www.revistageintec.net
ISSN: 2237-0722
revista GEINTEC
Gestão, Inovação e Tecnologias

New Features for Webcam Proctoring Using Python and Opencv

S. Harish[1]; D. Rajalakshmi[2]; T. Ramesh[3]; S. Ganesh Ram[4]; M. Dharmendra[5]
[1]Systems Engineer - Specialist, Infosys Ltd.
[1]ucs17204@rmd.ac.in
[2]Assistant Professor, Department of CSE, R.M.D. Engineering College, India.
[2]drl.cse@rmd.ac.in
[3]Assistant Professor, Department of CSE, R.M.K Engineering College, India.
[3]trh.cse@rmkec.ac.in
[4]Final Year, Department of CSE, R.M.D. Engineering College, India.
[4]ucs17137@rmd.ac.in
[5]Final Year, Department of CSE, R.M.D. Engineering College, India.
[5]ucs17130@rmd.ac.in

Abstract
Online examinations have turned out to be the new normal. However, it is not that easy to proctor the students as rigorously as in in-center examinations. It is essential to find an approach to proctor the online examinations too as rigorously as possible. There are already several webcam proctoring systems that are used in the real world, but these systems are not very accurate and miss out on detecting all possible malpractices and in certain cases due to defect in the system it detects a malpractice for someone who never even attempted any. This project focuses mainly on building features that can make the existing webcam proctoring system more advanced and rigorous. The project is aimed at building the following features namely head pose estimation, mouth opening detection, eye ball monitoring, number of persons detection, mobile phone detection and face spoofing detection. For each of these features, machine learning models are built using Python. All these features make use of the live webcam feed which is obtained using OpenCV and an output is obtained which gives information about the direction of the head and eyes, presence of more than one person and presence of mobile phone, opening of mouth, occurrence of face spoofing. All these outputs are recorded as a log file which can be used to identify any possible malpractices based on these features.

Key-words: Computer Vision, Convolutional Neural Networks, OpenCV, Python, Machine Learning, You Only Live Once (YOLO), DarkNet Residual Layer, Deep Neural Networks (DNN).

1. Introduction

Online examination systems are still at a very early stage and have a lot of improve with respect to its user interface as well as its proctoring abilities. Most of the online examination systems have all kinds of proctoring features but the accuracy of these features is still under scrutiny. Webcam, microphone and screen sharing are the three major data that are collected during online examinations. Of these, the data collected from webcam is still not used to its full potential. In the present day online proctoring systems, only face detection process is implemented.

Objective

The main objective here is to utilise the webcam feed more. In addition to detection of face from the webcam feed, several other features are detected and monitored. These features are head pose estimation, mouth opening detection, eyeball monitoring, identifying presence of more than one person, detecting the presence of mobile phone and identifying face spoofing. All the outputs obtained as part of the monitoring process will be stored in a separate log file.

Existing System

Most existing web proctoring systems focus only certain basic features such as identifying the presence of more than one person in the frame. While there are some systems that have more features to identify malpractice, these features are not very accurate and sometimes miss out on detecting some malpractices. Sometimes due to inaccuracy of the model, the system might function defectively and might produce false results. Further, the existing webcam proctoring systems do not capture live feeds and usually capture images every few seconds. Usually this is done to reduce the network usage of the person. However, due to this, the proctoring system might miss out many activities and due to time gap between two frames that are captured, it might produce false results due to noticeable differences in the two images (frames).

Proposed System

The proposed system brings out a lot more features and more accuracy to the existing system. Firstly, using OpenCV, live webcam feed is captured and processed, which means, several frames are captured per second. Next, in addition to just detecting number of faces, several other features such as

monitoring eye movement, identifying the presence of objects such as mobile phone and other electrical gadgets, face spoofing detection, estimating position and direction of the head and measuring the distance between lips to identify if the person is talking.

Expected Result

Several new features based on the webcam feed are implemented. The existing features of detecting multiple faces will be more accurate with lesser errors and defects. The newer features such as detecting eye ball movement, finding distance between the lips, estimating position and direction of the head, identifying mobile phones and electrical gadgets and face spoofing detection will be implemented. Tracking eye movement along with estimating head position and direction will prevent the person from looking elsewhere. Measuring distance between the lips can easily identify if a person is talking. Face spoofing detection ensures that there is a real person taking up the test. Mobile phone detection identifies the usage of mobile phones. Addition of these features will make the overall webcam proctoring system more rigorous.

2. Related Work

Face Detection and Recognition using OpenCV

There are several algorithms that can be used for detecting faces from an image. Some of the algorithms are Haar Cascade frontal face, Eigen Face Recognizer, Fisher Face Recognizer and Local Binary Patterns Histograms. Principal Component Analysis of face recognition is done and programming for face recognition using OpenCV is discussed. Various vector projections on one dimensional and two-dimensional planes and its matrices and vectors are discussed.

