. Other variables of this section is : att (row matrix which will have value '1' if a student is present and '0' otherwise), Name(string array data type of name of students) and many more. Basic algorithm of this section is discussed below:

- We load data set **MyNet**
- Detect faces using **vision.CascadeObjectDetector** and store it in facedetector varible.
- Create a loop where we take snapshots from webcam, save it in e variable.
- Pixel information of detected faces is saved in bboxes.
- From bboxes, we crop and resize face part of the picture.
- We comapre the detected faces from data set using **classify** function and if we get a match, we put the value in label variable.
- Now using a loop we compare label variable with Name variable, if we find a match we set the value of att as '1', otherwise as '0'
- We use a final loop where we set value of elements of att1 variable comparing it with att variable.

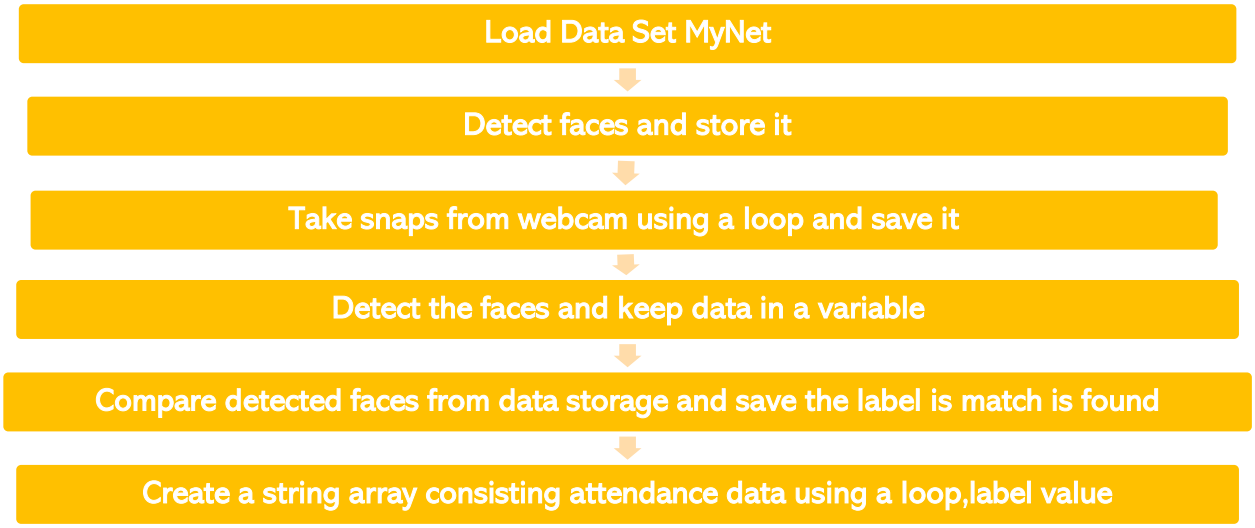

Fig 04: Flow Chart for Testing Model

4. Appdesigner :

Homeapp:

- In the callback function of the AttendanceMarkingApp pushed button we used a **callingapp** function. If the button is pushed it would call the finalProject app.

```
home.app.callingapp=FinalProject;
```

Attendance Marking App:

- We declared 3 global variable : webcamObject, imageObject, att1.

```
properties (Access = private)
    webcamObject;
    imageObject;
    att1; Desccription
    end
```

- Then we opened the webcam in axes. In the callback function of the StartWebcam push button we wrote the code showed below. If the button is pushed it would start the webcam. Our webcam resolution was 640x480. We set the resolution in the xy axis in that way. We took image from the webcam with the webcamObject., saved the split image in the imageObject and showed the preview in the axes. Then we called the ModeltestingFinal function whose parameter was webcam. It would pass output which is the attendance of the students from our dataset.

```
app.webcamObject = webcam;
app.imageObject = image(app.UIAxes);
axis(app.UIAxes,'ij')
app.webcamObject.Resolution = '640x480';
res = split(app.webcamObject.Resolution,'x');
app.UIAxes.XLim = [0,str2double(res{1})];
app.UIAxes.YLim = [0,str2double(res{2})];
app.webcamObject.preview(app.imageObject);
app.att1 = ModelTestingFinal(webcam);
```

- To show the list of the attendance, we created a table and a push button called 'ShowAttendance' which would show the list of attendance. We also saved the data in a excel sheet.

