OpenCV to detect faces and emotions in the webcam video stream

# scaleFactor & minNeighbors explanation :

In the OpenCV's **detectMultiScale** function, **scaleFactor** and **minNeighbors** are two important parameters that affect the object detection process:

* **scaleFactor** parameter is used to compensate for the fact that faces or objects in an image may appear at different sizes due to perspective and distance from the camera. A value of 1.1 means that the algorithm will increase the size of the search window by 10% at each image scale to find the object. A smaller value will increase detection time but will be more accurate, while a larger value will decrease detection time but will be less accurate.
* **minNeighbors** parameter is used to reduce false positives. It specifies how many neighbors each candidate rectangle should have to retain it. Higher values for this parameter mean that the algorithm will be more selective in the rectangles it considers to be objects. A higher value of this parameter will also cause the algorithm to run more slowly.

In summary, **scaleFactor** adjusts the size of the search window and **minNeighbors** adjusts the accuracy of the detection by reducing false positives.

# Version 3

This code uses OpenCV's **`CascadeClassifier`** to detect faces and emotions in the grayscale image. It loops through the detected faces, crops each face region from the frame, and applies the emotion detection to the face region. The processed frame is then displayed in a window.

Note that this code doesn't group the detected faces like the previous version did (Azure 4+), but that functionality could be added by keeping track of the face IDs and grouping them based on proximity or other criteria.

# Version 4

To display the captured frames, I need to add some code to the **while** loop that captures the frames from the webcam and processes them. Im using the **cv2.imshow()** function to display the frames.

This code captures a frame from the webcam video stream, converts it to JPEG format, and then displays it in a new window using the **cv2.imshow()** function. The **cv2.waitKey()** function waits for a key press and returns the ASCII code of the pressed key. In this example, if the 'q' key is pressed, the while loop breaks and the script stops.

# Version 5

**if \_\_name\_\_ == '\_\_main\_\_':** is a conditional statement that is commonly used in Python scripts. It allows you to specify a block of code that should only be executed if the script is being run directly (i.e., as the main program) and not if it is being imported as a module by another script.

Here's how it works:

* When you run a Python script, the interpreter sets a special variable called **\_\_name\_\_** to the value **'\_\_main\_\_'**. This indicates that the script is the main program being run.
* If the script is being imported as a module by another script, the **\_\_name\_\_** variable will be set to the name of the module instead of **'\_\_main\_\_'**.
* The **if \_\_name\_\_ == '\_\_main\_\_':** statement checks whether the **\_\_name\_\_** variable is equal to **'\_\_main\_\_'**. If it is, then the block of code that follows will be executed.

So in this context, **if \_\_name\_\_ == '\_\_main\_\_':** is checking whether the current script is the main program being run (as opposed to being imported as a module), and if it is, it will call the **main()** function.

This is a common practice in Python programming because it allows you to write scripts that can be both run as standalone programs and imported as modules into other scripts.

# Version 6

This code uses OpenCV library to recognize faces and emotions in a live webcam video stream. It loads the pre-trained face and emotion detection models from OpenCV and uses them to detect faces and emotions in each frame of the video stream.

The code captures a frame from the webcam, processes each detected face, detects emotions in the face region and labels them as either "Happy" or "Neutral". It draws rectangles around the detected faces and emotions and displays the processed frame with the emotion label next to the recognized face.

The code continues to run until the user presses the "q" key to quit, after which it releases the resources used by the webcam and closes the window.

The code defines two functions - **main()** and **recognize\_emotion\_and\_face()**. The **main()** function simply calls the **recognize\_emotion\_and\_face()** function to execute the script.

## How does it detect Happiness?

The code uses a pre-trained Haar Cascade classifier to detect smiles in the face region. If a smile is detected in the face region, the emotion is labelled as "Happy".

The Haar Cascade classifier is a machine learning-based approach that uses a set of positive and negative training images to train a classifier for object detection. In this case, the positive training images would be images of smiling faces, and the negative training images would be images of non-smiling faces.

During detection, the classifier analyzes the features of the image in a sliding window manner, looking for matches to the learned patterns of positive and negative examples. If a match is found, it labels that region as a smile, and if the smile region is large enough, it is labeled as a "Happy" emotion.

