Feature extraction and image classification using OpenCV

Last Updated : 23 Jul, 2025

This article is your ultimate guide to becoming a pro at image feature extraction and classification using OpenCV and Python. We'll kick things off with an overview of how OpenCV plays a role in feature extraction, and we'll go through the setup process for the OpenCV environment. You'll get to learn all about different feature extraction techniques and algorithms, with a focus on the ones specific to OpenCV.

Table of Content

What is Feature Extraction in Computer Vision?

Feature extraction is a crucial process in machine learning and data analysis. It focuses on identifying and extracting relevant features from raw data, transforming it into numerical features that machine learning algorithms can work with. This process helps convert raw data into a format suitable for machine learning models and plays a role in dimensionality reduction, simplifying large datasets into manageable groups. The goal is to reduce data complexity while retaining as much relevant information as possible.

Step-by-Step Guide to Feature Extraction using OpenCV

Step 1: Install OpenCV

Ensure you have OpenCV installed. You can install it using pip if you haven't already:

```
pip install opencv-python
```

Step 2: Import Libraries

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Step 3: Load an Image

Input Image

```
image = cv2.imread('GeeksForGeeks.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

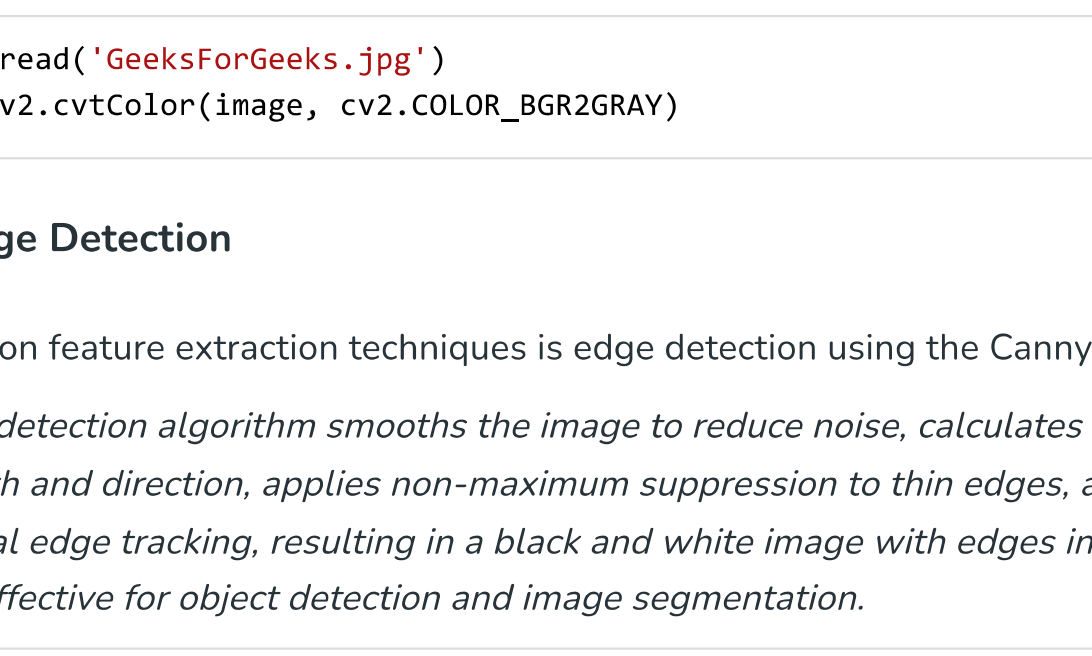
Step 4: Use Edge Detection

One of the common feature extraction techniques is edge detection using the Canny algorithm.

The Canny edge detection algorithm smooths the image to reduce noise, calculates the gradient to find edge strength and direction, applies non-maximum suppression to thin edges, and uses hysteresis for final edge tracking, resulting in a black and white image with edges in white. This makes it highly effective for object detection and image segmentation.