Face Detection and Tracking using OpenCV

An application for detection and tracking of faces from photos and videos using OpenCV is discussed. Various face detection algorithms such as Adaboost, Haar Cascade are discussed and comparison is done. The pre-requisites of face detection are discussed. Finding faces using colour and motion is discussed.

Live Object Identification Using Deep Learning Techniques and OpenCV

The various techniques for real time object detection are discussed. Several deep learning algorithms such as Region based Convolutional Neural Networks, Faster Region based Convolutional Neural Networks, Single Shot Detector, You Only Look Once are discussed and their performances are compared. The combination of Single Shot Detector and Mobile Nets are also discussed.

Face Modeling Process Based on Dlib

The face modeling process using a random forest model is discussed. A global optimization model which can be used to test the sample feature point localisation along with automatic localization of facial feature points is discussed. A method based on gradient enhancement is also discussed. The various principles of face point positioning are estimated. Several experiments for comparing the accuracy of various techniques are done.

Head Pose Estimation Using Deep Architectures

The problem of head pose estimation under Deep Convolutional Neural Networks is investigated. The various architectures of Deep Convolutional Neural Networks are compared and their accuracy is tested. It is proven that the addition of facial fiducial points can increase the accuracy of continuous head pose estimation.

3. Methodology

The first step is building an efficient machine learning model that can be used for detecting various activities of the person. There are several features and activities that are being proctored and each of these features and activities need separate machine learning models. The next step here would be to use OpenCV to obtain input from the webcam and convert the input from the webcam into several images. All these images obtained will be used to identify various features and activities of the person by using the machine learning algorithm that was built for the purpose. The next step is to create logs about the various activities of the person which can be later used for identifying if a person has indulged in any sort of malpractice or not.

Figure 1- A flow Diagram Representing the various Steps involved in the Project Including Two main Steps Namely (a) Building the Model and (b) Real Time Monitoring

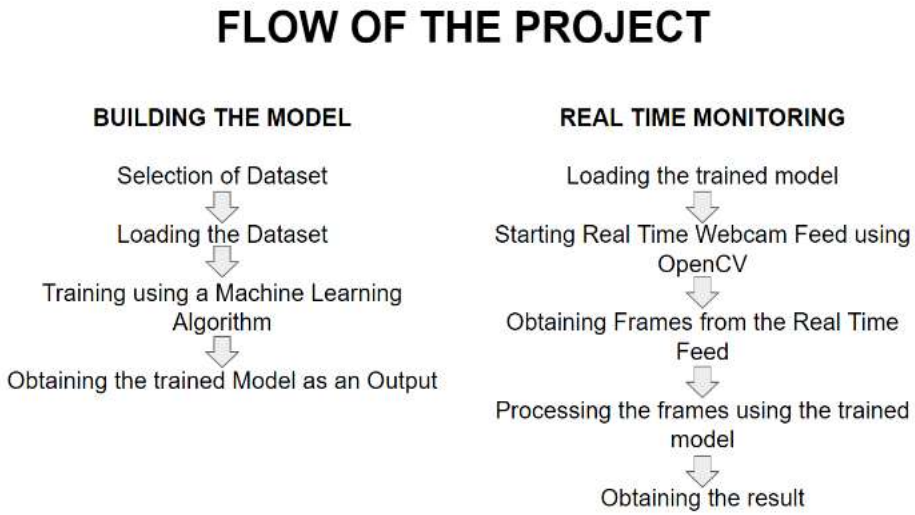
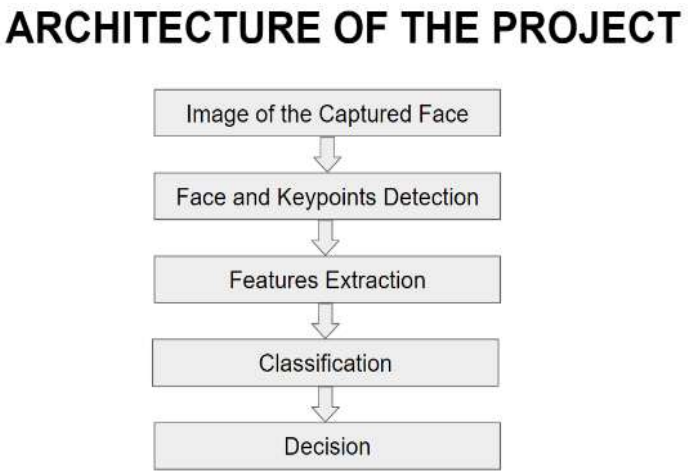


Figure 2- A Flow Diagram of the Steps in Processing the Images and obtaining the Final Result



Building Machine Learning Models

Two machine learning models are required to be built for implementing all these features. The first one is the implementation of a model based on Extra Trees Classifier for detecting face spoofing. The next one is the implementation of a model based on Convolutional Neural Networks for identifying facial landmarks. The face spoofing model is built using Extra Trees Classifier technique based on a dataset from Kaggle which consists of real and spoofed images. The face landmark model is built using Convolutional Neural Networks based on a dataset from Kaggle which consists of images and the face landmark points. A dataset from Kaggle is taken and it is pre-processed and the training and testing variables are set. Next, the Convolutional Neural Networks are designed. Finally, the model is obtained.

