## Syntax of cv2.rotate()

```
rotate(src, rotateCode)
```

where

| Parameter | Description |
|---|---|
| src | 2D array. The source image that is read into a 2D array. |
| rotateCode | cv2.ROTATE_90_CLOCKWISE to rotate the image by **90** degrees clockwise.<br>cv2.ROTATE_180 to rotate the image by **180** degrees.<br>cv2.ROTATE_90_COUNTERCLOCKWISE to rotate the image by **270** degrees clockwise, or **90** degrees counter clockwise. |

**Steps to rotate an image**

1. Read an image into an array using cv2.imread() function.
2. Rotate the image (2D array) by 90 degrees, 180 degrees, or 270 degrees, using cv2.rotate() function with the respective rotate code. The function returns the rotated image (2D array).
   **rot=cv2.rotate(src,cv2.ROTATE_()_CLOCKWISE)**
3. Display the rotated image using plt.imshow() function.

## Syntax of cv2.resize()

```
cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

where

- src is the source, original or input image in the form of numpy array
- dsize is the desired size of the output image, given as tuple
- fx is the scaling factor along X-axis or Horizontal axis
- fy is the scaling factor along Y-axis or Vertical axis
- interpolation could be one of the following values.
  - INTER_NEAREST
  - INTER_LINEAR
  - INTER_AREA
  - INTER_CUBIC
  - INTER_LANCZOS4

Based on the interpolation technique selected, respective algorithm is used. You can think interpolation as a method that decides which pixel gets which value based on its neighboring pixels and the scale at which the image is being resized.

**How could you resize an image?**

You can resize an image in three ways.

1. Preserve the Aspect Ration and increase or decrease the width and height of the image. Just to make things clear, Aspect Ratio is the ratio of image width to image height.
2. Scale the image only along X-axis or Horizontal axis. Meaning, change width, keeping height same as that of original image.
3. Scale the image only along Y-axis or Vertical axis. Meaning, change height, keeping width same as that of original image.

**Steps to scale(resize) an image**

1. Read an image into an array using cv2.imread() function.
2. Set **newdimension=(newwidht,h).**
3. Use **cv2.resize(srouceimage,newdimension,interpolation=INTER_AREA).** The function returns the resized image.
4. Display the resized image using plt.imshow() function.

**Syntax of cv2.wrapAffine()- can be used for all type of transformation**

