Research Question: How can a blur detection tool be effectively developed using the Haar Cascade classifier, leveraging OpenCV, NumPy and PyQt5 for efficient real-time image processing and user interface implementation?

Author: Siddhant Ray

1202, Lilium, Vasant Oasis, Makwana Road, Marol, Andheri (E), Mumbai - 400059

---***---

Abstract - *This research investigates the development of a blur detection tool utilizing the **Haar Cascade classifier**, a machine learning object detection method, implemented with OpenCV, NumPy, and PyQt5 libraries. The Haar Cascade classifier, a machine learning object detection method, is usually employed for real-time face detection activities. **OpenCV** is a robust library for computer vision tasks which is used for image processing and camera interfacing. **NumPy** library efficiently handles the numerical operations on image data. **PyQt5** provides a flexible graphical user-interface for seamless interaction. The methodology involves capturing live video frames from a Logitech 720p webcam using OpenCV detecting faces within these frames using the Haar Cascade classifier, and subsequently analyzing the sharpness of detected faces to determine blur levels. The Laplacian operator is applied to compute the variance, serving as the primary metric for blur detection. A high variance indicates a clear image, while a low variance indicates blurriness. The PyQt5 library entails several imports and enhances the usability of the tool by displaying the real-time video feed, and overlaying detection results, such as bounding boxes and update messages. The integration of the PyQt5 library allows for an interactive UI / UX design, enabling users to easily observe the blur detection outcomes. This research demonstrates the optimization and effective usage of these libraries to create a real-time blur detection system. This study provides a foundational framework for developing advanced image processing tools with practical real-world applications.*

Key Words: **Haar Cascade Classifier, OpenCV, Numpy, PyQt5, Operating System, Laplacian Variance, Blur detection, graphical user interface libraries, ellipse, bounding box, threshold, golden ratio**

1.INTRODUCTION

In the realm of digital image and video processing, detecting and mitigating image blur is crucial for ensuring high-quality visual information. This research explores the implications of a blur detection tool using the Haar Cascade classifier, integrated with OpenCV and NumPy for efficient image and video analysis. Haar Cascade classifiers facilitate rapid and precise face detection, acting as a medium for subsequent blur analysis. By integrating the Laplacian variance, the tool quantifies image sharpness, thereby easily differentiating between clear and blur images. The incorporation of PyQt5 enhances user interaction, providing accurate real-time feedback and a responsive user interaction. This study aims to enhance image quality across various applications through blur detection.

2. LIBRARIES INCORPORATED AND THEIR FUNCTIONS WITH RESPECT TO THE BLUR DETECTION TOOL

- **os** - The '**Operating System**' library is used primarily for file path operations, such as expanding the user's home directory to save captured photos
- **cv2** - The '**cv2**' library is a part of 'OpenCV (Open Source Computer Vision library)' which is used for computer vision in the form of video capture, image processing, face detection, and drawing bounding boxes and ellipses
- **sys** - The '**sys**' library provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter
- **numpy** - The '**numpy**' library is used for array manipulation and mathematical operations in the form of calculating face area and intersection area
- **PyQt5** - The '**PyQt5**' library is a set of Python bindings for Qt libraries, used to create and manage graphical user interfaces (GUIs). Here, '**QApplication**' manages the application's control flow and main settings, '**QMainWindow**' creates the main window for the application, '**QTimer**' periodically updates the frames, '**QImage**' and '**QPixmap**' are used to convert (format conversion) OpenCV images into a format that can be displayed in PyQt5 widgets.

3. HAAR CASCADE CLASSIFIER

The **Haar Cascade Classifier** is a machine learning computer-vision based approach introduced in 2001 by Intel Corporation (Gary Bradski and Adrian Kaehler). It is particularly used in face detection via a webcam, but it can be trained for other types of object detection as well. This classifier uses a cascade function to identify objects in an image based on features. These features are derived from Haar wavelets (simple rectangular patterns of different pixel intensities).

4. WORKING PRINCIPLE OF HAAR CASCADE CLASSIFIER

Below is the step by step methodology for the **working principle** of Haar Cascade Classifier:

1) **Integral Image Representation**: This is a quick way to calculate the sum of pixel values within a rectangular region of an image which helps to significantly speed up the computation of Haar Cascade features. It is a unique kind of technology where each pixel value is the cumulative sum of pixel values in any rectangular subset of the image.
2) **Haar Features**: These are simple rectangular features that can be calculated using the integral image. These features are the key point in capturing the difference in intensity between adjacent regions, which is particularly useful for detecting edges and changes in an image.
3) **AdaBoost Training**: Adaptive Boost Training is used to select the most relevant features among a large set and to train a cascade of classifiers. This training process involves selecting a small number of significant features from a large pool, which can be used to classify differences between objects (faces) and non-objects (background / interference). This pool of weak classifiers are then combined to form a strong classifier.
4) **Cascade of Classifiers**: The cascade architecture is a series of increasingly complex classifiers to quickly eliminate negative windows, while more complex classifiers are used to ensure higher accuracy in detection. Each stage of Haar Cascade applies a series of Haar features to the input images in the form of parameters. If the segment passes through all these stages, only then is it considered to contain the object of interest (face).