Reusable Code for Face Detection

A face detection model which is based on OpenCV's DNN module is used. It is a Caffe model which is based on the Single Shot - Multibox Detector (SSD) and uses ResNet-10 architecture as its backbone. A function for finding the faces from the image is defined that makes use of the model based on DNN and returns the output with the coordinates of the four corners of the face. The input is an image in the form of a numpy array and the model and the output is an array consisting of the four coordinates of the face. First, the image passed as the input is converted into blob format using OpenCV. Next, using the model, a confidence or a probability value for the presence of face is found out. If this probability value if greater than 0.5, then it is considered that a face is detected and the four coordinates around the face are estimated and passed as the output. A function for drawing a rectangle is defined that draws a rectangle based on the four coordinates obtained as output. The input is an image in the form of a numpy array and the array containing the four coordinates of the image and the rectangle drawn over the coordinates is the output. The rectangles are drawn along the coordinates by using OpenCV's rectangle function. The image and the four coordinates are passed as input to the rectangle function.

Reusable Code for Face Landmarks Detection

The face landmark model which is built based on Convolutional Neural Networks is used. A function that makes the rectangular box into a square box is defined. The array containing coordinates of the face are passed as the input and the output is an array with modified coordinates that makes the box into a square. The coordinates are modified in a way that such that the rectangular box gets expanded into a square box. A function for moving the box towards the direction specified by the vector offset is defined. An array containing the coordinates of the box and an array containing the values of offset for each coordinate are passed as the input. The modified coordinates based on the offset values are passed as the output. A function for detecting the various facial landmarks is defined. The input is an array containing coordinates of the face, the image and the face landmark model. The output is a numpy array consisting of the coordinates of various facial landmarks. A function for drawing points across the various facial landmarks is defined. The input consists of the numpy array with coordinates of various facial landmarks and the image. The output is the various points that are drawn across the various face landmarks in the image.

Head Pose Estimation

Head pose estimation is one of the features that can be implemented using the face landmark model. The head pose estimation is the identification of the direction in which the head is facing. It is implemented for four directions, namely, left, right, up and down. The first step would be to import the python files that consists of functions for face detection and face landmark detection. A function for creating the 3D coordinates that are obtained as a result of the facial landmark detection into 2D coordinates is defined so that an annotation box can be created. The image, the rotation vector, the translation vector, the camera matrix and an array containing the rear and frontal size and depth. The 3D coordinates are converted to 2D coordinates using the OpenCV's projectPoints function and is later converted to a numpy array. The numpy array of 2D coordinates is the output. A function for drawing an annotation box on the face for the purpose of head pose estimation is defined. The image, rotation vector, translation vector and camera matrix are passed as the input. The lines are drawn using OpenCV's polylines and line function. The lines drawn are the output. A function for getting the points for estimating the head pose in sideways direction is defined. The image, rotation vector, translation vector, camera matrix is passed as the input. An already defined function that is used for obtaining 2D coordinates is used to obtain the 2D coordinates. From these coordinates, two points are obtained. A tuple of these two points is returned as the output. A numpy array consisting of various key landmarks are initialised. The camera matrix is initialised based on the focal length. The required models for face detection and face landmark detection are initialised to variables.

The webcam is started using the OpenCV's VideoCapture function. Each frame of the webcam feed is taken and processed and an image is obtained. The face is detected in this image. Further, specific face landmarks are also identified and are marked. A line is drawn from the face landmark coordinate of nose using functions already defined and further the angle is calculated. Depending on the value of the angle, the direction of the head is estimated as either left, right, up or down. The output is displayed on the live feed as well as recorded in the log file.

Mouth Opening Detection

Mouth opening detection is also one of the features that can be implemented using the face landmark model. This is done by the measuring the distance between the upper lip and the lower lip using the face landmark coordinates of the mouth obtained. The first step would be to import the python files that consists of functions for face detection and face landmark detection. The face

detection model and the face landmark models are initialised to variables. The default points of the inner lip and the outer lip are initialised to arrays. These default points are the points when the mouth is closed.

The webcam is started using the OpenCV's VideoCapture function. Each frame of the webcam feed is taken and processed and an image is obtained. The face is detected in this image. Further, specific face landmarks are also identified and are marked. In this case, the landmark points of the lips are marked. For each image or frame that is obtained, the landmark points for the mouth are obtained and recorded. The landmark points of the processed image and the default landmark points are compared and if the landmark points of the processed image deviates from the default landmark points, the output is displayed on the webcam feed screen that the mouth is open and also the output that is obtained is stored in the log file.