```
edges = cv2.Canny(gray_image, 100, 200)
plt.imshow(edges, cmap='gray')
plt.title('Edge Image')
plt.show()
```

Output:



output

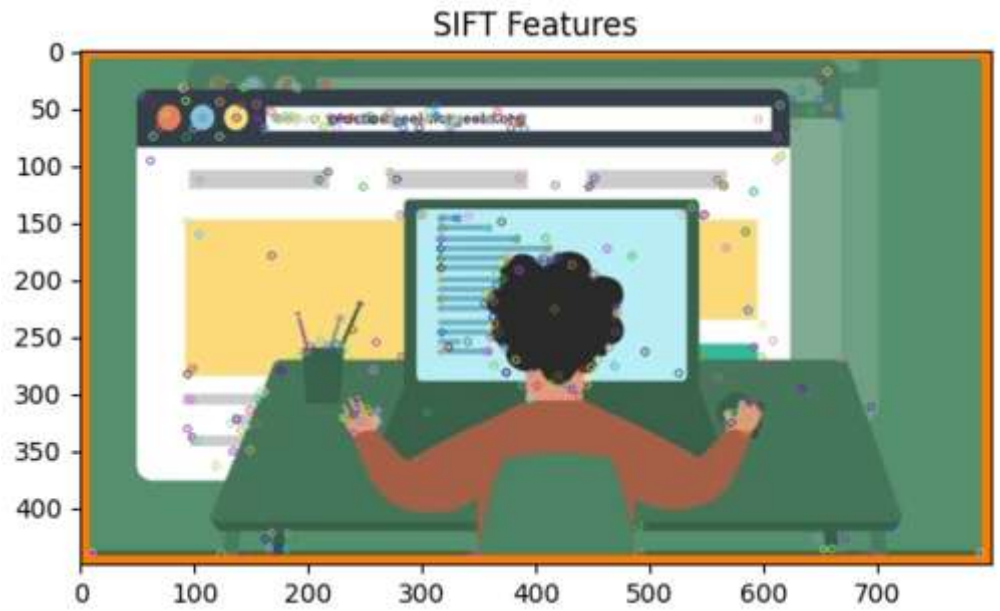
Step 5: Use Feature Detectors

OpenCV provides several feature detectors. Two commonly used ones are SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF).

First, we are going to implement SIFT, SIFT can find specific points in an image by looking at their size, orientation, and local details.

```
sift = cv2.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(gray_image, None)
image_with_sift = cv2.drawKeypoints(image, keypoints, None)
plt.imshow(cv2.cvtColor(image_with_sift, cv2.COLOR_BGR2RGB))
plt.title('SIFT Features')
plt.show()
```

Output:

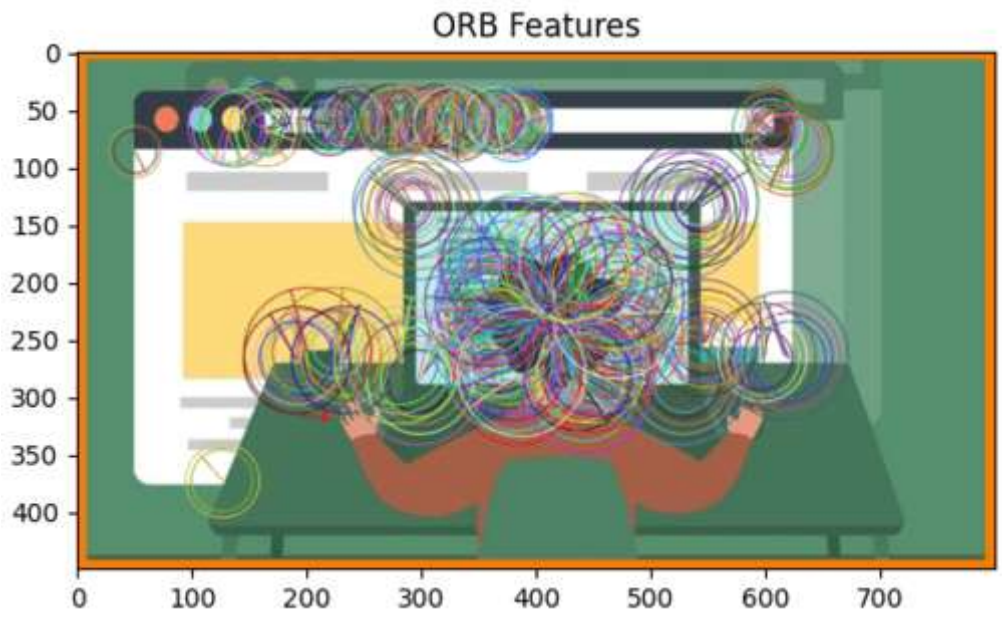


SIFT Output

Now, we will be implementing ORB and ORB is a faster alternative to SIFT, suitable for real-time applications.

```
orb = cv2.ORB_create()
keypoints, descriptors = orb.detectAndCompute(gray_image, None)
image_with_orb = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(cv2.cvtColor(image_with_orb, cv2.COLOR_BGR2RGB))
plt.title('ORB Features')
plt.show()
```

Output:



ORB Output

In a nutshell, OpenCV offers feature extraction methods like SIFT and ORB that are crucial for identifying important features in images. SIFT is super robust, but it can be a bit of a computer hog. On the other hand, ORB is a faster option that performs just as well. By using these methods, you can easily extract features from images and unlock a whole world of possibilities in image processing and computer vision.

Implementing Image Classification with OpenCV

Step 1: Install the necessary libraries

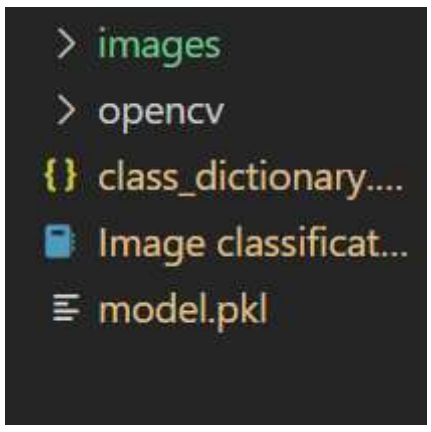
```
pip install PyWavelets
pip install opencv-python
```

Step 2: Importing Necessary Libraries

Importing necessary libraries like NumPy for numerical operations, OpenCV for image processing, and Matplotlib for plotting images.