5. APPLICATION OF HAAR CASCADE CLASSIFIER IN THE BLUR DETECTOR

<u>Part of code</u>: face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

Function: This is a constructor method which initializes / loads the pre-trained Haar Cascade classifier.

- '**OpenCV**' is imported as '**cv2**'
- '**CascadeClassifier**' is used for object detection
- **'haarcascade_frontalface_default.xml'** is the unique XML file containing the pre-trained model and was incorporated as part of installing Haar Cascade through the pip function
- **cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'** concantenates (joining two or more strings) the directory path and XML file name to form the full path to the pre-trained model. The '+' operator is used for the concatenation here

<u>Part of Code:</u> faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

Function:

- '**gray**' is used to grayscale the image where the face is to be detected
- '**scaleFactor=1.1**' is a parameter specifying how much the image size is reduced at each image scale and helps in creating a scale pyramid
- '**minNeighbors=5**' is a parameter that defines how many neighbors each candidate rectangle should have to retain it. A higher value results in fewer detection but higher quality, hence we have chosen a median value
- '**minSize=(30, 30)**' sets the minimum possible object size. Objects smaller than this size are ignored and the function is not called if the objects are smaller than this size

6. OPENCV LIBRARY

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV is used in a wide array of real life activities such as image processing, video processing, face detection, object detection, motion tracking, so on and so forth. The syntax to install OpenCV in Python is:

pip install opencv-python

For additional functionality such as support for non-free algorithms or GUI support, to unlock more features, one can install OpenCV with the syntax:

pip install opencv-contrib-python

7. COMMON OPENCV SYNTAXES AND THEIR FUNCTIONS

- img = **cv2.imread**('image.jpg'): Reads an image from a file
- img = **cv2.imwrite**('output.jpg', img): Saves an image to a specified file
- img = **cv2.imshow**('Window Name', img): Displays images
- **cv2.waitKey**(0): Waits for a key event indefinitely
- **cv2.destroyAllWindows**(): Closes all OpenCV windows
- gray = **cv2.cvtColor**(img, cv2.COLOR_BGR2GRAY): Used for grayscaling, changes the entire representation of an image from colors to black and white
- resized_img = **cv2.resize**(img, (width, height)): Resizes an image to specified dimensions
- **cv2.rectangle**(img, (x, y), (x + w, y + h), (0, 255, 0), 2): Draws a rectangle on an image; parameters are coordinates of the top-left corner, bottom-right corner, color (in BGR format) and thickness
- **cv2.circle**(img, (x, y), radius, (0, 255, 0), 2): Draws a circle on an image; parameters include the center, radius, color and thickness
- vid = **cv2.VideoCapture**(0): Captures a video from a camera of video file; parameters contain camera index or the filename

8. APPLICATION OF OPENCV IN BLUR DETECTION TOOL

Part of Code: cv2.CascadeClassifier

Function: Used to initialize the Haar Cascade classifier using the pre-trained model for frontal face detection

Part of Code: cap = cv2.VideoCapture(0)

Function:

- Used to start the webcam; variable name is '**cap**'
- 0 represents the default camera of the system
- '**VideoCapture**' function is used to capture the video from the webcam

Part of Code: frame_width = int(cap.get(3)) frame_height = int(cap.get(4))

Function:

- These retrieve the height & width of the video frames captured from the webcam
- The '**cap**' here was the variable used to define the video capture in OpenCV
- cap.get(3) retrieves the **width** of video frames
- cap.get(4) retrieves the **height** of video frames
- The numbers of 3 and 4 are identified as constants defined in OpenCV's pre-trained model for height and width respectively
- '**CV_CAP_PROP_FRAME_WIDTH**' has the identifier 3
- '**CV_CAP_PROP_FRAME_HEIGHT**' has the identifier 4
- Essentially, '**cap.get(3)**' and '**cap.get(4)**' are short-cut ways of retrieving the frame's height and width respectively

Part of Code: golden_ratio = 1.618 oval_width = int(frame_width / golden_ratio) oval_height = int(frame_height / golden_ratio)

Function:

- Defined a 'golden-ratio' to 1.618 which is the most perfect size for an oval for an average human head to fit inside the oval
- '**oval_width = int(frame_width / golden_ratio)**' divides the frame width by the

golden ratio (1.618) to determine the width of the oval

- **oval_height = int(frame_height / golden_ratio)** divides the frame height by the golden ratio to determine the height of the oval
- '**int()**' function is used to typecast the result into an integer since pixel dimensions are typically in whole numbers

<u>Part of Code:</u> center_coordinates = (frame_width // 2, frame_height // 2)

Function:

- '**center_coordinates**' is a tuple which is immutable (cannot be modified)
- '**frame_width//2**' uses integer division to find the horizontal center of the frame (x-coordinate of the center of the frame)
- '**frame_height//2**' uses integer division to find the vertical center of the frame (y-coordinate of the center of the frame)

<u>Part of Code:</u> axes_length = (oval_width // 2, oval_height // 2)

Function:

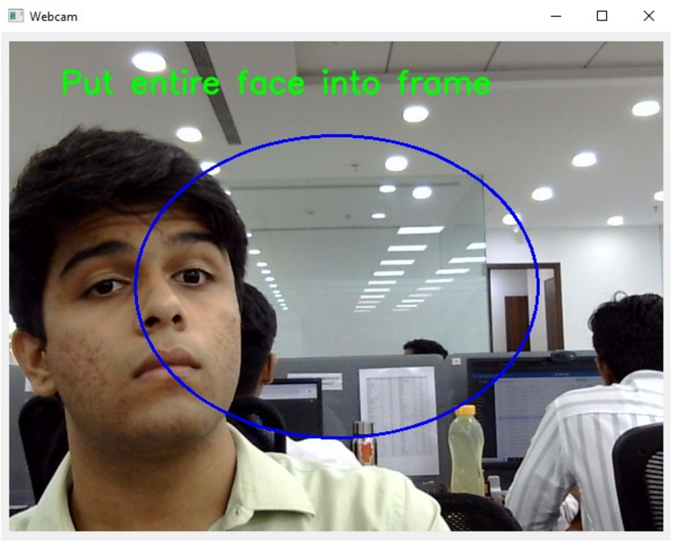
- '**oval_width//2**' divides the oval width by 2 to get the semi-major axis length
- '**oval_height//2**' divides the oval height by 2 to get the semi-minor axis length
- '**axes-length**' tuple contains the lengths of semi-major and semi-minor axes of the oval

<u>Part of Code:</u> 'color = (255, 0, 0) thickness = 2'

Function:

- '**color = (255, 0, 0)**' sets the color of the oval to blue. This follows the BGR format (Blue, Green Red). Here, a high value of the blue component increases the intensity of blue colour onto the ellipse / oval. Since the red and green component is 0, the color of the ellipse is purely blue without any presence of the red and green component. BGR format is followed by OpenCV instead of RGB since BGR has been followed for a very long time and it improves interoperability
- '**thickness = 2**' sets the thickness of the oval's outline to 2 pixels

Demonstration of making of the ellipse



BLUE COLOR ELLIPSE AND OUTLINE THICKNESS 2 PX

<u>Part of Code:</u> def create_oval_mask(frame_shape, center, axes):

Function:

- This line defines a function called '**create_oval_mask**' that takes three parameters
- <u>Parameter 1</u> is called '**frame_shape**' which is the expected shape of the video frame. It is typically a tuple representing the dimensions of the frame entailing the height, width, and channels of the image in the frame
- <u>Parameter 2</u> is called '**center**' which is a tuple representing the center coordinates (x,y) of the oval. This tuple is immutable (cannot be modified) and only accepts values generated by the system
- <u>Parameter 3</u> is called '**axes**' which is a tuple representing the lengths of the semi-major and semi-minor axes of the oval. This tuple too is immutable (cannot be modified) and only accepts values generated by the system

Part of Code: mask = np.zeros(frame_shape[:2], dtype=np.uint8)

Function:

- This line is used to create an empty binary mask
- '**np.zeros**' is a function from the '**NumPy**' library that creates a 1D array filled with zeros
- '**frame_shape[:2]**' is used to slice the 'frame_shape' tuple to get the first two elements, which are the height and width of the frame. This ignores the image channel dimensions (example: RGB) because the mask is a single-channel gray-scaled image
- '**d-type=np.uint8**' specifies the data type of the array elements. The '**np.uint8**' here refers to an 8-bit unsigned integer data type. '**uint8**' here stands for '**unsigned 8-bit integer**'. '**Unsigned**' means the values have to be non-negative (positive integers) from 0 to 255. The '**8**' here indicates it stores 8 bits (1 byte) of memory. Essentially, '**np.uint8**' stands for a positive 8 bit unsigned integer data type. The advantages of this are that it is memory efficient since it takes up only 8 bits per channel making it suitable for handling large images. Apart from this, this data type is extremely compatible since most image processing libraries and hardware expect image data in this format

Part of Code: cv2.ellipse(mask, center, axes, 0, 0, 360, 255, -1)

Function:

- '**cv2.ellipse**' is an OpenCV function that draws an ellipse on an image
- '**mask**' is the blank image created using '**np.zeros**'
- '**center**' is a tuple (x,y) representing the center coordinates of the ellipse
- '**axes**' is a tuple representing the lengths of the semi major and semi minor axes of the ellipse. The lengths are halved the get the complete width and height of the ellipse
- The first '**0**' represents the angle of rotation of the ellipse. A value of '**0**' means there is no rotation; the ellipse is aligned with the axes
- The '**0**' and '**360**' define the **start** and **end** angles of the ellipse respectively. Since the start angle is 0 and the end angle is 360, a full ellipse is drawn
- '**255**' here represents the color of the ellipse which corresponds to **white** naturally and corresponds to **blue** in a **multi-channel image**
- '**-1**' here represents the thickness of the ellipse's outline. The value of '**-1**' means the ellipse is filled. If a positive value were provided, it would specify the thickness of the ellipse's border

Part of Code: return mask

Function:

- '**return**' keyword: This keyword is used to **exit** a function and return a value
- This line returns the '**mask**' array, which contains the drawn ellipse
- The function '**create_oval_mask**' generates and returns this mask, which can be used for further image processing functions

9. CREATING A FUNCTION TO CHECK IF FACE IS IN OVAL AND ITS EXPLANATION

Part of Code: def is_face_in_oval(face, oval_mask, required_percentage=0.8):

Function:

- Here, the name of the function is '**is_face_in_oval**' which is an immutable tuple
- The '**face**' represents a part of the tuple representing the coordinates of the top-left corner (x,y), width (w) and height (h) of the detected face
- The '**oval_mask**' represents the formation of a binary mask image (same size as the frame) with an oval drawn on it
- The '**required_percentage=0.8**' forms a threshold of 80% / 0.8 representing the

minimum percentage of the face that must be inside the oval to consider it valid

<u>**Part of Code:**</u> x, y, w, h = face

Function:

- The purpose of this piece of code is to unpack the 'face' tuple into individual variables
- The '**x**' represents the <u>x-coordinate</u> of the top-left coordinate of the face
- The '**y**' represents the <u>y-coordinate</u> of the top-left coordinate of the face
- The '**w**' represents the <u>width</u> of the face
- The '**h**' represents the <u>height</u> of the face

<u>**Part of Code:**</u> face_mask = np.zeros_like(oval_mask)

Function:

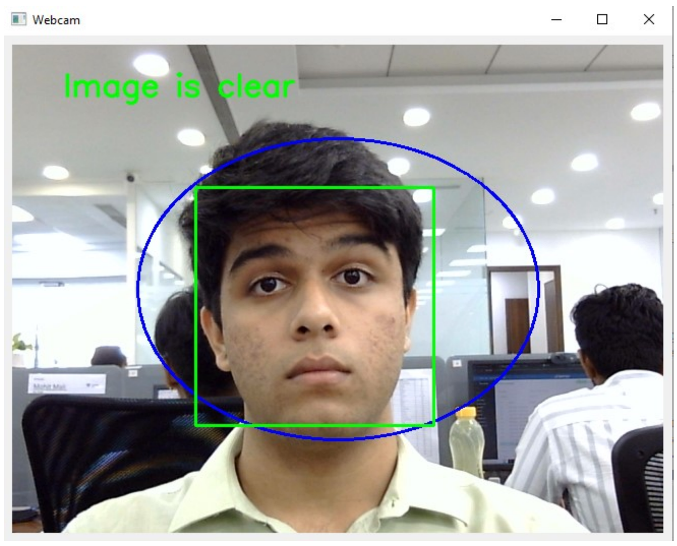
- The purpose of this piece of code is to create an invisible blank binary mask of the same size as '**oval_mask**'
- The '**face_mask**' is the variable which is defined to the blank binary mask
- The '**np.zeros_like(oval_mask)**' is a NumPy function that creates a 1D array of zeros with the same shape and type as the given array
- The created '**face_mask**' will be a binary image with all pixels initially set to zero

<u>**Part of Code:**</u> cv2.rectangle(face_mask, (x, y), (x + w, y + h), 255, -1)

Function:

- This feature draws a rectangle (bounding box) whenever a face is detected
- The '**face_mask**' is the image on which to draw the rectangle
- The '**(x,y)**' is the <u>top left corner</u> of the rectangle
- The '**(x+w, y+h)**' is the <u>bottom right corner</u> which is found by adding the width and height of the rectangle to the x-coordinate and y-coordinate of the top left corner respectively
- The '**255**' represents that the color of the grayscale (255) is <u>white</u>
- The '**-1**' represents the <u>thickness of the rectangle</u> and means that the area of the rectangle will be filled

<u>**Visual Demonstration of the bounding box when a face is detected**</u>



<u>**Part of Code:**</u> intersection = cv2.bitwise_and(oval_mask, face_mask)

Function:

- Performs a bit-wise AND (logic gates) operation between the 2 masks. This operation keeps only the pixels that are **white** (255) in both masks. Given below is the <u>**Truth table**</u> for a bit-wise **AND operation**

Bit A	Bit B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

- The '**cv2.bitwise_and**' contains two parameters: '**oval_mask**' which is the first binary mask with the oval and the '**face_mask**' which is the second binary mask with the face rectangle
- The '**intersection**' mask will have white pixels only in the area where the face rectangle **overlaps** with the oval

Part of Code: intersection_area = np.sum(intersection // 255)

Function:

- Calculates the area of intersection between the face rectangle and oval in terms of number of white pixels
- The ‘**intersection // 255**’ converts the intersection mask into a binary mask to ensure area calculations are simplified; AND, OR and XOR operations can be carried out smoothly; there is consistency in data representation
- The ‘**np.sum**’ **sums** up all the values to determine the area of the intersection

Part of Code: return intersection_area >= required_percentage * face_area

Function:

- Determines if the intersection area is at least the required percentage of the face area
- The ‘**required_percentage*face_area**’ calculates the minimum required area of the face that should be inside the oval
- The ‘**intersection_area>=**’ checks if the intersection area exceeds or at least meets this minimum required area
- The ‘**return**’ value is used here to give ‘**True**’ if the intersection area is greater than or equal to the required percentage 0.8 multiplied by the face area and will return ‘**False**’ if the output is otherwise (intersection area is lesser than the required percentage 0.8 multiplied by the face area)

10. PYQT5 LIBRARY

PyQt5 stands for Python Qt5 which is a set of Python bindings for Qt libraries, mainly used for creating GUIs. Qt is a powerful cross platform C++ library used to develop graphical user interfaces (GUIs) and multi-platform applications that run on various software and hardware applications such as Linux and Windows. PyQt5 is special since it aids cross-platform compatibility, provides a rich set of widgets, allows integration with Python, is up-to-date and compatible with modern GUI developments, uses a signal and slot mechanism for event handling making it easier to manage user interactions and interface components. Lastly, there is an extensive documentation dataset and a large community providing support and examples on how to effectively use PyQt5.