Eye Tracking

Eye tracking is also one of the features which can be implemented using the face landmark model. It used the coordinates obtained from the face landmarks and detects the eyes. The movement of these coordinates is tracked and recorded. The first step would be to import the python files that consists of functions for face detection and face landmark detection. A function for finding the ROI (Region of Image) on mask of the size of the eye and finding extreme points of the eye is defined. A blank mask, an array containing facial landmarks of the eyes and a numpy array containing all facial landmarks are passed as the input. A mask is created using OpenCV's fillConvexPoly function. Also, the coordinates of the extreme points of the eye are calculated. The mask in the form of numpy array and the extreme points as an array are returned as an output. A function for finding and returning the eye ball positions is defined. The array of end points and the actual values of the eye points for the processed image are passed as input. By comparing both the values, the output is returned as either normal, left, right or up. A function for finding the largest contour on an image divided by a midpoint and subsequently the eye position is defined. The thresholded image of one side containing the eyeball as numpy array, the midpoint between the eyes as integer, the image and an array containing extreme points of the eye are passed as the input. The OpenCV's find Contours and moments functions are used to find the position of the eye. The position as an integer is returned as the output. A function for pre-processing the thresholded image is defined. The thresholded image as a numpy array is passed as the input. The OpenCV's erode, dilate, medianBlur and bitwise_not functions are

used to pre-process the image. The pre-processed thresholded image in the form of a numpy array is returned as the output. A function for printing the position of the eye is defined. The left eye position, right eye position and the image are passed as the input. By comparing the values of the left and right eye positions, the output is printing as looking left, right or up. The face detection model and the face landmark models are initialised to variables. The default coordinates for the left eye and the right eye are also initialised to variables.

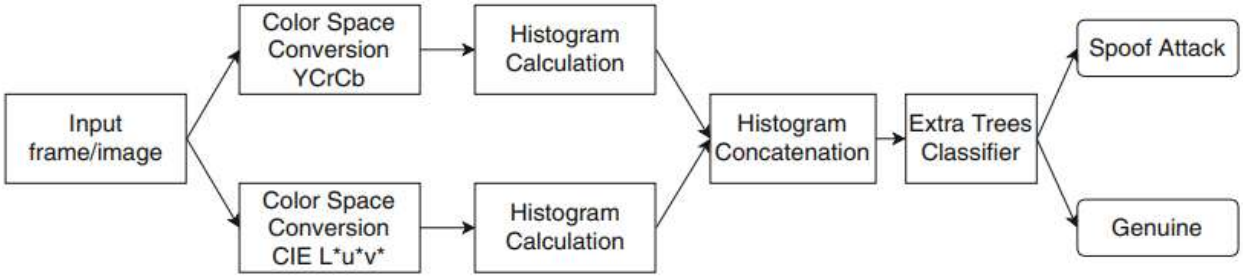
The webcam is started using the OpenCV's VideoCapture function. Each frame of the webcam feed is taken and processed and an image is obtained. The face is detected in this image. Further, specific face landmarks are also identified and are marked. The image is further processed by implementing the functions that are defined earlier for finding the contour, masking the eye ball coordinates and identifying the direction of the eye ball. The direction of the eye ball is displayed as output on the live webcam feed as well as recorded in the log file.

Face Spoofing

Face spoofing is a practice in which instead of a real person being in front of a webcam, a photo of the person from a printed source or an electronic gadget or a video of the person is placed in front of the webcam. This process of identifying if there is a real person in front of the webcam or a photo or video of the person is called as face spoofing detection. The first step would be to import the python files that consists of functions for face detection. A function for calculating the histogram of the RGB image is defined. The image in the form of numpy array is passed as input. The histogram array is built by using OpenCV's calcHist function. The histogram array is converted to a numpy array and is returned as the output. The face detection model is stored in a variable.

The webcam is started using the OpenCV's VideoCapture function. Each frame of the webcam feed is taken and processed and an image is obtained. The face is detected in this image. Colour conversions are performed on the obtained image by using OpenCV's cvtColor function. The histograms are calculated for the two-colour space conversions. Now further calculations are done and the probability of the image being false is calculated. If the probability value if greater than or equal to 0.7, then, the output is displayed as a fake image, else, the output is displayed as a real image. In case, if the output displayed is a fake image, the output is stored in the log file.