```
1.  cv2.warpAffine(src, Mat, dsize, dst, flags, borderMode, borderValue)
```

Parameters

- **src:** Source image or input image
- **dst:** Output image that has the size and the same type as the source image
- **Mat:** The transformation matrix
- **dsize:** Output image size
- **flags:** Interpolation methods
- **borderMode:** The method for pixel interpolation
- **borderValue:** Value used for the constant border. By default, the borderValue parameter is set as 0.

Translation is the shifting of an image along the *x*- and *y*-axis. To translate an image using OpenCV, we must:

1. Load an image from disk

   **# Read an image**

   **img = cv2.imread('D:/downloads/opencv_logo.PNG')**

   **rows,cols,_ = img.shape**
2. Define an affine transformation matrix

   **# Create the transformation matrix**

   **M = np.float32([[1,0,100],[0,1,50]])**

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

3. Apply the **cv2.warpAffine** function to perform the translation. The function returns the transformed image.

   **# Pass it to warpAffine function**
   **dst = cv2.warpAffine(img,M,(cols,rows))**
4. Display the translated image using plt.imshow() function.

**# Displaying the translate image**
**plt.imshow(dst)**

## Syntax of cv2.Canny(): For Edge Detection

***Syntax:*** *cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)*

***Where:***

- *Image: Input image to which Canny filter will be applied*
- *T_lower: Lower threshold value in Hysteresis Thresholding*
- *T_upper: Upper threshold value in Hysteresis Thresholding*
- *aperture_size: Aperture size of the Sobel filter.*
- *L2Gradient: Boolean parameter used for more precision in calculating Edge Gradient.*

**Steps to detect edge of an image**

1. Read an image into an array using cv2.imread() function.
2. Convert image to grayscale

   **gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)**

3. Setting parameter values

   **t_lower = 50  # Lower Threshold**
   **t_upper = 150  # Upper threshold**

4. Applying the Canny Edge filter

   **edge = cv2.Canny(img, t_lower, t_upper)**

5. Display the rotated image using plt.imshow() function.

   **plt.imshow(edge)**

**Syntax of cv2.blur():**

- The **kernel** corresponds to the number of pixels we consider when blurring each individual pixel. Larger kernels spread the blur around a wider region, as each pixel is modified by more of its surrounding pixels.

***Syntax:*** *cv2.blur(src, ksize[, dst[, anchor[, borderType]]])*

***Parameters:***

***src:*** *It is the image whose is to be blurred.*

***ksize:*** *A tuple representing the blurring kernel size.*

***dst:*** *It is the output image of the same size and type as src.*

***anchor:*** *It is a variable of type integer representing anchor point and it's default value Point is (-1, -1) which means that the anchor is at the kernel center.*

***borderType:*** *It depicts what kind of border to be added. It is defined by flags like* ***cv2.BORDER_CONSTANT****,* ***cv2.BORDER_REFLECT****, etc*

***Return Value:*** *It returns an image.*

**Steps to blur an image**

1. Read an image into an array using cv2.imread() function.
2. Setting kernel size
   **ksize = (10, 10)**

3. # Using cv2.blur() method
   **Blur_image = cv2.blur(image, ksize)**

4. Display the rotated image using plt.imshow() function.
   **plt.imshow(Blur_image)**

**Syntax of cv2.findContours():**Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. Contours come handy in shape analysis, finding the size of the object of interest, and object detection. OpenCV has **findContour()** function that helps in extracting the contours from the image. It works best on binary images, so we should first apply canny edges.

**Steps to find contour of an image**

1. Read an image into an array using cv2.imread() function.
2. Convert image to grayscale
   **gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)**

3. Setting parameter values
   **t_lower = 50  # Lower Threshold**
   **t_upper = 150  # Upper threshold**

4. Applying the Canny Edge filter
   **edge = cv2.Canny(img, t_lower, t_upper)**
5. Finding Contours

**contours, hierarchy = cv2.findContours(edged,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)**

- First parameter is source image,
- second is contour retrieval mode,
- third is contour approximation method and
- it outputs the image(altered), contours, and hierarchy. '*contours*' is a Python list of all the contours in the image.
- Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object.

6. **# Draw all contours**
   # -1 signifies drawing all contours
   **cv2.drawContours(image, contours, -1, (0, 255, 0), 3)**
   - First parameter is source image
   - Second is no. of contours,
   - Third,-1 specifies drall all contours
   - Fourth is color (r,g,b) →(0,255,0) indicates green color
   - Fifth is thickness of contour drawing

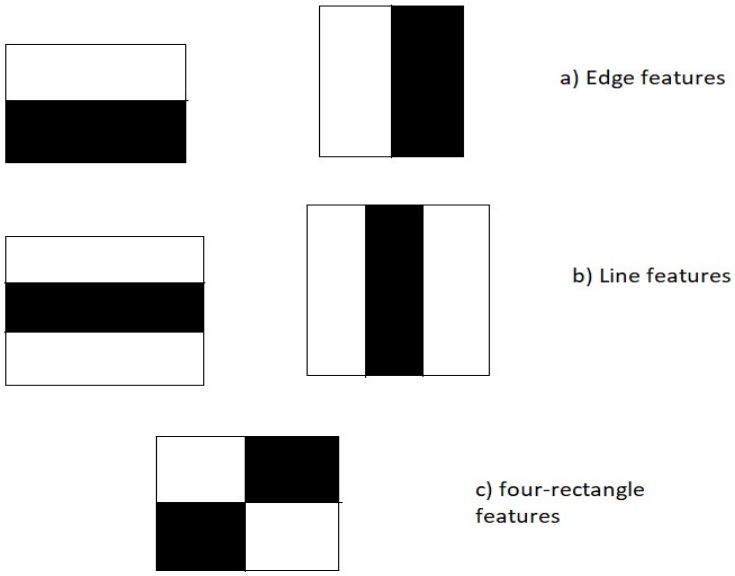
## Face detection using Cascade Classifier using OpenCV-Python

- We are going to see how to detect faces using a cascade classifier in OpenCV Python.
- Face detection has much significance in different fields of today's world. It is a significant step in several applications, face recognition (also used as biometrics), photography (for auto-focus on the face), face analysis (age, gender, emotion recognition), video surveillance, etc.
- One of the popular algorithms for facial detection is "**haarcascade**". It is computationally less expensive, a fast algorithm, and gives high accuracy.

**Haarcascade file can be download from here:** haarcascade_frontalface_default.xml

It works in four stages:

- **Haar-feature selection**: A Haar-like feature consists of dark regions and light regions. It produces a single value by taking the difference of the sum of the intensities of the dark regions and the sum of the intensities of light regions. It is done to extract useful elements necessary for identifying an object. The features proposed by viola and jones are:




- **Creation of Integral Images**: A given pixel in the integral image is the sum of all the pixels on the left and all the pixels above it. Since the process of extracting Haar-like features involves calculating the difference of dark and light rectangular regions, the introduction of Integral Images reduces the time needed to complete this task significantly.
- **AdaBoost Training**: This algorithm selects the best features from all features. It combines multiple "weak classifiers" (best features) into one "strong classifier". The generated "strong classifier" is basically the linear combination of all "weak classifiers".
- **Cascade Classifier**: It is a method for combining increasingly more complex classifiers like AdaBoost in a cascade which allows negative input (non-face) to be quickly discarded while spending more computation on promising or positive face-like regions. It significantly reduces the computation time and makes the process more efficient.

OpenCV comes with lots of pre-trained classifiers. Those XML files can be loaded by cascadeClassifier method of the cv2 module. Here we are going to use haarcascade_frontalface_default.xml for detecting faces.

**Stepwise Implementation:**

**Step 1: Loading the image**

