

The above piece of code imports various libraries that includes:

**Numpy:** A popular numerical computing library in Python, often used for array operations and mathematical computations.

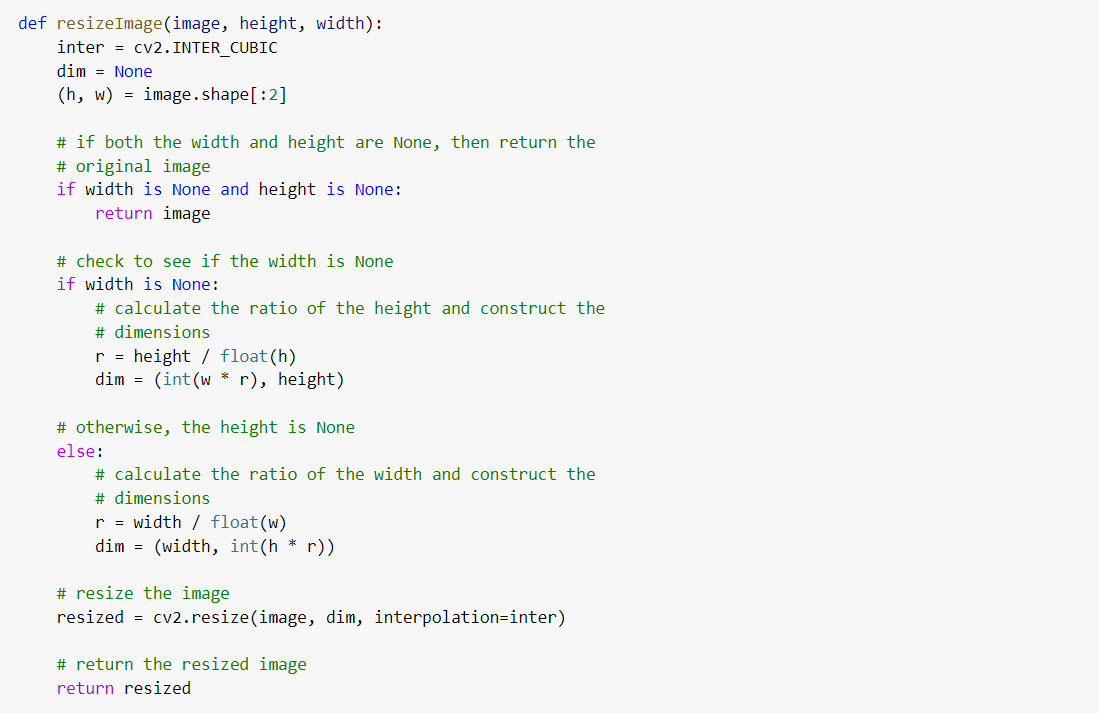
**OS:** A way of using operating system dependent functions like reading or writing to the file system.

**Cv2:** An open source computer vision library used for image and video processing related functions.

**Matplotlib**: A plotting library in Python, used for plotting graphs and visualizations.

**TensorFlow:** One of the most famous open-source libraries for Machine Learning related applications. We have used this library for Neural Network implementation in our case.

**Scikit-Learn:** We have imported the train\_test\_split function from this library, to split our data into training and test sets.



In the above piece of code, we have defined a Python function called **resizeImage**. We have used the openCV library to **resize an input** to a new Height and/or Width. We are taking in three parameters, the image to be resized, the desired image height and width. The inter values store the result of interpolation, using **INTER\_CUBIC**, the **dim** variable stores the dimensions. The initial pieces of code deals with None values of height and width parameters. The resized image is stored in the **resize variable** and returned.