# Version 9

The code is a Python script that uses OpenCV to recognize emotions and faces in a webcam video stream. It loads the pre-trained face, eye, and smile detection models from OpenCV and initializes variables for smile and crows feet detection.

Then it opens a connection to the default webcam, captures a frame from the video stream, and converts it to JPEG format. The captured frame is then displayed, and the program checks for key presses. It processes each detected face by drawing a rectangle around it, cropping the face region from the frame, and detecting eyes and smiles in the face region.

It checks if both eyes and smile are detected, draws rectangles around the eyes and smile, and searches for crow's feet or wrinkles around the outer corners of the eyes. It calculates the confidence percentage based on the number of detected eyes and smiles and sets the smile and crow's feet detection variables to True if they are detected.

It determines the emotion label based on the detected face region and the confidence percentage and adds the confidence percentage to the text to display next to the recognized face. Finally, it releases the webcam and closes all windows.

Overall, the code is a facial recognition program that detects emotions and faces in real-time using a webcam video stream. It is written in a modular way and follows best practices such as error handling and code comments.

# Version 10

In this code, **recognize\_emotion\_and\_face()** function captures a frame from the webcam video stream and performs face detection and emotion recognition on each detected face. It uses OpenCV's pre-trained face, eye and smile detection models to detect faces and the presence of eyes and smiles in each face region.

The function starts by opening a connection to the default webcam using the **cv2.VideoCapture()** method. It then captures a frame from the webcam using the **cap.read()** method and converts it to JPEG format using the **cv2.imencode()** method. The captured frame is displayed on the screen using the **cv2.imshow()** method.

Next, the function performs face detection on the captured frame using the **face\_cascade.detectMultiScale()** method. It uses a pre-trained face detection model from OpenCV's **haarcascades** library to detect faces in the frame. The **detectMultiScale()** method takes in several parameters, including **scaleFactor**, **minNeighbors**, and **flags**, which control the sensitivity and accuracy of the face detection algorithm. The **scaleFactor** parameter controls the amount by which the image is resized at each image scale. The **minNeighbors** parameter specifies how many neighbors a candidate rectangle should have to retain it.

For each detected face, the function draws a rectangle around the face region using the **cv2.rectangle()** method. It then extracts the grayscale and color regions of interest (ROIs) corresponding to the detected face using the **cv2.cvtColor()** and array slicing operations.

Next, the function performs eye and smile detection on the face region using the pre-trained eye and smile detection models. It uses the **eye\_cascade.detectMultiScale()** and **smile\_cascade.detectMultiScale()** methods to detect eyes and smiles in the face region.

For each detected eye and smile, the function draws a rectangle around the eye and smile regions using the **cv2.rectangle()** method. It also performs additional processing on the smile region to check if a smile is present and if it is a big smile. This is done by calling the **detect\_smile()** function, which takes in the grayscale and color smile ROIs as arguments and returns a tuple containing the boolean values indicating if a smile and big smile were detected in the ROI.

Finally, the function checks if both eyes and a smile are present in the face region and if so, sets the **crows\_feet\_detected** variable to True if crow's feet or wrinkles are present around the outer corners of the eyes.

# Version 12

## Overall

This code is implementing facial detection using the OpenCV library. It loads pre-trained models for detecting faces, eyes, and smiles, and then defines functions for detecting smiles and sadness in facial regions of interest.

The code imports the necessary libraries: OpenCV, NumPy, Matplotlib, Pickle, and Pandas. The face, eye, and smile detection models are loaded from the OpenCV library using Haar feature-based cascade classifiers.

The code then initializes variables for smile and crows feet detection and defines the minimum size of the smile to be considered a "big" or "normal" smile. An empty list is created to store the data, and a variable for the image name is initialized.

The function **show\_graph** is defined to display a graph with the given label in a new window. It appends the data to the **data\_list** and plots the data using Matplotlib.

