# FaceAPI V2

This code checks if the response from the API contain any face detection, if yes it will parse the response and extract the emotions, otherwise it will give a proper feedback.

Please make sure to use the correct endpoint and subscription\_key for your Azure Face API instance, and use the appropriate image path when testing the function.

Also, as a reminder, this code should be thoroughly tested and audited for security and performance before deployment.

# FaceAPI V3

This code uses the "returnFaceAttributes" and "returnFaceId" parameters to return additional information about the faces detected in the image, such as facial landmarks, head pose, gender, age, etc.

Please make sure to use the correct endpoint and subscription\_key for your Azure Face API instance, and use the appropriate image path when testing the function. As the response will contain a lot of information, you can extract the desired information by parsing the response\_json accordingly.

Also, as a reminder, this code should be thoroughly tested and audited for security and performance before deployment.

In addition, please note that the Azure Face API also supports face recognition, which allows you to compare a face in an image with a set of known faces, and determine whether the face in the image is a match. To perform face recognition, you would need to create a face group, and add the known faces to the group. Then you can use the "identify" API method to identify a face in an image by comparing it to the faces in the group.

# FaceAPI V4

# FaceAPI V5

This code will extract the faceIds from the response of the face detection API and use them to call the group API endpoint. Then it parses the response to extract the groups of similar faces, where each group contains an array of face IDs that belong to the same person.

Please note that that what I have coded is just an example, it is very broad and not actually specific to my end goal and the actual implementation might require some adjustments based on my specific use case, but overall, it’s a step towards my finishing line. If someone else wished to us this they must, make sure to replace the headers and endpoint with your their own subscription key and endpoint, and test it with the correct image path that they themselves desire.

This code defines a function recognize\_emotion\_and\_face(image\_path) that recognizes emotions and faces in an image using the Azure Face API.

The function first sets the subscription\_key and endpoint for the Azure Face API, and then defines headers for the API request, including the subscription key.

The function then takes the path of the image to be processed as an argument, reads the binary data from the image file, and defines the parameters for the API request.

A POST request is then sent to the Azure Face API endpoint to detect faces in the image. The API request includes the image data, headers, and parameters. If the request is successful, the function parses the JSON response to obtain the detected face IDs.

The function then sends another POST request to group the detected faces, using the group endpoint of the Azure Face API. If the request is successful, the function parses the JSON response to obtain the groups of faces detected in the image, and returns the groups. If there are no faces detected in the image, the function returns None and prints an error message.

The code also includes error handling for common issues that may occur while processing the image and calling the Azure Face API. The function returns None if an error occurs, and prints an error message indicating the type of error that occurred.

# Face API V6

This code defines a Python function recognize\_emotion\_and\_face() that captures a video stream from the default webcam and uses the Azure Face API to detect faces and recognize emotions from each frame of the video.

First, the code initializes the Azure Face API subscription key, endpoint, and request headers. It then defines the parameters for the API request, including the facial attributes to be returned and the option to return face IDs.

Next, the code enters an infinite loop where it captures a frame from the webcam video stream, converts it to JPEG format, and sends a POST request to the Azure Face API endpoint with the image data and request parameters. If the response contains detected faces, the code sends a second API request to group the faces and return information about each group.

Finally, if faces are detected, the code returns a dictionary of information about the face groups, including the group ID, the number of faces in each group, and the emotion and other facial attributes of each face in the group. If no faces are detected, the code prints a message indicating that no faces were found in the image.

Note that the code also includes error handling to catch and report any errors that occur during the API requests or response parsing.

## does the code continue to try to detect faces after it finds one and sends it off to azure?

Yes, the code continues to detect faces after it finds one and sends it off to Azure. The while loop in the recognize\_emotion\_and\_face function captures frames from the webcam video stream continuously until the user stops the program. Each frame is then processed and sent to the Azure Face API for face detection and recognition. If a face is detected in a frame, the code groups the detected face(s) using the Face API's group method, which groups faces based on similarity. Finally, the code returns the group information to the caller function. If no face is detected in a frame, the code prints a message stating that no face was detected and continues to capture and process the next frame.

# Face API V7

This is a Python script that uses the Azure Face API to recognize emotions and faces in a webcam video stream. Here is a step-by-step breakdown of what the code does:

1. The script imports the necessary libraries: requests, json, and OpenCV (cv2).
2. The script sets the Azure Face API subscription key and endpoint variables to be used in the API requests.
3. The script defines the headers for the API request, including the subscription key.
4. The function recognize\_emotion\_and\_face() is defined. This function opens a connection to the default webcam and sets parameters for the API request.
5. The function enters a while loop to continuously capture frames from the webcam video stream.
6. Inside the while loop, the script captures a frame from the webcam and converts it to JPEG format.
7. The script then sends a POST request to the Azure Face API endpoint to detect faces in the captured image.
8. The script parses the JSON response from the API to check if any faces were detected in the image.
9. If one or more faces are detected, the script extracts the face IDs from the response, groups them, and returns the resulting groups.
10. If no faces are detected, the script prints a message to indicate this.
11. Once a face has been detected and processed, the script releases the resources used by the webcam and closes the window.

In summary, this script uses the Azure Face API to detect faces and recognize emotions in a webcam video stream. It continuously captures frames from the webcam, sends them to the API for processing, and returns information about the detected faces.

# Face API V8

At this point in the code (line 80), we are handling the case where an error occurs while parsing the response JSON returned by the Face API after detecting faces in the image.

The code tries to parse the response JSON using the json.loads() method, which can raise a json.decoder.JSONDecodeError exception if the response is not valid JSON.

If such an exception is raised, the code prints an error message indicating that an error occurred while parsing the response, along with the error message returned by the exception. This message helps to provide additional information about the cause of the error.

Finally, the function returns None to indicate that an error occurred, so that the caller can handle the error appropriately.

This block of code tries to parse the JSON response returned by the Face API. If there is an error while parsing the JSON, a JSONDecodeError is raised, and the error message is printed to the console. (line 80 – 86)

This block of code (line 89 – 117) checks whether any faces were detected in the image. If at least one face is detected, the code extracts the face IDs from the JSON response and calls the Face API's "group" API to group the faces by similarity. If there is an error while calling the "group" API or parsing the JSON response, an error message is printed to the console. If the group call is successful, the code extracts the groups from the JSON response and returns them. If no faces were detected in the image, a message is printed to the console. Finally, the while loop is broken and the OpenCV video capture and display resources are released.

Note that this code assumes that the endpoint and headers variables have been defined earlier in the script. These variables contain the endpoint and headers required to make requests to the Face API. Also, the code assumes that the response variable contains the JSON response returned by the Face API's "detect" API, and that the cap variable contains an instance of OpenCV's VideoCapture class, which is used to capture images from a webcam.

These two lines of code are releasing the resources acquired by the OpenCV VideoCapture object and closing all OpenCV windows.

cap.release() releases the resources acquired by the VideoCapture object. This includes the webcam if it was used, freeing it up for use by other applications.

cv2.destroyAllWindows() is used to close any OpenCV windows that might have been opened during the program execution. It takes no arguments and simply closes any windows that were opened during the program's execution.