```
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

We have used the Indian Cricketer Dataset and the directory structure is discussed below.

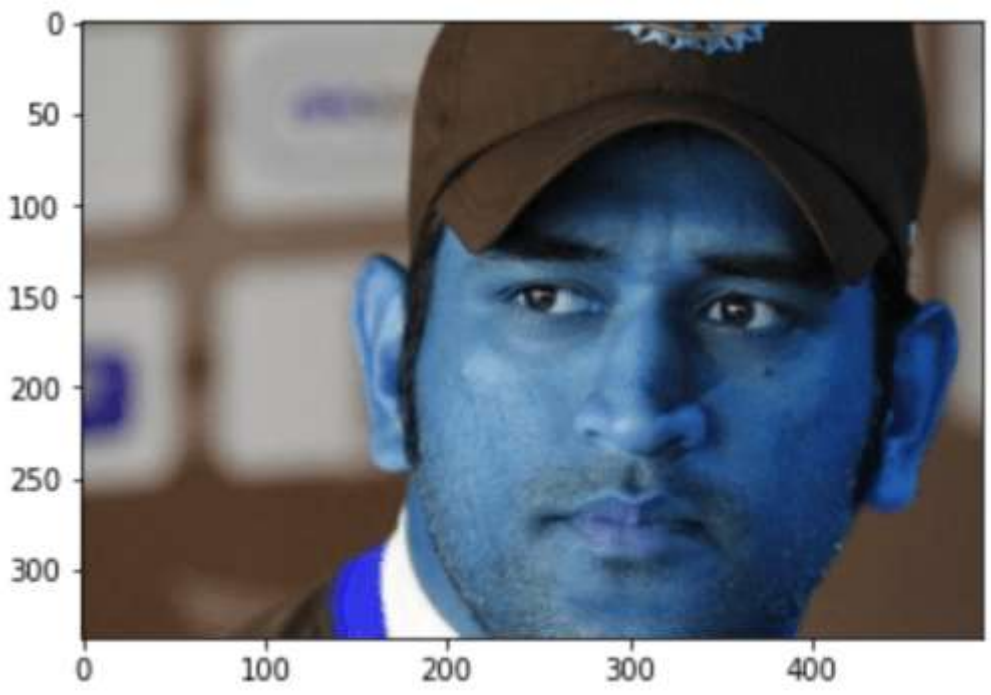


Directory Structure

Step 2: Loading and Displaying the Image

```
img = cv2.imread('images\ms_dhoni\dhoni.jpg')
plt.imshow(img)
```

Output:



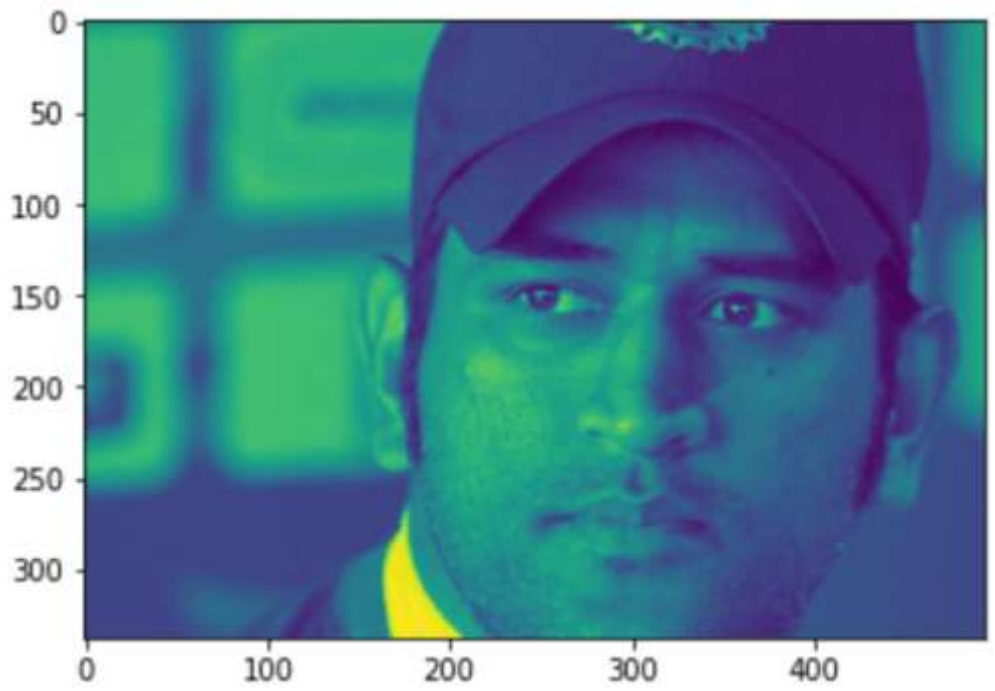
output

Step 4: Converting Image to Grayscale

Converting the loaded image to grayscale using cv2.cvtColor and displaying the grayscale image. The shape of the grayscale image is also printed.

```
#Converting the image to gray
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray.shape
plt.imshow(gray)
```