- We created an array that have 3 column. Student name in the 1st column, id in the 2nd column and attendance in the 3rd column. We turned the array into a table with **array2table** function. Saved the data in a variable and showed it in an excel sheet using **writetable** function.

```
t = ["Ramisa" 72 app.att1(1);"Ramim" 82 app.att1(2);"Zeal" 101
app.att1(3);"Anindya" 81 app.att1(4);"Tasmin" 55 app.att1(5);"Samiyee" 07
app.att1(6)];

app.UITable.Data = array2table(t);
data = app.UITable.Data;
filename = 'att.xlsx';

writetable(data,filename)
```

Main Code:

The main objective of our project is attendance marking and the file that is doing the task is Testing Model Final Function. After function callback of appdesigner passes webcam input into the function, it gives att1 output which is our ultimate attendance sheet. Code of the main file is given below :

```
function att1 = ModelTestingFinal(c)
load recog;
faceDetector=vision.CascadeObjectDetector;
name = ["Ramisa","Ramim","Zeal","Anindya","Tasmin","Samiyee"];
n = length(name);
att = zeros(1,n);
att1 = strings;
idx=1;
for idx = 1:20
    e=c.snapshot;

    bboxes =step(faceDetector,e);

    if(sum(sum(bboxes))~=0)

        es=imcrop(e,bboxes(1,:));

        es=imresize(es,[227 227]);

        label=classify(recog,es);
        for i=1:n
            if strcmp(name(i),char(label)) == 1
                j = i;
                att(1,j) = 1;
                break
            end
```

```
        end
        for i=1:n
            if att(1,i) == 1
                att1(i) = "present";
            else att1(i) = "absent";
            end
        end
    end
    idx = idx + 1;
end
%clear('cam')
```

Interface of the App:

Home page Interface:

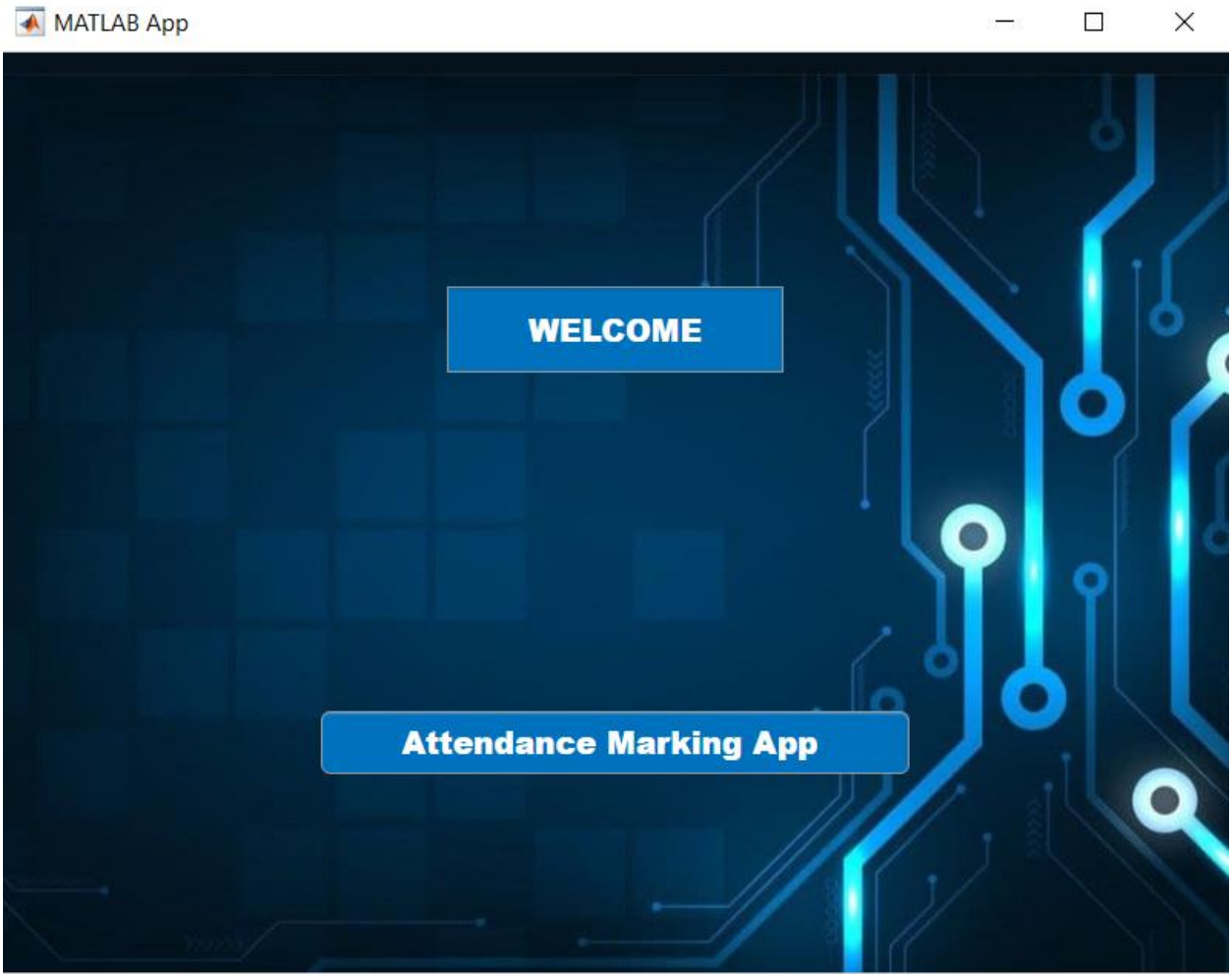

Fig 05: Home Interface of the App

Final Project App Interface:

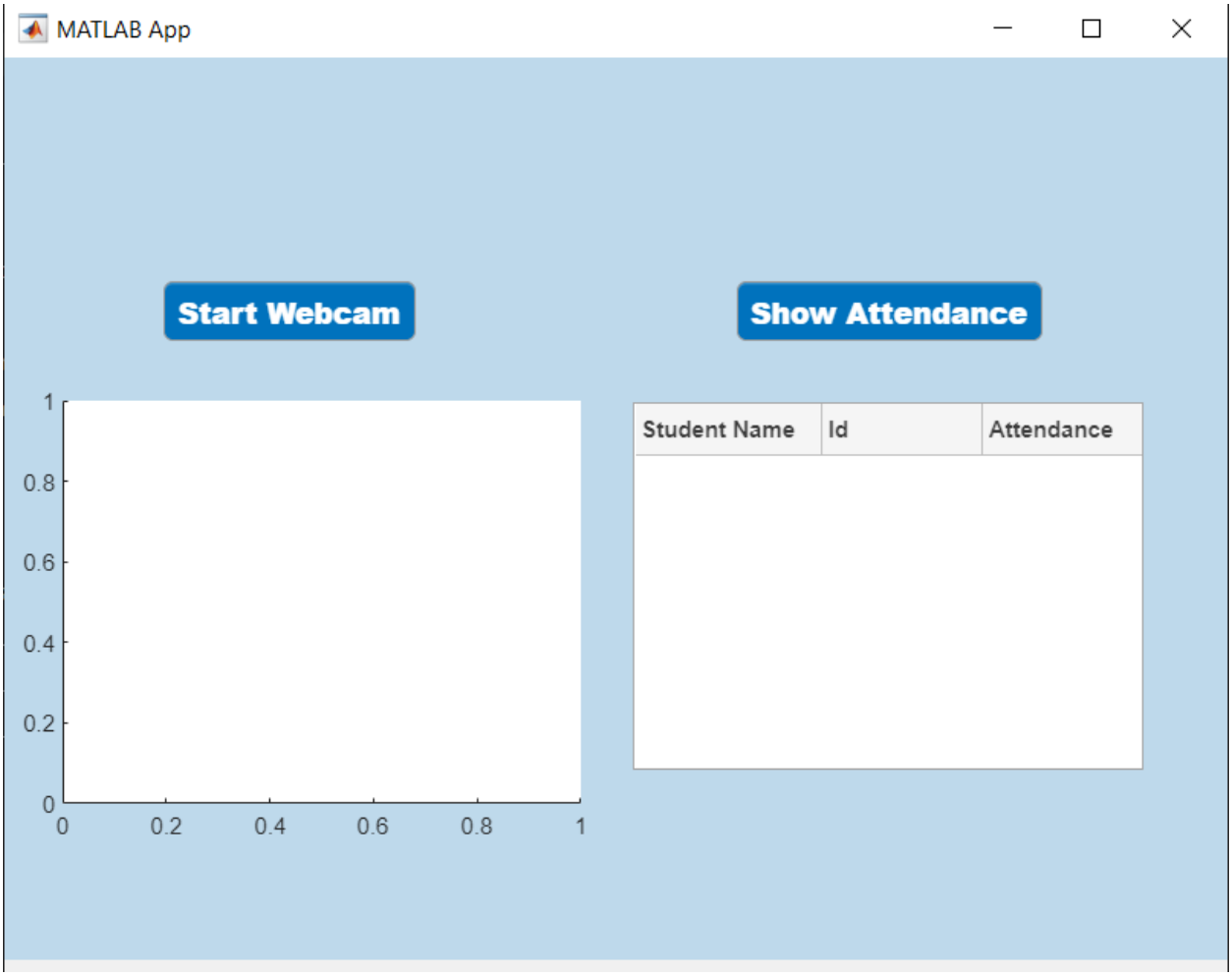

Fig 06: Attendance Marking App Interface

Test Case:

Case 01:

Inputs:

We will take input of students named “Ramisa” and “Tasmin” through the webcam.