Figure 3- A flow Diagram Describing the various Steps involved in the Process of Face Spoofing Detection



Identifying the Presence of more than One Person and Mobile Phone

Identification of the presence of more than one person and the presence of mobile phone is a very important feature in the webcam proctoring process. This can be implemented using a YOLOv3 model that is built using pretrained YOLOv3 weights. A function for loading the darknet weights and to decide whether batch normalization should be applied or not is defined. The weights of the YOLOv3 layers are defined in an array. Further, based on the weights, decision is made on whether batch normalization is required or not. A function that processes the image and identifies the various objects in the image is defined. The image, the YOLOv3 predictions and the list containing various classes of objects are passed as the input. The prediction along with image are returned as the output. A function to define a single Darknet Convolutional Layer is defined. The number of filters in the Convolutional Layer, the size of the kernel in the Convolutional Layer, the Convolutional Layer strides and the variable that indicates about whether to use custom batch normalization layer or not is passed as the input. The image is padded and the Convolutional Layer is defined and depending on the variable the custom batch normalization is either used or not used and it is passed as the output. A function to define a single DarkNet Residual Layer is defined. The number of filters in each Convolutional Layer is passed as the input and the single DarkNet Residual Layer is defined and returned back as the output. A function to define a single Darknet Block is defined. The number of filters in each Residual Layer and the number of Residual Layers in the block are passed as the input. The Darknet Block is defined based on the configuration given as input and is returned back as the output. A function to create the entire DarkNet is defined. This function makes use of the several sub functions that are defined as part of the DarkNet creation process and finally returns a keras model as the output. A function to define the YOLO Convolution Layer is defined. The number of filters for the Convolution Layer and the name of the Layer are passed as the input. The YOLO Convolution Layer is defined using the DarkNet functions and is returned as the output. A function that defines the output projections of the YOLOv3 is defined. The number of filters for the Convolutional Layer,

anchors, list of classes in the dataset and name of the layer are passed as the input. The function makes use of the Darknet functions and returns the prediction as the output. A function to get the bounding boxes from the network predictions is defined. The YOLO predictions, anchors and the list of classes in the dataset is passed as the input. The class probabilities, the bounding box, objectness and the prediction box are returned as the output. A function for creating a YOLOv3 model is defined. The function makes use of all the previously defined functions and creates a YOLOv3 model and returns the YOLOv3 model as the output. This function is called and the obtained model is stored in a variable.

YOLOv3 model can be utilised to classify more than 80 objects and is super-fast and nearly as accurate as single shot detector. It has fifty-three convolutional layers with each of them followed by a batch normalization layer and a leaky RELU activation. Anchor boxes help the model to specialize better. Consider an example of a person who is standing and a car. A person would require a tall box while the car would require a fat box. This is achieved through different size anchor boxes. This means that each of the objects will have more than just one bounding box. To decide which bounding box is kept non-maximal suppression is employed. Anchor boxes also help when different objects have their centres at the same place in predicting both the objects.

The webcam is started using the OpenCV's VideoCapture function. Each frame of the webcam feed is taken and processed and an image is obtained. The image obtained is resized to a size that YOLOv3 model supports. This image is processed based on the model and the boxes of the objects, the prediction score of each class and the number of objects of each class are passed as the input. Based on the obtained data, if the count of the class 'person' is more than one, more than one person is printed as output, if the count of the class 'person' is zero, no person is printed as output and if the class 'cell phone' is detected, mobile phone detected is printed as the output. All these outputs are also stored in the log file.

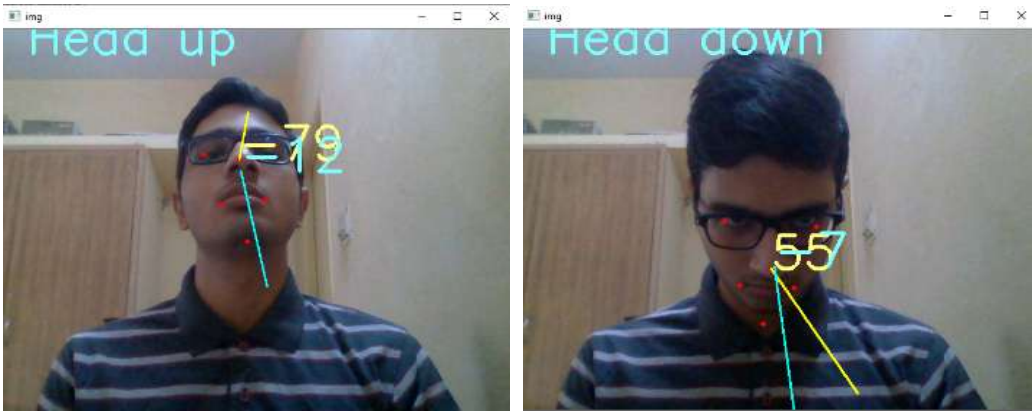
4. Results

Head Pose Estimation

On execution of the code developed for Head Pose Estimation, a window opens and starts capturing real time webcam feed. Each frame from the real time feed is processed and the output is estimated based on the value of the angles. The possible output values are 'Head Up', 'Head Down', 'Head Left' and 'Head Right'. All the output values that are obtained are also recorded and stored as

a log file. The frame rate of the webcam feed is subject to the GPU of the machine on which the code is executed.