The **detect\_smile** function takes the grayscale and color images of a region of interest and detects the presence of a smile in the region of interest. It uses the smile detection model to detect smiles in the grayscale image, then checks if the detected smile is a "big" or "normal" smile based on the minimum width and height values. If a smile is detected, the function returns True for **smile\_detected** and **big\_smile\_detected**, otherwise, it returns False for both.

The **detect\_sadness** function takes the grayscale and color images of a region of interest and detects the presence of sadness in the region of interest. It detects the face region using the face detection model, then detects the eyebrows and mouth in the face region using the eye and smile detection models. It checks if the distance between the eyebrows and the top of the eyes is above a threshold value and the mouth is in a "down" orientation. If sadness is detected, the function returns True for **sadness\_detected**, otherwise, it returns False.

## Graph Code

This is a Python function named **show\_graph** that creates a graph using the Matplotlib library to display the given data in a new window.

The function takes one argument, **label**, which is the label that will be displayed on the x-axis of the graph.

First, there are two commented out lines of code: **plt.xlabel(label)** and **plt.show()**. These two lines would set the x-axis label to the **label** argument and then display the graph window. However, they are not being used in this version of the function.

Next, the **data** variable is set to the **label** argument. This line assumes that the **label** argument contains the actual data that needs to be plotted.

The **data\_list** variable is assumed to be a list that already exists outside of this function and that it stores all the data that has been plotted in previous calls to the **show\_graph** function. The **data** variable is then appended to this list using the **append()** method.

The **print()** statement is used to display the **data** argument, which is assumed to be the data that is being plotted. This is just for debugging purposes and can be removed in the final version of the code.

Finally, the **plt.plot()** function is used to plot the data in **data\_list**, which includes the new data that was just appended to the list. The **plt.draw()** function updates the graph with the new data, and the **plt.pause(0.001)** function adds a small delay to allow the graph to update before the function exits.

## Smile Code

This code defines a function **detect\_smile** that takes in two arguments, **smile\_roi\_gray** and **smile\_roi\_color**. The function's purpose is to detect the presence of a smile in a region of interest (ROI) in an image.

The function first uses the **detectMultiScale** function from OpenCV to detect any smiles in the grayscale image **smile\_roi\_gray**. The detected smiles are stored in a variable called **smiles**.

The function then checks if a smile is detected and if it's a big smile. It iterates over the detected smiles and draws a rectangle around each smile in the color image **smile\_roi\_color**. If the width and height of the rectangle are greater than the minimum width and height for a big smile, the function sets **big\_smile\_detected** to **True**. If the width and height of the rectangle are greater than the minimum width and height for a small smile, the function sets **smile\_detected** to **True**.

The function returns a tuple of two Boolean values, **smile\_detected** and **big\_smile\_detected**, indicating whether a smile was detected and whether it was a big smile or not.

## Sadness Code

This code is a function that detects the presence of sadness in a region of interest, which is a facial region specified by the input **face\_roi\_gray** and **face\_roi\_color**. It first sets the threshold distance between the eyebrows and the eyes and the minimum distance between the mouth corners and the bottom edge of the face to be considered an "up" and "down" orientation, respectively.

Then, it detects the face region using the **face\_cascade** object, which is a pre-trained classifier for detecting faces. Inside the loop for detected faces, it extracts the region of interest for eyes and mouth using the **eye\_cascade** and **smile\_cascade** objects respectively, which are pre-trained classifiers for detecting eyes and smiles.

Next, it finds the topmost point of the eyes and checks if the distance between the eyebrows and the top of the eyes is above the threshold and the mouth is in a "down" orientation. If both conditions are met, it sets **sadness\_detected** as **True**. It also sets **eyebrow\_detected** as **True** if the distance between the eyebrows and the top of the eyes is above the threshold.

Finally, it returns two boolean values **sadness\_detected** and **eyebrow\_detected** indicating whether sadness and eyebrows are detected in the region of interest.

## Conclusion

The program is attempting to detect facial expressions in images using OpenCV. It uses pre-trained models for detecting faces, eyes, and smiles, and then applies various criteria to determine if a smile or frown is present. It also has functions for displaying a graph of the detected expressions and for saving the data to a file.

The code itself is well-organized, with comments explaining the various functions and variables.