**def readImages(folderPath) :**

**images = []**

**for filename in os.listdir(folderPath) :**

**img = cv2.imread(os.path.join(folderPath, filename))**

**if img is not None :**

**images.append(img)**

**raw\_images = np.array(images)**

**images = np.array(images)**

**images = images.reshape(images.shape[0], -1)**

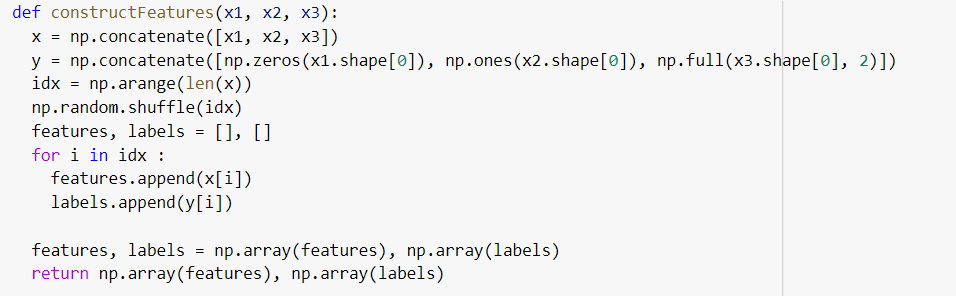
**images = images.astype('float64')**

**images /= 255.0**

**return raw\_images, images**

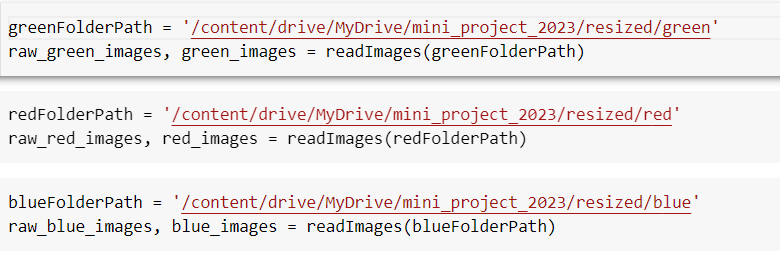
In the above piece of code, we have defined a **readImages function** which takes in the folder path as the argument. The function will **read the file** while iterating in the folder path using **cv2.imread method** of the OpenCV library and if the value of the image is not None, it will append the image to our **img variable.**

The image is converted into a **Numpy array** and stored in the **raw\_images** variable and the **images** variable. raw\_images now stores the images before reshaping and normalization is applied to it while images will be further worked on. The images is then reshaped into a 2 dimensional array. After resizing, it is converted to **float type**, so as to normalize it by a **factor of 255** because by industry standard RGB values are encoded as an 8-bit integer, which ranges from **0 to 255**.

****

Now, we have defined a constructFeatures function, which takes in 3 arguments, ‘**x1**’, ‘**x2**’ and ‘**x3**’. The function first **concatenates** the three input arrays along the first axis using Numpy’s **np.concatenate()**, creating a new array **x** that has all the rows of x1, x2, and x3. Next, the function creates a new array **y** that has the same number of rows as **x**, with the first x1.shape[0] rows set to 0, the next x2.shape[0] rows set to 1, and the last x3.shape[0] rows set to 2. In this way, we have created a label array that has a **unique label for each row** of x.

The function then creates an array of indices ‘**idx’** that has the **same length as x** and contains the integers from **0 to len(x) - 1**. It shuffles the indices randomly using np.random.shuffle() to create the training data and making sure to improve generalisation and reduce the risk of overfitting. The function then creates two empty lists, **features and labels**. It loops through the shuffled indices and **appends** the feature and label values of the corresponding row in **x** and **y** to these lists, respectively.



The code reads all the images located in the **green directory** of a specified path in Google Drive. The function **readImages() is called** to read the images and returns two lists of NumPy arrays as specified in the function description above. Similarly, we have done the same for red and blue training images.



We have called the constructFeatures() function on our respective red, blue and green arrays.The implementation of constructFeatures() method is explained above.

Sure, the above code is a Python function that uses the OpenCV library to resize an input image to a new height and/or width. The function takes in three arguments: the image to be resized, and the desired height and width of the output image. It first sets an interpolation method to be used by the cv2.resize() function, which in this case is cv2.INTER\_CUBIC to produce high-quality results.

Next, the function calculates the dimensions of the output image based on the desired height and/or width, while preserving the aspect ratio of the input image. If either the height or width is not specified, the function calculates the missing dimension based on the aspect ratio of the input image. The function then resizes the input image to the new dimensions using the cv2.resize() function and the specified interpolation method.

Finally, the function returns the resized image. This function is a useful utility for resizing images to a specific size while preserving the aspect ratio of the original image, which is important to avoid distorting the image's content.