```
img = cv2.imread('image.jpg')
```

**Step 2: Converting the image to grayscale**

Initially, the image is a three-layer image (i.e., RGB), So It is converted to a one-layer image (i.e., grayscale).

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

**Step 3: Loading the required haar-cascade XML classifier file**

CascadeClassifier method in cv2 module supports the loading of haar-cascade XML files. Here, we need "haarcascade_frontalface_default.xml" for face detection.

```
face_classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)
```

**Step 4: Applying the face detection method on the grayscale image**

- This is done using the **cv2::CascadeClassifier::detectMultiScale** method, which returns boundary rectangles for the detected faces (i.e., x, y, w, h).
- It takes two parameters namely, **scaleFactor and minNeighbors**.
    - **ScaleFactor** determines the factor of increase in window size which initially starts at size "minSize", and after testing all windows of that size, the window is scaled up by the "scaleFactor", and the window size goes up to "maxSize". If the "scaleFactor" is large, (e.g., 2.0), there will be fewer steps, so detection will be faster, but we may miss objects whose size is between two tested scales. (default scale factor is 1.3).
    - Higher the values of the **"minNeighbors"**, less will be the number of false positives, and less error will be in terms of false detection of faces. However, there is a chance of missing some unclear face traces as well.

```
faces_rect = face_classifier.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=9)
```

**Step 5: Iterating through rectangles of detected faces**

Rectangles are drawn around the detected faces by the rectangle method of the cv2 module by iterating over all detected faces.

```
for (x, y, w, h) in faces_rect:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), thickness=2)

plt.figure(figsize=(20,10))
plt.imshow(img)
plt.axis('off')
```