Output:



Output

```
plt.imshow(gray, cmap='gray')
```

Output:

Output

Step 5: Loading Haar Cascades for Face and Eye Detection

The code initializes face and eye classifiers using pre-trained Haar cascade models and applies the face classifier to a grayscale image to detect faces. The results are stored in the faces variable, which contains a list of rectangles representing detected faces. This setup enables the identification of facial regions within the image, allowing for further processing or feature extraction.

```
face_cascade =
cv2.CascadeClassifier('./opencv/haarcascades/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('./opencv/haarcascades/haarcascade_eye.xml')

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
faces
```

Output:

```
array([[172,  55, 268, 268]])
```

The code extracts the coordinates and dimensions of the first detected face from the faces variable. Specifically, x and y represent the top-left corner coordinates of the face, while w and h represent the width and height of the bounding box around the face. This step isolates the region of interest for further processing, such as drawing rectangles around the detected face or performing additional analyses

```
(x,y,w,h) = faces[0]
```

```
x,y,w,h
```

Output:

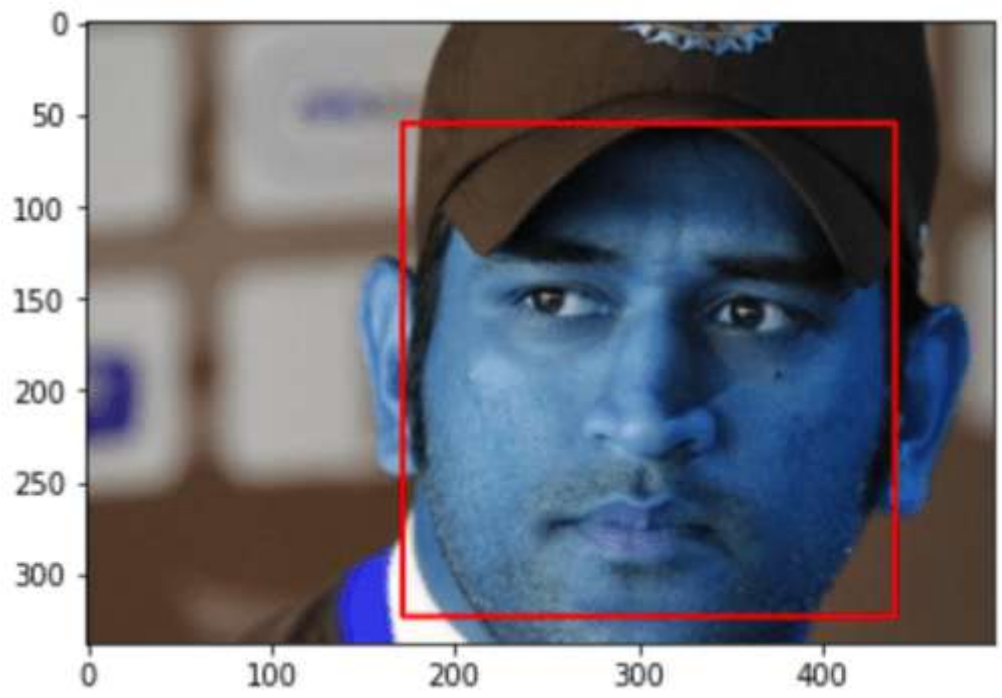
```
(172, 55, 268, 268)
```

Step 6: Detecting Faces and Drawing Rectangles Around Them

Detecting faces in the grayscale image using detectMultiScale. The first detected face's coordinates are used to draw a rectangle around it on the original image. The resulting image is displayed.

```
face_img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(face_img)
```

Output:



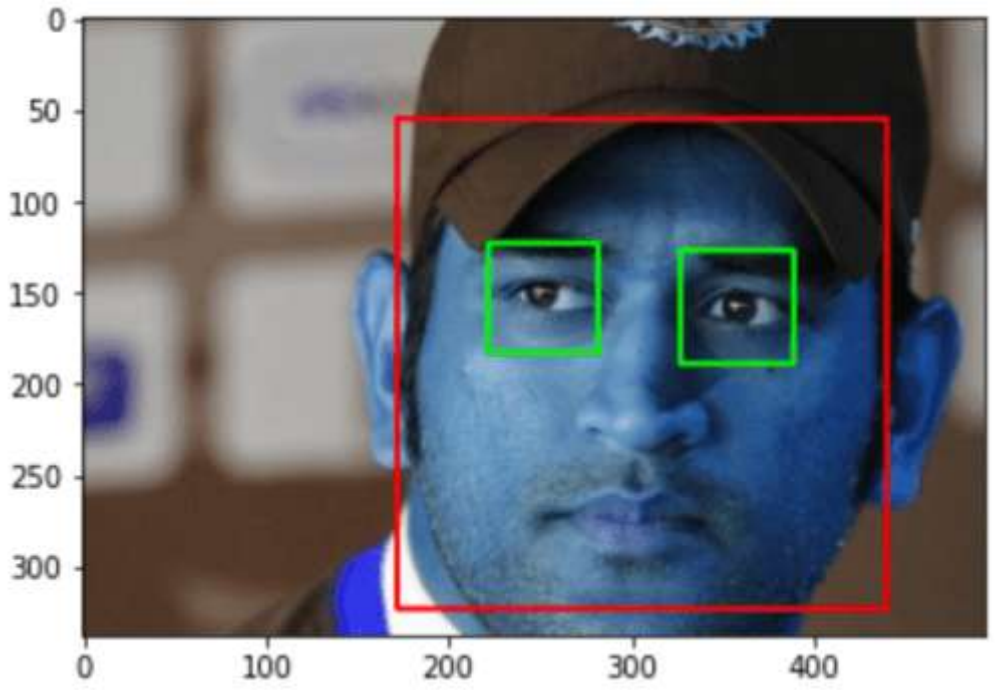
Output

Step 7: Detecting Faces , eyes and Drawing Rectangles Around Them

The code detects faces and eyes in an image and visualizes the results. It first closes any open OpenCV windows, then loops through the detected faces to draw red rectangles around them. For each detected face, it defines regions of interest (ROI) in both grayscale and color images. Within each face ROI, it detects eyes and draws green rectangles around them. Finally, it displays the image with the detected faces and eyes, showing the rectangles in grayscale. This provides a clear visual representation of the detected features..

```
cv2.destroyAllWindows()
for (x,y,w,h) in faces:
    face_img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = face_img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)


plt.figure()
plt.imshow(face_img, cmap='gray')
plt.show()
```

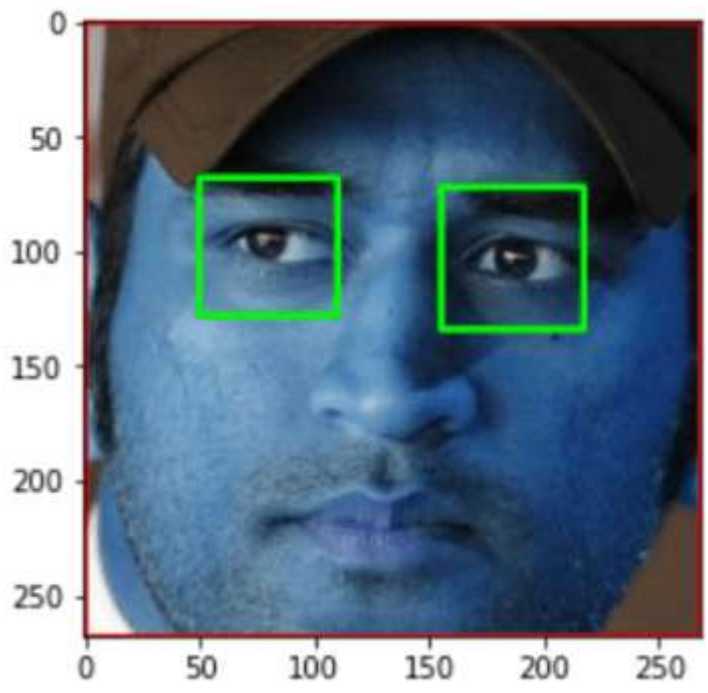
Output:



Output

```
#Plotting the cropped Image
%matplotlib inline
plt.imshow(roi_color, cmap='gray')
plt.show()
```

Output:



output

```
cropped_img = np.array(roi_color)
cropped_img.shape
```

Output:

```
(268, 268, 3)
```

Step 8: Performing a 2D wavelet transform on an image.

The purpose of this code is to apply a 2D wavelet transform to an input image, which can be useful in various image processing tasks, including denoising, compression, and feature extraction. The specific wavelet used and the level of decomposition can be adjusted according to the requirements of the task.