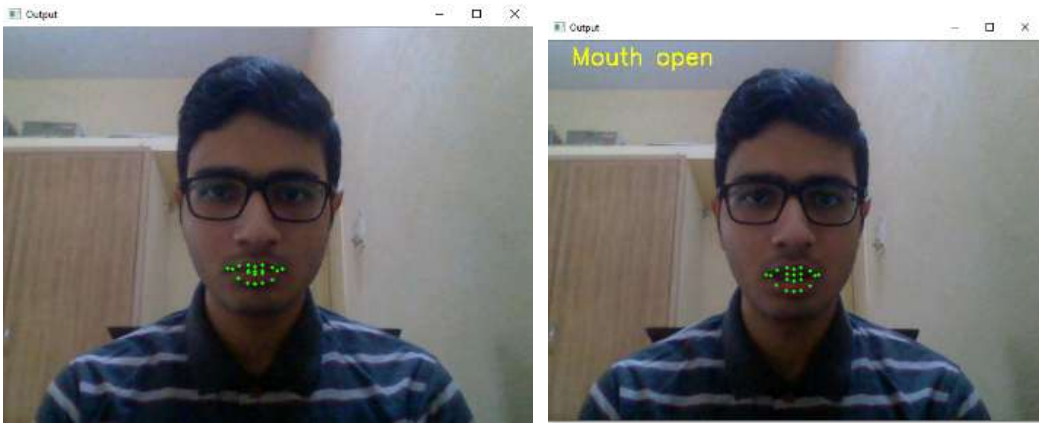
Figure 4- (A) Output for Head Pose Estimation when Head Posture is Upward. (B) Output for Head Pose Estimation when head Posture is Downward



Mouth Opening Detection

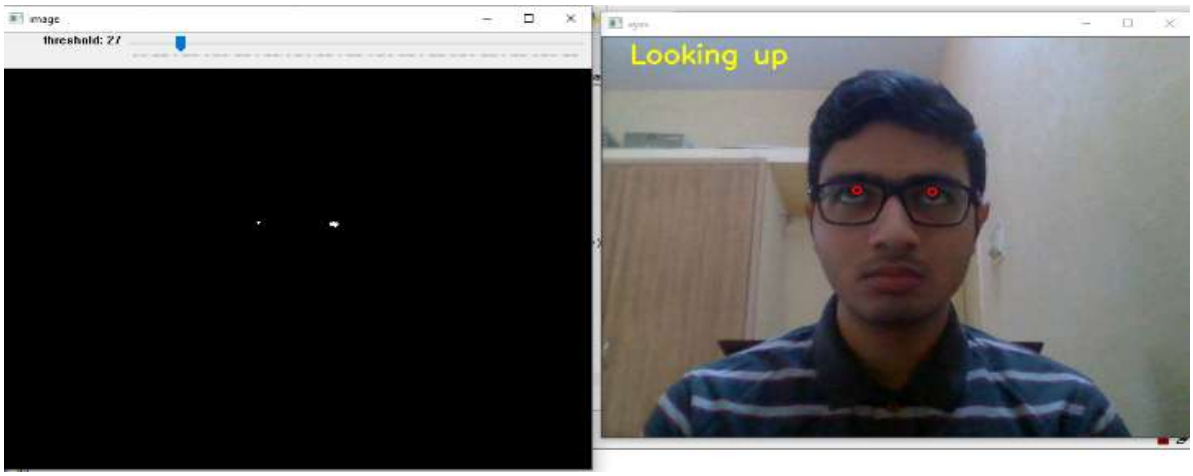
On execution of the code developed for Mouth Opening Detection, a window opens and starts capturing real time webcam feed. Each frame from the real time feed is processed and the results are estimated by comparing the default landmark coordinates of a closed mouth and the landmark coordinates of the mouth in the processed image. The deviations in the default values and the values of the image will result in the output being shown as 'Mouth Open'. All the output values that are obtained are also recorded and stored as a log file. The frame rate of the webcam feed is subject to the GPU of the machine on which the code is executed.

