OpenCV Cheat Sheet

In [ ]:

```
import cv2
```

GUI

In [ ]:

```
cv2.imshow('image', img)
cv2.waitKey(0)   # 0=indefinitely, otherwise delay in ms
cv2.destroyAllWindows()
```

In [ ]:

```
cv2.namedWindow('window')
cv2.setMouseCallback('Image mouse', mouse_callback, param=None)
def mouse_callback(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN | cv2.EVENT_LBUTTONUP |cv2.EVENT_LBUTTOND
                cv2.EVENT_MOUSEMOVE | cv2.EVENT_MOUSEWHEEL:
    pass
```

Colors

In [ ]:

```
b = img_OpenCV[:, :, 0]
g = img_OpenCV[:, :, 1]
r = img_OpenCV[:, :, 2]
# --- or ---
b, g, r = cv2.split(img)

img = = cv2.merge((r, g, b))

img_rgb = img_bgr[:, :, ::-1]
# --- or ---
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

In [ ]:

```
img_gry = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
img_col = cv2.applyColorMap(img_gry, cv2.COLORMAP_JET)
```

Image Manipulation

In [ ]:

```
pix_b = img[0, 0, 0]
pix_bgr = img[0, 0]
img_b = img[:, :, 0]
img_slice = img[0:10, 0:20]

img[:] = 128  # img.fill(128)
img[:, :, 0] = 0
```

In [ ]:

```
# stack horizontally
img_lr = np.concatenate((img_l, img_r), axis=1)
```

File I/O

In [ ]:

```
img = cv2.imread('img.png')
img = cv2.imread('img.png', cv2.IMREAD_GRAYSCALE)
```

In [ ]:

```
cv2.imwrite('img.jpg', img)
```

Video

In [ ]:

```
capture = cv2.VideoCapture(0)  # 0=index_camera, also video filename
assert capture.isOpened()

width = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
height = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)
fps = capture.get(cv2.CAP_PROP_FPS)

while capture.isOpened():
    ret, frame = capture.read()
    if not ret: break
capture.release()
```

In [ ]:

```
fourcc = cv2.VideoWriter_fourcc(*'AVC1')
# https://gist.github.com/takuma7/44f9ecb028ff00e2132e
writer = cv2.VideoWriter(video_path, fourcc, fps, width, height, is_color)
writer.write(frame)
writer.release()
```

In [ ]:

```
# navigating video files
num_frames = capture.get(cv2.CAP_PROP_FRAME_COUNT)
capture.set(cv2.CAP_PROP_POS_FRAMES, <FRAME_INDEX>)
```

Drawing Shapes

In [ ]:

```
pt1, pt2 = (0, 0), (100, 100)
pts = np.array([[250, 5], [220, 80], [280, 80]], np.int32).reshape((-1, 1, 2))
color = (255, 255, 255)
lineType = cv2.LINE_4 | cv2.LINE_8 | cv2.LINE_AA
thicknes = -1  # fill shape
```

In [ ]:

```
cv2.line(img, pt1, pt2, color, thickness=1, lineType=8, shift=0)
cv.arrowedLine(img, pt1, pt2, color, thickness=1, lineType=8, shift=0, tipLengt
cv2.rectangle(img, pt1, pt2, color, thickness=1, lineType=8, shift=0)
cv2.circle(img, center, radius, color, thickness=1, lineType=8, shift=0)
cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color,
            thickness=1, lineType=8, shift=0)
cv2.polylines(img, pts, is_closed, color, thickness=1, lineType=8, shift=0)
```

In [ ]:

```
rect = (0, 0, 50, 50)
is_intersecting, pt1, pt2 = clipLine(rect, pt1, pt2)
```

Drawing Text

In [ ]:

```
font_face = cv2.FONT_HERSHEY_SIMPLEX or cv.FONT_HERSHEY_DUPLEX or ...
cv.putText(img, text, org, fontFace, fontScale, color,
           thickness=1, lineType=8, bottomLeftOrigin=False)  # !bottomLeftOrigi
```

In [ ]:
```
font_scale = cv2.getFontScaleFromHeight(fontFace, pixelHeight, thickness=1)
(width, height), baseLine = cv2.getTextSize(text, fontFace, fontScale, thicknes
```

Geometric Transformations

In [ ]:
```
interpolation = cv2.INTER_NEAREST | cv2.INTER_LINEAR | cv2.INTER_CUBIC |
                cv2.INTER_AREA | cv2.INTER_LANCZOS4
resized_img = cv2.resize(img, (width, height), interpolation=cv2.INTER_LINEAR)
resized_img = cv2.resize(img, None, fx=0.5, fy=0.5)
```

In [ ]:
```
# Translation
M = np.float32([[1, 0, translate_x],
                [0, 1, translate_y]])
```

In [ ]:
```
# Rotation
M = cv2.getRotationMatrix2D(center_height, center_width), angleDeg, scaleFactor
```

In [ ]:
```
pts_1 = np.float32([[0,0], [0,1], [1,0]])
pts_2 = np.float32([[1,1], [1,3], [4,1]])
M = cv2.getAffineTransform(pts_1, pts_2)
```

In [ ]:
```
# Affine Transformation
image = cv2.warpAffine(img, M, (output_height, output_width))
```

In [ ]:
```
# Perspective Transformaion
pts_1 = np.float32([[0,0], [0,1], [1,0], [1,1]])
pts_1 = np.float32([[0,0], [0,2], [2,0], [3,3]])
M = cv2.getPerspectiveTransform(pts_1, pts_2)
image = cv2.warpPerspective(img, M, (300, 300))
```

Image Filtering

In [ ]:
```
kernel = np.ones((5, 5), np.float32) / 25
# ddepth=-1 => output will have same depth as source
image = cv2.filter2D(img, ddepth, kernel)
```