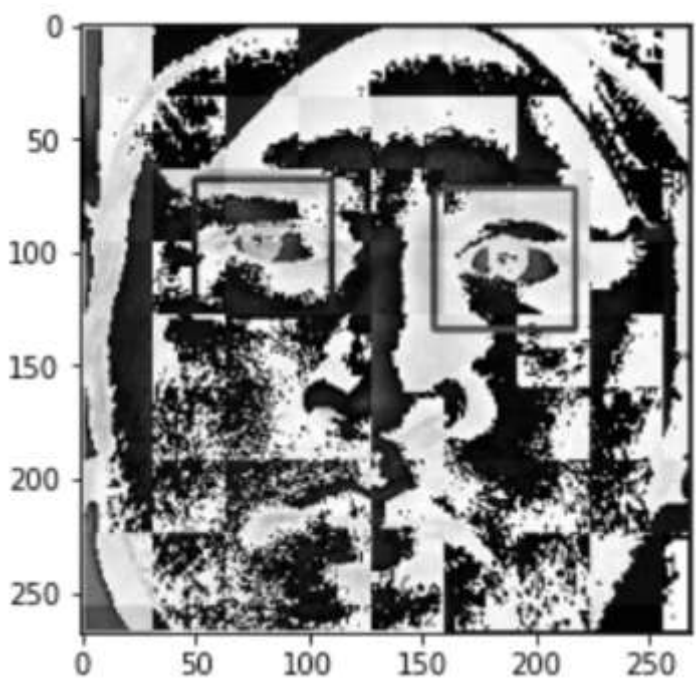
```
import numpy as np
import pywt
import cv2

def w2d(img, mode='haar', level=1):
    imArray = img
    #Datatype conversions
    #convert to grayscale
    imArray = cv2.cvtColor( imArray,cv2.COLOR_RGB2GRAY )
    #convert to float
```

```
    imArray =  np.float32(imArray)
    imArray /= 255;
    # compute coefficients
    coeffs=pywt.wavedec2(imArray, mode, level=level)

    #Process Coefficients
    coeffs_H=list(coeffs)
    coeffs_H[0] *= 0;

    # reconstruction
    imArray_H=pywt.waverec2(coeffs_H, mode);
    imArray_H *= 255;
    imArray_H =  np.uint8(imArray_H)

    return imArray_H
```

Step 9: Function to Get Cropped Image with Two Eyes

```
im_har = w2d(cropped_img,'db1',5)
plt.imshow(im_har, cmap='gray')
```

Output:



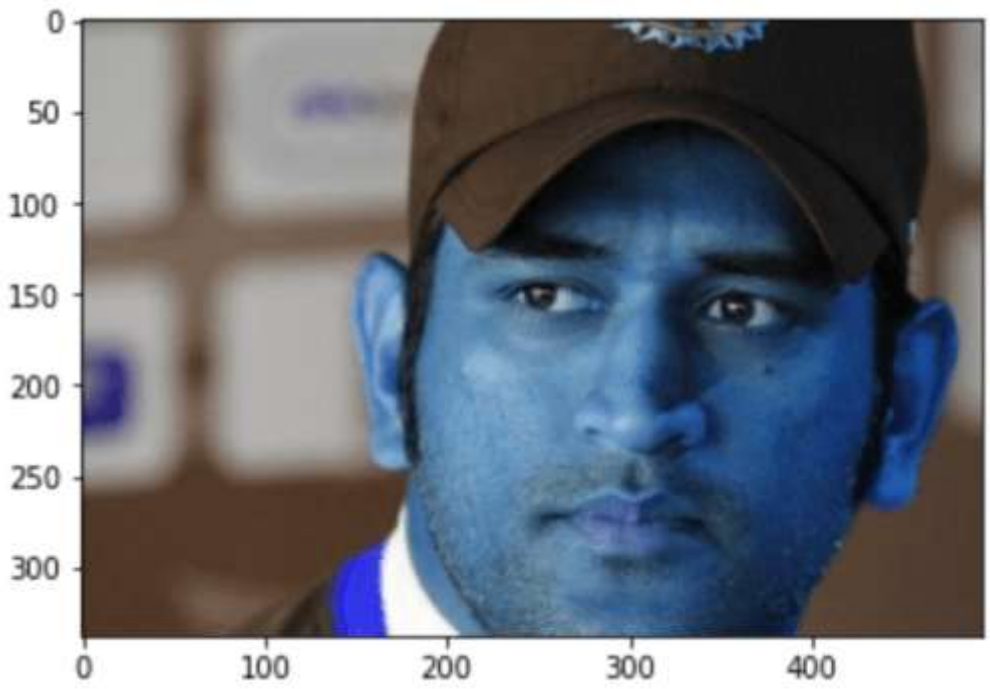
Output

Step 10: Preprocessing: Load image, detect face. If eyes >=2, then save and crop the face region

Lets write a python function that can take input image and returns cropped image (if face and eyes >=2 are detected)

```
def get_cropped_image_if_2_eyes(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        if len(eyes) >= 2:
            return roi_color

original_image = cv2.imread('images\ms_dhoni\dhoni.jpg')
plt.imshow(original_image)
```