Figure 5- (A) Output for Mouth Opening Detection when Mouth is Closed. (B) Output for Mouth Opening Detection when Mouth is Open



Eye Tracking

Figure 6- Output for Eye Tracking when the Eyes are looking Upwards and Threshold Value is Set to 27

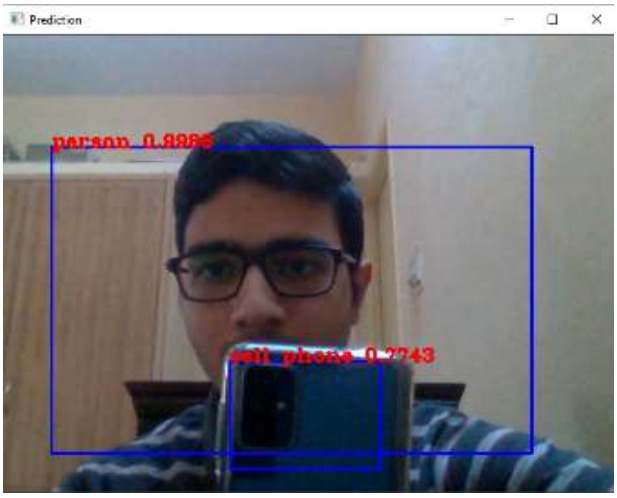


On execution of the code developed for Eye Tracking, a window opens and starts capturing real time webcam feed. Each frame from the real time feed is processed. The landmark coordinates for various positions of the eye are estimated. The image that is obtained is converted to threshold image and processed. By comparing the default values of the eye coordinates and the coordinates of the eye in the obtained image, the result is decided as 'Looking Left', 'Looking Right' or 'Looking Up' and the output is displayed. All the output values that are obtained are also recorded and stored as a log file. The frame rate of the webcam feed is subject to the GPU of the machine on which the code is executed.

Identifying the Presence of more than One Person and Mobile Phone

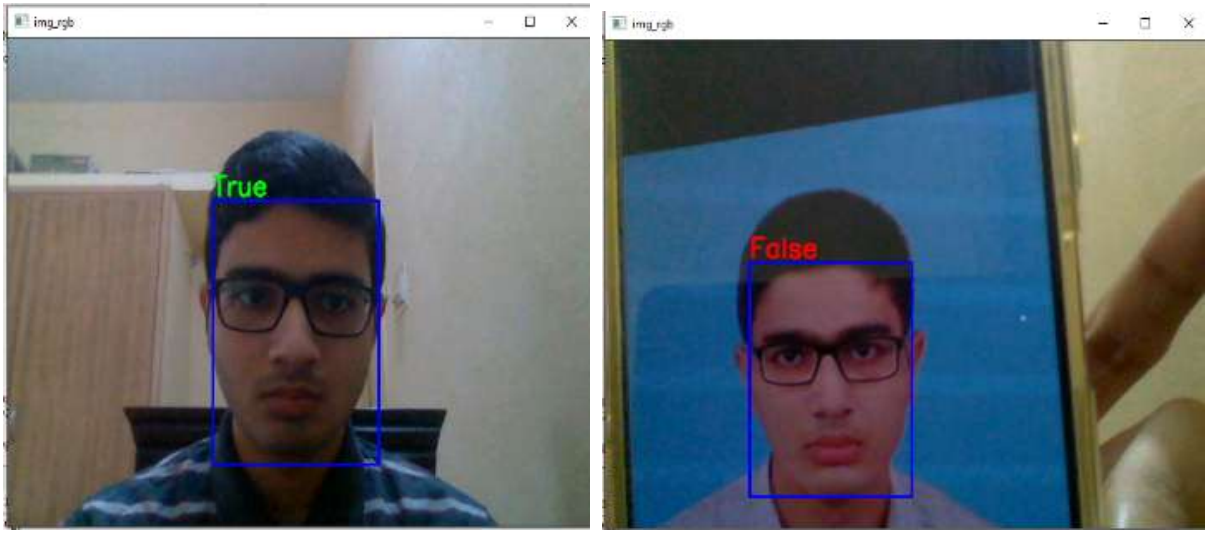
On execution of the code developed for identifying the presence of more than one person and mobile phone, initially with the help for YOLOv3 weights, the YOLOv3 model is built using DenseNet configurations. Further, a window opens and starts capturing real time webcam feed. Each frame from the real time feed is processed. Each frame or image is processed using the YOLO model and the objects in the image are identified and labelled. Further, the count of each object in the image is calculated. Based on these values the output will be displayed as 'No person detected', 'More than one person detected', 'Mobile Phone detected' or no output will be displayed. All the output values that are obtained are also recorded and stored as a log file. The frame rate of the webcam feed is subject to the GPU of the machine on which the code is executed.

Figure 7- Output for Identifying the Presence of more than One Person when there is a Person and a Mobile Phone in Front of the Webcam



Face Spoofing

Figure 8- (A) Output for Face Spoofing Detection when an Actual Person is in Front of the Webcam. (B) Output for Face Spoofing Detection when a Photo of a Person is shown in Front of the Webcam



On execution of the code developed for Face Spoofing Detection, a window opens and starts capturing real time webcam feed. Each frame from the real time feed is processed. All these images undergo colour conversions and their histograms are calculated. These values are processed and compared with the help of the Face Spoofing Model. Suppose the model returns a probability value greater than or equal to 0.7, the result is calculated as 'False' and displayed. Otherwise, 'True' is displayed. All the output values that are obtained are also recorded and stored as a log file. The frame rate of the webcam feed is subject to the GPU of the machine on which the code is executed.