11. COMPONENTS OF PYQT5 LIBRARY

- **QtCore:** Core non-GUI classes used by other modules
- **QtGui:** Classes for window system integration, event handling, 2D graphics, basic imaging, fonts and text
- **QtWidgets:** Classes for creating desktop style GUIs
- **QtMultimedia:** Classes for audio, video, radio and camera functionality
- **QtNetwork:** Classes to make network programming easier and more portable
- **QtSql:** Classes for database integration using SQL
- **QtSvg:** Classes for displaying the contents of SVG (Scalable Vector Graphics)
- **QtOpenGL:** Classes for integrating OpenGL (Open Graphics Library) functionality

12. APPLICATION AND SETUP OF PYQT5 LIBRARY IN BLUR DETECTOR

APPLICATION:

Part of Code: from PyQt5.QtWidgets import QApplication, QLabel, QMainWindow, QVBoxLayout, QWidget

Function:

- The ‘**QApplication**’ manages application wide-resources and settings
- The ‘**QLabel**’ displays text or images

- The '**QMainWindow**' is the main application window providing a framework for building a main user interface
- The '**QBoxLayout**' is used to arrange widgets vertically
- The '**QWidget**' provides the base class for all UI objects

Part of Code: from PyQt5.QtGui import QImage, QPixmap

Function:

- The **'Q_Image**' represents an image in a format that can be manipulated at the pixel level
- The '**QPixmap**' is optimized to display images on the screen

Part of Code: from PyQt5.QtCore import QTimer, Qt

Function:

- The '**QTimer**' fires timeout signals at specified intervals, used for repetitive tasks
- The '**Qt**' contains miscellaneous identifiers used throughout PyQt5 such as alignment and event types

SETUP:

Part of Code: class App(QMainWindow):

Function: Defines a new class called '**App**' that inherits from the '**QMainWindow'**, which provides a main application window

Part of Code: def __init__(self):

Function: A constructor method for initializing the '**App**' class. 'self' refers to the instance of 'QMainWindow' class representing the main window of our application

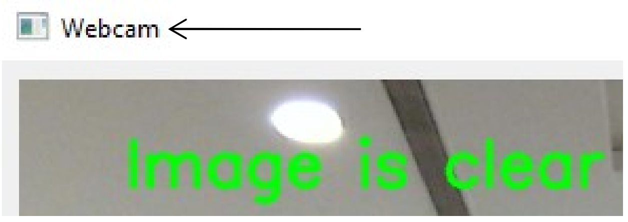
Part of Code: super().__init__()

Function: Calls the constructor of the '**QMainWindow**' class using the '**super**' keyword to ensure proper initialization

Part of Code: self.setWindowTitle("Webcam")

Function: Sets the title of the main window. 'self' refers to the instance of 'QMainWindow' class representing the main window of our application

Visual Representation:



Part of Code: self.setGeometry(100, 100, frame_width, frame_height)

Function:

- Sets the size and position of the window
- The '**frame_width**' sets the width of the frame
- The '**frame_height**' sets the height of the frame
- The '**100, 100**' represents the x-coordinate and the y-coordinate of the top left corner of the screen which mean the window will start 100 pixels from the left of the screen and 100 pixels from the top of the screen

Part of Code: self.central_widget = QWidget(self)

Function:

- Creates a central widget which will contain other widgets
- '**self**' refers to the instance of '**QMainWindow**' class representing the main window of our application
- In the context of machine learning and the model of blur detector, a **widget** refers to an interactive tool or an interface element

- A widget aids interactive visualization, parameter tuning, data exploration, model comparison and provide a smooth user interface experience

Part of Code: self.setCentralWidget(self.central_widget)

Function:

- Sets '**central_widget**' as the central widget for the main window
- '**self**' refers to the instance of '**QMainWindow**' class representing the main window of our application
- '**.setCentralWidget**' is a method provided by the '**QMainWindow**' class to set a central widget for the main window
- '**self.central_widget**' is the widget we have set as the central content of our main window

Part of Code: self.layout = QVBoxLayout(self.central_widget)

Function:

- '**layout**' is an instance variable that holds the layout manager for the central widget
- '**QVBoxLayout**' is a layout manager provided by PyQt that arranges widgets vertically
- '**self.central_widget**' is the central widget of the main window, which we had defined earlier in our code and the layout manager is being applied to this widget

Part of Code: self.label = QLabel(self)

Function:

- Used to create a label widget in PyQt
- Labels are commonly used to display text or images within a graphical user interface
- '**label**' is the instance variable that holds the QLabel widget
- '**QLabel**' is the widget provided by PyQt that can display text or images
- '**self**' (inside the QLabel constructor) specifies that the label widget will be a child of the '**self**' object which means it will be placed in the main window or the parent widget specified by '**self**'

Part of Code: self.layout.addWidget(self.label)

Function:

- Adds the '**label**' to the layout

Part of Code: self.frame_timer = QTimer()

Function:

- '**frame_timer**' is an instance variable holding the QTimer object
- '**QTimer**' is a class provided by PyQt that allows the user to create a timer object. These timers emit signals at specified intervals, which can be used to trigger functions periodically