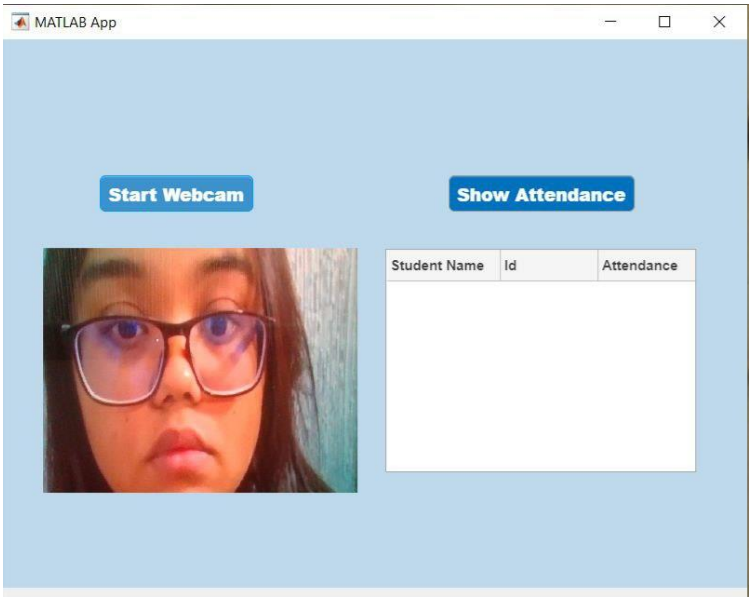

Fig 07: Input of Ramisa in webcam

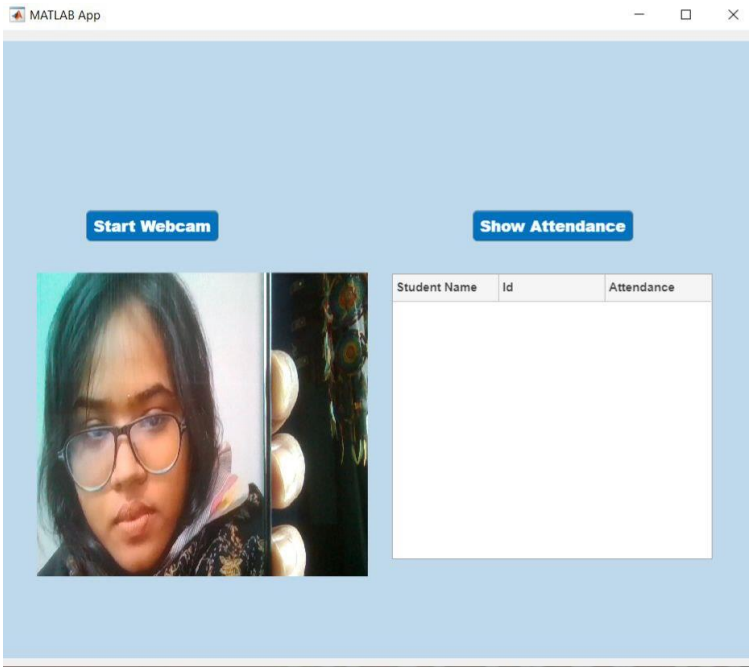

Fig 08: Input of Tasmin in webcam

Output:

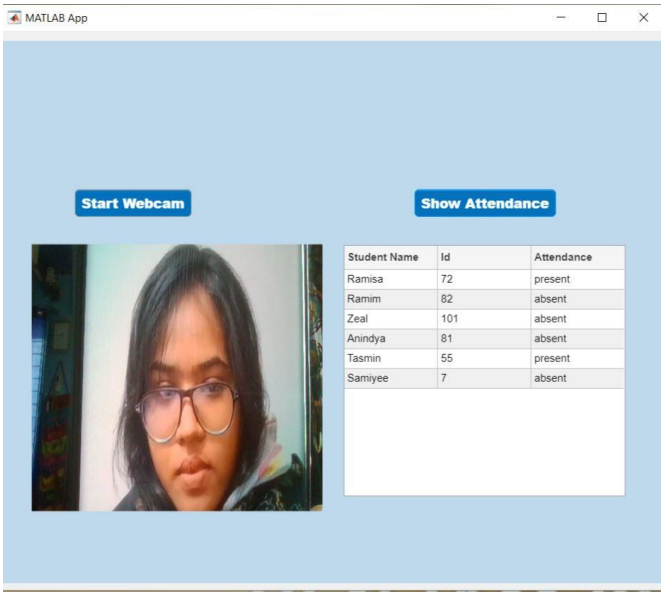

Fig 09: Output of Test case 01

Case 02:

Inputs

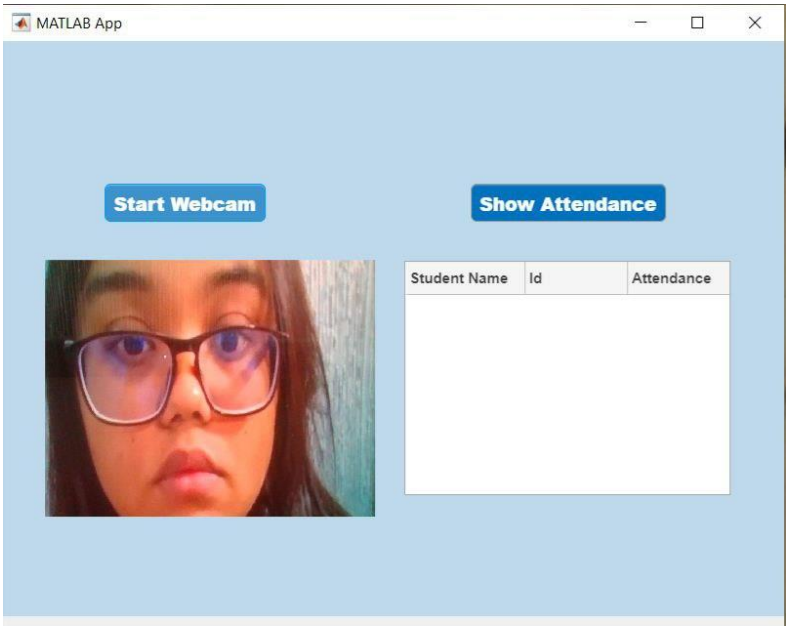

Fig 10: Input of Ramisa in Webcam

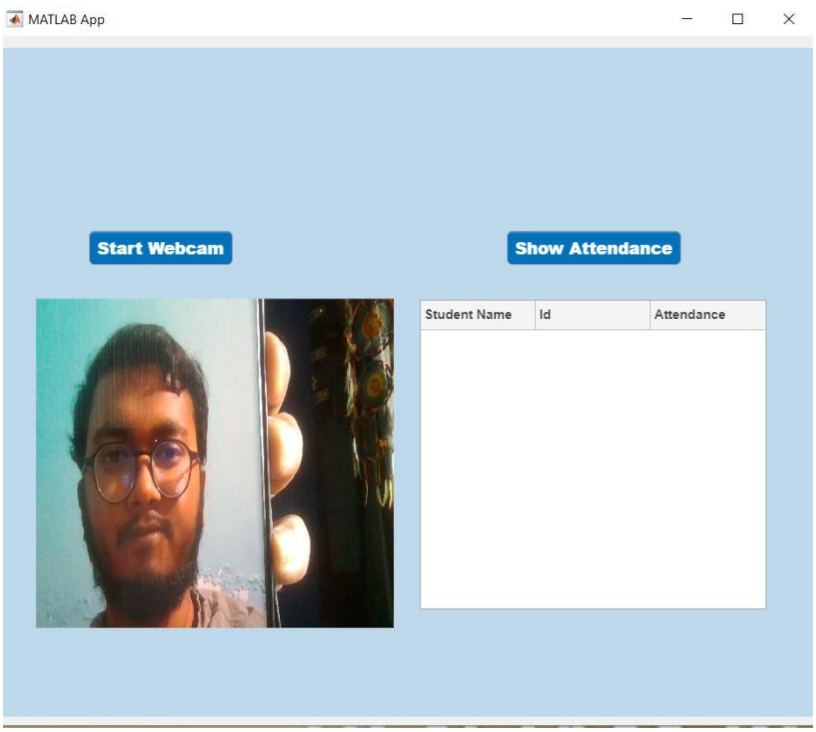

Fig 11: Input of Ramim in Webcam

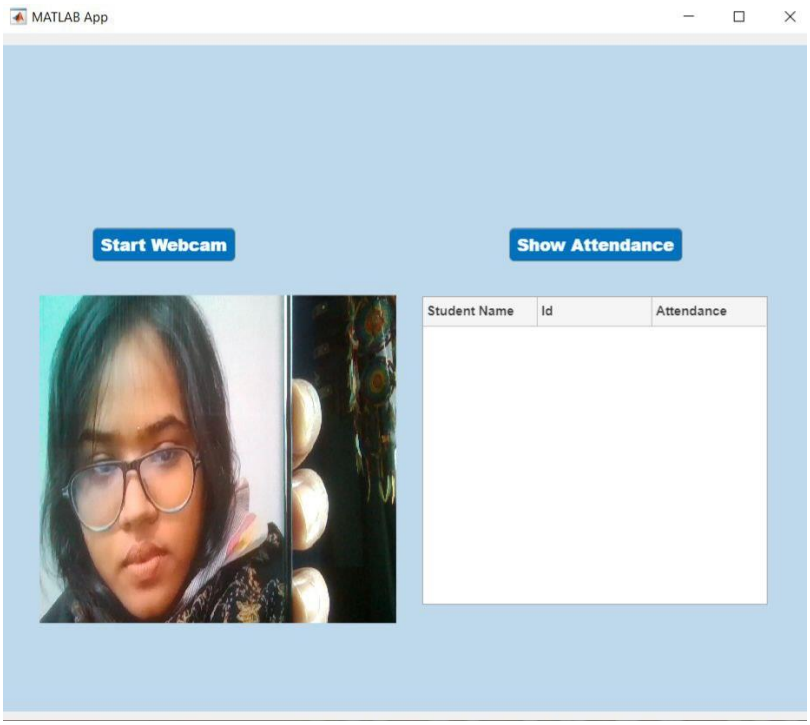

Fig 12: Input of Tasmin in Webcam

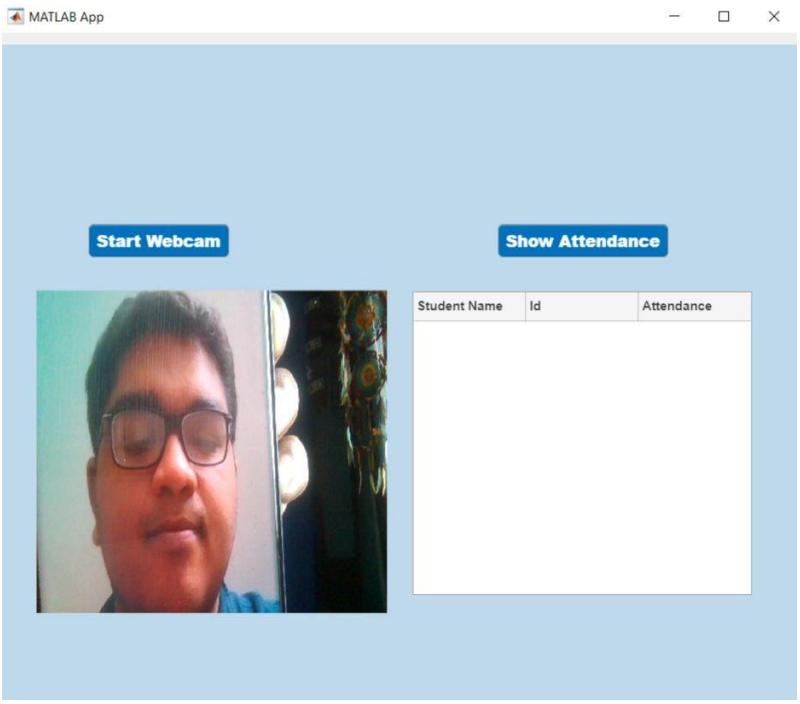

Fig 13: Input of Samiyee in Webcam

Output:

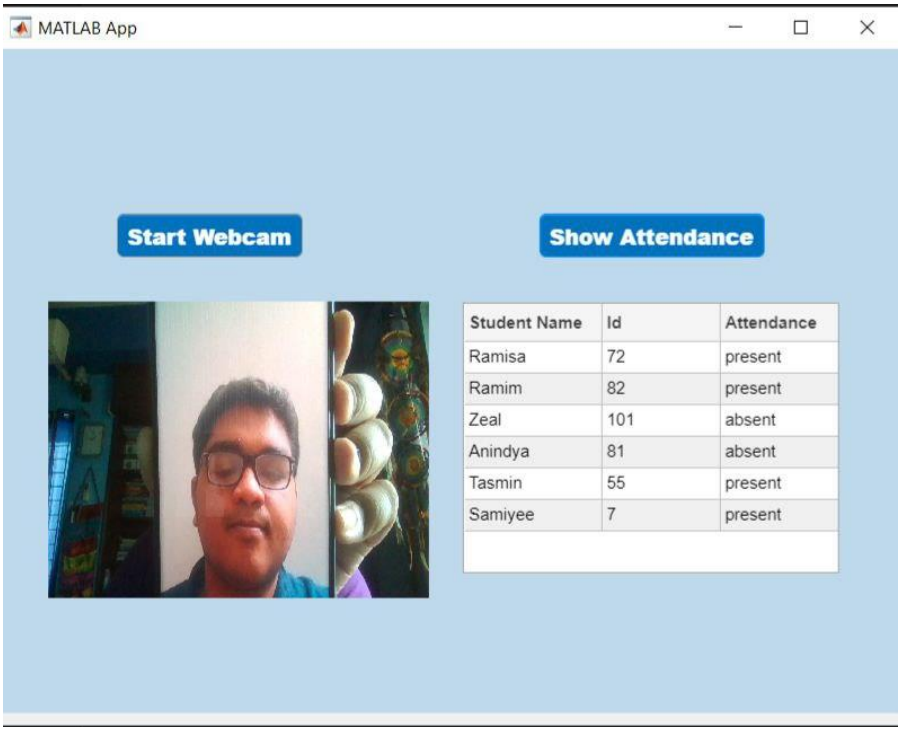

Fig 14: Output of Case 02

Applications:

Our project can be used to automatically mark attendance in classrooms, office and other institutions. People will show their faces into webcam and attendance will be saved for that date automatically.

Conclusion:

Even though this project was able to successfully detect, recognize faces and make attendance sheet along with it, this system is not 100% accurate. There is sometimes error in recognition. If we make database with more snapshots, the recognition will be more accurate. At the same time, enhancing the depth of train model program will provide more accurate results further.