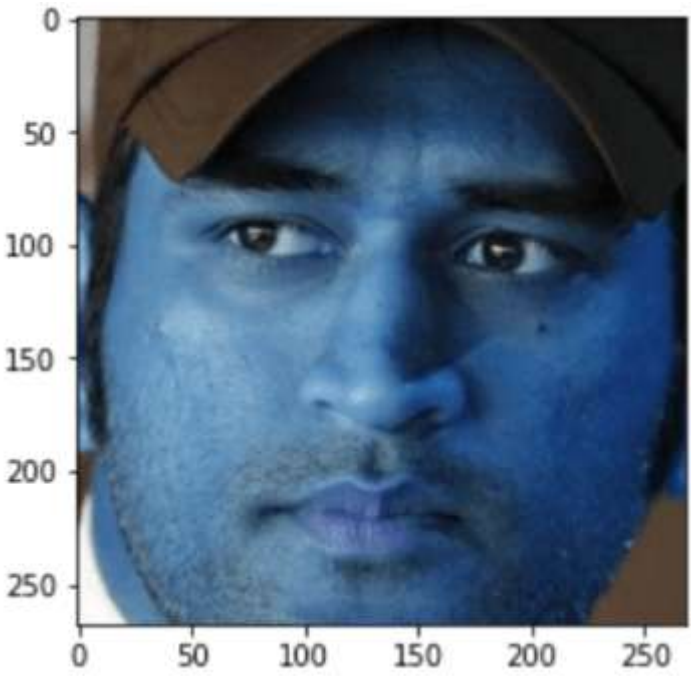
Output:



Output

```
cropped_image = get_cropped_image_if_2_eyes('images\ms_dhoni\dhoni.jpg')
plt.imshow(cropped_image)
```

Output:



Output

Step 9: Preparing and Organizing the Dataset

```
path_to_data = "./images/"
path_to_cr_data = "./images/cropped/"

import os
img_dirs = []
for entry in os.scandir(path_to_data):
    if entry.is_dir():
        img_dirs.append(entry.path)
img_dirs
```

Output:

```
['./images/bhuvneshwar_kumar',
 './images/dinesh_karthik',
 './images/hardik_pandya',
 './images/jasprit_bumrah',
 './images/k._l._rahul',
 './images/kedar_jadhav',
 './images/kuldeep_yadav',
 './images/mohammed_shami',
 './images/ms_dhoni',
 './images/ravindra_jadeja',
 './images/rohit_sharma',
```

```
    './images/shikhar_dhawan',
    './images/vijay_shankar',
    './images/virat_kohli',
    './images/yuzvendra_chahal']
```

Go through all images in dataset folder and create cropped images for them. There will be cropped folder inside dataset folder after you run this code.

```
import shutil
if os.path.exists(path_to_cr_data):
     shutil.rmtree(path_to_cr_data)
os.mkdir(path_to_cr_data)
```

1. **Initialization:** Begin by initializing an empty list *cropped_image_dirs* to store paths to directories containing cropped images, and an empty dictionary *celebrity_file_names_dict* to store file paths for each celebrity.
2. **Iterate through Directories:** Iterate through each directory (img_dirs) containing images of different celebrities.
3. **Detect and Crop Faces:** For each image in the directory, attempt to detect and crop the face if both eyes are visible using the get_cropped_image_if_2_eyes function.
4. **Create and Save Cropped Images:**
    1. If a valid face is detected, create a folder (if it doesn't already exist) to store cropped images for that celebrity.
    2. Save the cropped image to the appropriate folder with a unique file name.
    3. Update the *celebrity_file_names_dict* with the file path of the cropped image.
    4. Use a count variable to ensure each saved image has a unique name within its respective celebrity's folder.
5. **Repeat for Each Celebrity:** Repeat steps 2-4 for each directory containing images of different celebrities.

```
cropped_image_dirs = []
celebrity_file_names_dict = {}

for img_dir in img_dirs:
    count = 1
    celebrity_name = img_dir.split('/')[-1]
    print(celebrity_name)

    celebrity_file_names_dict[celebrity_name] = []

    for entry in os.scandir(img_dir):
        roi_color = get_cropped_image_if_2_eyes(entry.path)
        if roi_color is not None:
            cropped_folder = path_to_cr_data + celebrity_name
            if not os.path.exists(cropped_folder):
                os.makedirs(cropped_folder)
                cropped_image_dirs.append(cropped_folder)
                print("Generating cropped images in folder: ",cropped_folder)

            cropped_file_name = celebrity_name + str(count) + ".png"
            cropped_file_path = cropped_folder + "/" + cropped_file_name

            cv2.imwrite(cropped_file_path, roi_color)
            celebrity_file_names_dict[celebrity_name].append(cropped_file_path)
            count += 1
```