Part of Code: self.frame_timer.timeout.connect(self.update_frame)

Function:

- '**frame_timer**' is the QTimer object that was previously created
- '**timeout**' is a signal emitted by the QTimer object when the timer runs out after the specified intervals elapses
- '**connect**' is a method used to connect a signal to a slot (a function or method to be executed when the signal is emitted)
- '**self.update_frame**' is the method that will be called each time the timeout signal is emitted by the '**QTimer**'

Part of Code: self.frame_timer.start(33)

Function:

- '**frame_timer**' is the **QTimer** object previously created
- '**start**' is a method of the **QTimer** class that starts the timer
- '**33**' is the argument that specifies the interval in milliseconds, in this scenario, 33 milliseconds

<u>**Part of Code:**</u> self.current_frame = None

Function:

- ‘**current_frame**’ is an instance variable that will hold the latest frame from the webcam
- ‘**None**’ initializes the ‘**current_frame**’ to ‘**None**’ indicating that no frame has been captured yet

<u>**Part of Code:**</u> self.face_position = None

Function:

- ‘**face_position**’ is an instance variable that will hold the coordinates or bounding box of the detected face
- ‘**None**’ initializes the ‘**face_position**’ to ‘**None**’, indicating that no face has been detected yet

<u>**Part of Code:**</u> self.blur_text = ""

Function:

- ‘**blur_text**’ is an instance variable holding text messages regarding the clarity of the images. These messages include messages about whether an image is clear, blurry, or the face needs to be fully fitted into the frame
- ‘**=””**’ initializes the ‘**blur_text**’ with an empty string. This means, initially, when the code is run, there is no message stored in ‘**blur_text**’

13. UPDATE FRAME METHOD

The ‘**update_frame**’ method captures a frame from the webcam, processes it to detect faces, and assesses image clarity. It starts capturing and copying the frame, then creates an oval mask and draws an ellipse on the frame. The frame is then converted to grayscale for face detection using Haar Cascade classifier. For each detected face, it checks if the face is within the oval mask, preprocesses the face region, and calculates the variance of the Laplacian to determine if the image is blurred. It updates ‘**self.blur_text**’ with the result, draws a bounding box around the detected face, and overlays the blur status text on the frame. Finally, the frame is converted to ‘**QImage**’ and displayed in a **label widget**.

<u>Exploring how this works in even more detail:</u>

<u>**Part of Code:**</u> def update_frame(self)

Function:

- This line defines the method ‘**update_frame**’ as part of a class
- It does not take any parameters other than ‘**self**’, which refers to the instance of the class

<u>**Part of Code:**</u> ret, frame = cap.read()

Function:

- ‘**ret**’ is a boolean value that indicates if a frame was captured successfully or not. If the frame was captured correctly, ‘**ret**’ is set to **True** and if the frame was not captured correctly, ‘**ret**’ is set to **False**
- ‘**frame**’ is a variable that holds the captured image data. This is typically a NumPy array containing the pixel values of the image captured from the webcam
- ‘**cap.read()**’ is a method provided by OpenCV’s ‘**VideoCapture**’ class. This method attempts to capture a frame from the webcam / video stream associated with the ‘**cap**’ object

<u>**Part of Code:**</u> if not ret: return

Function:

- ‘**if_not_ret**’ condition checks whether the ‘**ret**’ variable is ‘**False**’
- If ‘**ret**’ is **False**, ‘**not_ret**’ becomes **True**
- ‘**return**’ causes the method to exit immediately, returning control to the caller without executing any further code within the method
- Essentially, this is a guard clause which ensures the method exits early if the frame capture fails, preventing any further processing of the frame when there is no valid image data to work with

Part of Code: self.current_frame = frame.copy()

Function:

- '**current_frame**' is an instance variable of the class that will store the copied frame. This variable is used to retain the current frame's data for further processing and display
- '**frame.copy()**' creates a deep copy of the '**frame**' array. The '**frame**' variable contains the image data captured from the webcam, and calling '**copy()**' creates an independent duplicate of this data

Part of Code: oval_mask = create_oval_mask(frame.shape, center_coordinates, axes_length)

Function:

- '**oval_mask**' is a variable that will store the binary mask created by the 'create_oval_function'. This mask is typically a binary image where the oval area is marked
- '**create_oval_mask (.....)**' is a function created to call '**create_oval_mask**', which is a custom function designed to generate an oval-shaped mask
- '**frame_shape**' is an attribute of the '**frame**' variable that provides the dimensions of the captured frame. '**frame_shape**' returns a tuple containing the height, width, and number of color channels of the frame '**(height, width, channels)**'
- '**center_coordinates**' is an argument specifying the center coordinates of the oval to be created. It is usually a tuple (x,y) indicating the position of the oval's center
- '**axes_length**' is an argument specifying the lengths of the minor and major axes of the oval. It is a tuple (major_axis_length, minor_axis_length)

Part of Code: face_detected = True

Function:

- Sets a flag indicating a face has been detected in the oval
- '**face_detected**' is a boolean variable used to track if a face is detected

Part of Code: face_roi = self.current_frame[y:y + h, x:x + w]

Function:

- This line of code extracts the region of interest (ROI) of the face from the current frame
- '**y:y+h**' represents the range of rows to select (height of the face)
- '**x:x+w**' represents the range of columns to select (width of the face)

Part of Code: face_roi_gray = cv2.cvtColor(face_roi, cv2.COLOR_BGR2GRAY)

Function:

- The purpose of this line of code is to convert the face ROI to grayscale
- Cv2.cvtColor is the OpenCV function to convert an image from BGR format to grayscale
- '**face_roi**' represents the extracted face region
- '**cv2.COLOR_BGR2GRAY**' is the conversion code for BGR to grayscale

Part of Code: laplacian_var = cv2.Laplacian(face_roi_gray, cv2.CV_64F).var()

Function:

- Calculates the variance of the Laplacian o the grayscale face ROI to assess the blurriness
- The '**cv2.Laplacian(.....)**'syntax is the OpenCV syntax to apply the Laplacian filter to detect edges
- '**face_roi_gray**' is used to grayscale the image

- '**cv2.CV_64F**' specifies the data depth for the output image
- '**.var**' computes the variance of the Laplacian-filtered image

14. CODE LOGIC FOR BLUR / NON-BLUR DETECTION THROUGH LAPLACIAN VARIANCE

<u>**Part of Code**</u>: if laplacian_var < 100:
self.blur_text = "Image is blurred"
else:
self.blur_text = "Image is clear"

Function:

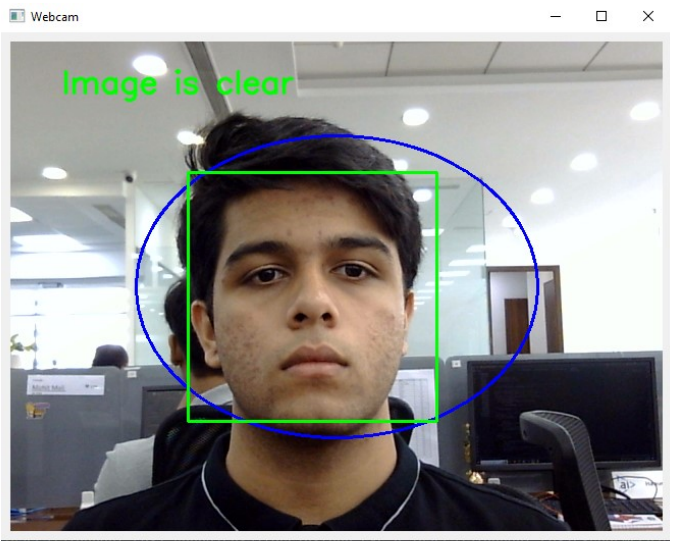
- '**if**' is a condition to check if the following condition is '**True**' or '**False**'
- '**laplacian_var<100**' condition checks if the variable '**laplacian_var**' (which represents the Laplacian variance of the image) is less than 100
- In Laplacian variance, the value of 100 is used to differentiate between clear and blurry images. If the value is less than 100, it is considered to be a **blurry** image
- If the value of the Laplacian Variance is greater than 100, it is considered a **clear** image
- '**self.blur_text = "Image is blurred"**': Here the '**blur_text**' is an instance variable of the class that stores a message about the clarity of the image; '**"Image is blurred"**' refers to a string message indicating that the image is blurred. It indicates that the image has been detected blurry based on the current analysis
- '**else:**
  self.blur_text = "Image is clear"': This piece of code is executed if the image is not considered blurred i.e. variance is above the threshold; '**"Image is clear"**' refers to a string message indicating that the image has been detected clear based on the current analysis

<u>**Part of Code**</u>: if not face_detected:
self.blur_text = "Put entire face into frame"

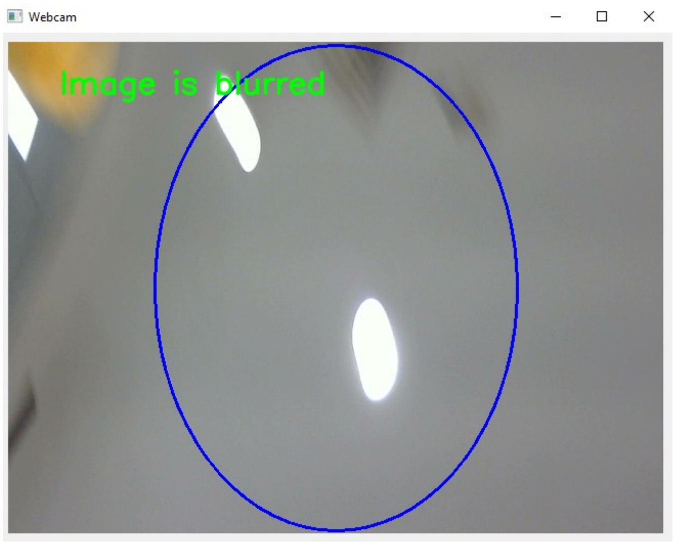
Function:

- The '**if not**' is a condition here which checks if the face has been detected or not. It will execute its block of code only if a face has **not** been detected
- The '**self.blur_text = "Put entire face into frame"**' indicates a string message indicating that the entire face is not in the frame and hence prints out a text message instructing the user to put his / her entire face into the frame