Log File

On execution of the code for all the developed modules, the output that is displayed is stored in a .txt file along with the date and time. This ensures that the user can keep track of the various activities that took place in front of the webcam.

Figure 9- Screenshot of the Text File where the Results obtained are Stored along with Date and Time

5. Conclusion and Future Work

In this project, several features for webcam proctoring were developed. Initially, a model based on Extra Trees Classifier was built for Face Spoofing Detection and model based on Convolutional Neural Networks was built for Face Landmarks Detection. Using the Face Landmarks Detection model along with OpenCV, various features such as Head Pose Estimation, Mouth Opening Detector and Eye Tracker for live webcam feed were implemented. Using Face Spoofing Detection model along with OpenCV, face spoofing detection for webcam feed was implemented. A YOLOv3 model was built using YOLOv3 weights and this model was used along with OpenCV to find the number of people in the live webcam feed and also to detect the presence of mobile phone. All the outputs obtained was also automatically recorded into a log file.

There is always a possibility for building better models based on better datasets which can improve the performance in the actual use. Also, the field of Artificial Intelligence is constantly advancing. There is always a scope for better ways to implement these features through research. It is

possible to build models with newer or more advanced Neural Network configurations. With better hardware capability, more accurate models can be built by training them for a greater number of epochs. Research and study can be done by comparing various models built on same dataset and identifying the more accurate model. There is also scope for improving the frame rate of the live webcam feed that is obtained using OpenCV.

References

Khan, M., Chakraborty, S., Astya, R., & Khepra, S. (2019). Face Detection and Recognition Using OpenCV. *In International Conference on Computing, Communication, and Intelligent Systems (ICCCIS),* 116-119.

Goyal, K., Agarwal, K., & Kumar, R. (2017). Face detection and tracking: Using OpenCV. *In International conference of Electronics, Communication and Aerospace Technology (ICECA), 1,* 474-478.

Chandan, G., Jain, A., & Jain, H. (2018). Real time object detection and tracking using Deep Learning and OpenCV. *In International Conference on Inventive Research in Computing Applications (ICIRCA),* 1305-1308.

Ren, X., Ding, J., Sun, J., & Sui, Q. (2017). Face modeling process based on Dlib. *In Chinese Automation Congress (CAC),* 1969-1972.

Felea, L.I., Florea, L., Florea, C., & An, C.V. (2018). Head Pose Estimation Using Deep Architectures. *In International Conference on Communications (COMM),* 505-508.

Vishnu Raj R.S., Atithi Narayan S., & Kamal Bijlani, K. (2015). Heuristic based online proctoring system. *IEEE 15th International Conference on Advanced Learning Technologies.*

Kumbhar, P.Y., Attaullah, M., Dhere, S., & Hipparagi, S. (2017). Real time face detection and tracking using OpenCV. *International Journal for Research in Emerging Science and Technology, 4*(4).

Chen, H., Chen, Y., Tian, X., & Jiang, R. (2019). A cascade face spoofing detector based on face anti-spoofing R-CNN and improved Retinex LBP. *IEEE Access, 7,* 170116-170133.

Daniel, N., & Anitha, A. (2018). A Study on Recent Trends in Face Spoofing Detection Techniques. *In 3rd International Conference on Inventive Computation Technologies (ICICT),* 583-586.

Simanjuntak, G.D., Ramadhani, K.N., & Arifianto, A. (2019). Face Spoofing Detection using Color Distortion Features and Principal Component Analysis. *In 7th International Conference on Information and Communication Technology (ICoICT),* 1-5.

Wen, D., Han, H., & Jain, A.K. (2015). Face spoof detection with image distortion analysis. *IEEE Transactions on Information Forensics and Security, 10*(4), 746-761.

Liu, X., Liang, W., Wang, Y., Li, S., & Pei, M. (2016). 3D head pose estimation with convolutional neural network trained on synthetic images. *In IEEE international conference on image processing (ICIP),* 1289-1293.

Liu, L., Liu, J., & Huang, Z. (2017). Human-Eye Tracking and Location Algorithm Based on AdaBoost-STC and RF. *In 2nd International Conference on Cybernetics, Robotics and Control (CRC)*, 180-183.

Lu, Y., Zhang, L., & Xie, W. (2020). YOLO-compact: An Efficient YOLO Network for Single Category Real-time Object Detection. *In Chinese Control and Decision Conference (CCDC)*, 1931-1936.

Atoum, Y., Chen, L., Liu, A.X., Hsu, S.D., & Liu, X. (2017). Automated online exam proctoring. *IEEE Transactions on Multimedia, 19*(7), 1609-1624.

Prathish, S., & Bijlani, K. (2016). An intelligent system for online exam monitoring. *In International Conference on Information Science (ICIS)*, 138-143.