Output:

```
bhuvneshwar_kumar
Generating cropped images in folder:  ./images/cropped/bhuvneshwar_kumar
```

```
dinesh_karthik
Generating cropped images in folder:  ./images/cropped/dinesh_karthik
hardik_pandya
Generating cropped images in folder:  ./images/cropped/hardik_pandya
jasprit_bumrah
Generating cropped images in folder:  ./images/cropped/jasprit_bumrah
k._l._rahul
Generating cropped images in folder:  ./images/cropped/k._l._rahul
kedar_jadhav
Generating cropped images in folder:  ./images/cropped/kedar_jadhav
kuldeep_yadav
Generating cropped images in folder:  ./images/cropped/kuldeep_yadav
mohammed_shami
.....
yuzvendra_chahal
Generating cropped images in folder:  ./images/cropped/yuzvendra_chahal
```

Now you should have cropped folder under datasets folder that contains cropped images ,Manually examine cropped folder and delete any unwanted images

Step 10: Creating a Dictionary for Training Files

```
celebrity_file_names_dict = {}
for img_dir in cropped_image_dirs:
    celebrity_name = img_dir.split('/')[-1]
    file_list = []
    for entry in os.scandir(img_dir):
        file_list.append(entry.path)
    celebrity_file_names_dict[celebrity_name] = file_list
celebrity_file_names_dict
```

Output:

```
{'bhuvneshwar_kumar': ['./images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar1.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar10.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar11.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar12.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar13.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar14.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar2.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar3.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar4.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar5.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar6.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar7.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar8.png',
  './images/cropped/bhuvneshwar_kumar\\bhuvneshwar_kumar9.png'],
```

Step 11: Creating a Class Dictionary

```
class_dict = {}
count = 0
for celebrity_name in celebrity_file_names_dict.keys():
    class_dict[celebrity_name] = count
    count = count + 1
class_dict
```

Output:

```
{'bhuvneshwar_kumar': 0,
 'dinesh_karthik': 1,
 'hardik_pandya': 2,
 'jasprit_bumrah': 3,
 'k._l._rahul': 4,
 'kedar_jadhav': 5,
 'kuldeep_yadav': 6,
 'mohammed_shami': 7,
 'ms_dhoni': 8,
 'ravindra_jadeja': 9,
 'rohit_sharma': 10,
 'shikhar_dhawan': 11,
 'vijay_shankar': 12,
 'virat_kohli': 13,
 'yuzvendra_chahal': 14}
```

Images in cropped folder can be used for model training. We will use these raw images along with wavelet transformed images to train our classifier. Let's prepare X and y now

Step 12: Preparing the Feature Vectors

We're getting everything ready to train an image classifier. First, we go through each celebrity's pictures, making sure they're all the same size. Then, we use the Wavelet Transform to pull out the high-frequency parts from the images.

After that, we stack all these processed images together and add them to our list of features, which we call X. At the same time, we also add the corresponding labels to a separate list called y. This step is important because it ensures our dataset is set up correctly for training the classifier. Each image is represented by its features and has an associated label.

```
X, y = [], []
for celebrity_name, training_files in celebrity_file_names_dict.items():
    for training_image in training_files:
        img = cv2.imread(training_image)
        if img is None:
            continue
        scalled_raw_img = cv2.resize(img, (32, 32))
        img_har = w2d(img,'db1',5)
        scalled_img_har = cv2.resize(img_har, (32, 32))
        combined_img =
np.vstack((scalled_raw_img.reshape(32*32*3,1),scalled_img_har.reshape(32*32,1)))
        X.append(combined_img)
        y.append(class_dict[celebrity_name])
```

Step 13: Training the SVM Classifier

We will use SVM with rbf kernel tuned with heuristic finetuning

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC(kernel = 'rbf', C = 10))])
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Output:

```
0.5166666666666667
```

Now let's evaluate the model.

```
print(classification_report(y_test, pipe.predict(X_test)))
```

Output:

```
              precision    recall  f1-score   support

           0       0.50      0.25      0.33         4
           1       0.00      0.00      0.00         3
           2       0.33      0.50      0.40         2
           3       0.00      0.00      0.00         0
           4       0.40      0.40      0.40         5
           5       0.00      0.00      0.00         3
           6       1.00      0.33      0.50         3
           7       1.00      1.00      1.00         2
           8       0.47      0.78      0.58         9
           9       0.00      0.00      0.00         2
          10       0.56      0.75      0.64        12
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         3
          13       0.73      0.89      0.80         9

    accuracy                           0.52        60
   macro avg       0.36      0.35      0.33        60
weighted avg       0.45      0.52      0.46        60
```

Conclusion

In conclusion, becoming a pro at extracting and classifying image features using OpenCV and Python involves really understanding image processing, being super careful with feature extraction, and using top-notch classifiers. Throughout this journey, we learned the crucial steps of setting up the OpenCV environment, exploring different feature extraction techniques, and creating a powerful image classifier.

40+ Top Computer Vision Projects [2025 Updated] 4 min read