15. VISUAL REPRESENTATION OF IDENTIFICATION BETWEEN BLUR / NON-BLUR LIVE ON THE LOGITECH 720P WEBCAM

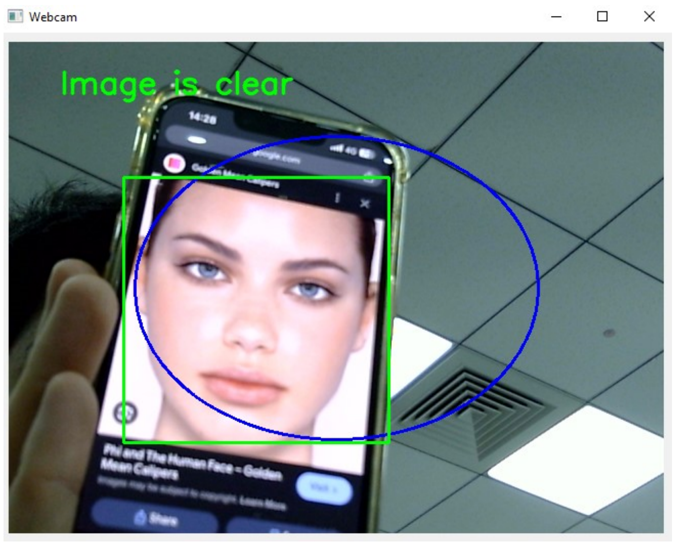


<u>**CLEAR IMAGE (LAPLACIAN VARIANCE > 100)**</u>

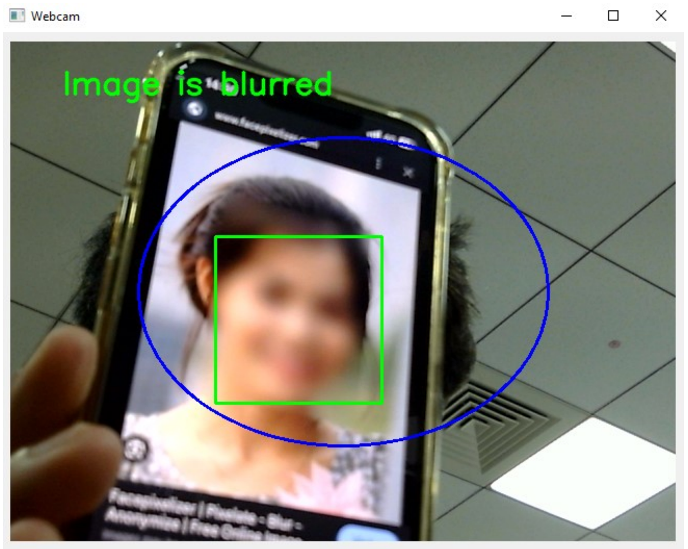


<u>**BLUR IMAGE (LAPLACIAN VARIANCE < 100)**</u>

16. VISUAL REPRESENTATION OF IDENTIFICATION BETWEEN BLUR / NON-BLUR WHEN SHOWN A FACE FROM A MOBILE PHONE ON THE LOGITECH 720P WEBCAM



CLEAR FACE IMAGE (LAPLACIAN VARIANCE > 100)



BLUR FACE IMAGE (LAPLACIAN VARIANCE < 100)

17. CONCLUSIONS

In conclusion, in the realm of digital image processing, the development of an efficient and reliable blur detection tool is paramount. This research explored the creation of a blur detector using Laplacian variance method, implemented within the OpenCV framework. Through detailed experimentation and analysis, the tool demonstrated its capability to accurately differentiate between clear and blurred images by analysis variances in pixel intensities.

The implementation leverages Haar Cascade classifier for robust face detection and the Laplacian operator to compute the variance, serving as an indicator of image sharpness. By setting an appropriate threshold for the Laplacian variance, the tool effectively identified blurred regions, facilitating various applications such as image quality assessment, automated photography correction, and real-time video stream enhancement.

The use of PyQt5 for the GUI (Graphical User Interface) provided a seamless and interactive user experience, enabling real-time feedback on the image clarity status. This integration highlights the importance of combining advanced image processing techniques with user friendly interfaces create practical and accessible tools.

Future work in this field can explore the incorporation of machine learning models to adaptively adjust thresholds and enhance the robustness of blur detection mechanism. Additionally, expanding the tool's capabilities to detect and quantify motion blur and defocus blur can further improve its utility across diverse imaging applications.

Overall, this research underscores the potential of the Laplacian variance as a simple yet powerful metric for blur detection, paving the way for advancements in automated image quality control and enhancement technologies.

ACKNOWLEDGEMENT

I would like to acknowledge Prisma AI, a software company in Mumbai, Maharashtra, since they provided me with the platform to undertake this project of a blur detection tool and carry out in-depth research on the same

REFERENCES

[1] https://ijcatr.com/archives/volume2/issue4/ijcatr02041019.pdf
[2] https://www.researchgate.net/publication/272709906_Blur_Detection_Methods_for_Digital_Images-A_Survey
[3] https://github.com/topics/blur-detection

[4] https://www.grafiati.com/en/literature-selections/image-blur-detection/

BIOGRAPHIES

I'm Siddhant Ray, a 16-year-old student at Jamnabai Narsee International School. I'm a technogeek who loves to code and constantly seeks new knowledge. My passion for technology drives me to learn and innovate, shaping my journey in the world of programming and beyond.