```
      __Sharpening Kernels__                       __Sobel Kernels__                  __Laplacian Kernels__

 |||||----     |||||----|-    |||||----    |||||----     |||||----     |||||----    |||||----     |||||----
 |----|----|   ---|----||     |----|----|  |----|----|   |----|----|   |----|----|  |----|----|   |----|----|
 | 0 | -1 | 0 |  -1 | -1 | -1 |  | 1 | 1 | 1 |  | -1 | 0 | 1 |  | -1 | -2 | 1   | 1 | 1 | 1 |  | 0 | 1 | 0 |  | 1 | 4 | 1 |
 | -1 | 4 | -1   | -1 | 8 | -1 |  | 1 | -8 | 1   | -2 | 0 | 2    || 0 | 0 | 0    | 1 | -8 | 1    | 1 | -4 | 1    | 4 |-20 |
 || 0 | -1 | 0   | -1 | -1 | -1   || 1 | 1 | 1   || -1 | 0 | 1   || -1 | -2 |    || 1 | 1 | 1   || 0 | 1 | 0    4 || 1 | 4 |
 |             |              |            |            1 |           |            |            1 |
```

In [ ]:
```
# Unsharp Mask
smoothed = cv2.GaussianBlur(img, ksize, sigmaX)
# cv2.addWeighted(src1, alpha, src2, beta, gamma)
# dst = src1*alpha + src2*beta + gamma
unsharped = cv2.addWeighted(img, 1.5, smoothed, -0.5, 0)
```

In [ ]:
```
# Gausssian Blur
ksize = (width, height)
```

```
# sigmaX=0 => computed from ksize.width and ksize.height
image = cv2.GaussianBlur(img, ksize, sigmaX)
```

In [ ]:

```
# Median Blur
ksize1 = 5   # width == height
image = cv2.medianBlur(img, ksize1)
```

In [ ]:

```
# Bilateral Blur
# dia<0 => computed from sigmaSpatial
image = cv2.bilateralFilter(img, dia, sigmaColor, SigmaSpatial)
```

In [ ]:

```
# Canny Edge
image = cv.Canny(img, loThreshold1, hiThreshold, sobelApertSize=3)
```

Arithmetic Ops

In [ ]:

```
# Saturation Arithmetic
# src1, src2: array or scalar
image = cv2.add(src1, src2)
image = cv2.subtract(src1, src2)
```

In [ ]:

```
# Blending
image = cv2.addWeighted(src1, alpha, src2, beta, gamma)
```

In [ ]:

```
# Bitwise
image = cv2.bitwise_not(img)
image = cv2.bitwise_and(src1, src2)
image = cv2.bitwise_or(src1, src2)
image = cv2.bitwise_xor(src1, src2)
```

In [ ]:

```
# lowerb: inclusive lower-bound array/scalar
# upperb: inclusive upper-bound array/scalar
mask = cv2.inRange(img, lowerb, upperb)
```

Morphological Ops

In [ ]:

```
shape =  cv.MORPH_RECT | cv.MORPH_ELLIPSE | cv.MORPH_CROSS
cv2.getStructuringElement(shape, ksize)
```

In [ ]:

```
image = cv2.dilate(img, kernel, iterations=1)
image = cv2.erode(img, kernel, iterations=1)
```

In [ ]:

```
image = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)       # erosion → dilation
image = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)      # dilation → erosion
image = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)   # dilation - erosion
image = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)     # original - opening
image = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)   # closing - original
```

Histogram

NOTE: cv2.calcHist() is much faster than np.histogram() and plt.hist()

In [ ]:

```
# images: list of images
# channels: list of channel idxs, e.g. grayscale: [0], color: [0, 1, 2]
# mask : None => no mask
# histSize: list of # bins
# ranges: range of intensity to measure, e.g. [0, 256]

# cv2.calcHist([image], [channel], mask, [histSize], [range])
hist = cv2.calcHist([img_bgr], [0], None, [256], [0, 256])
```

In [ ]:

```
# Masks
mask = np.zeros((100, 100), np.uint8)
mask[10:90, 10:90] = 255
```

Histogram Equalization

In [ ]:

```
# Grayscale
image = cv2.equalizeHist(img_gry)

# Color
H, S, V = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
V_eq = cv2.equalizeHist(V)
image = cv2.cvtColor(cv2.merge([H, S, eq_V]), cv2.COLOR_HSV2BGR)
```

In [ ]:

```
# CLAHE
# cv2.createCLAHE(clipLimit, tileGridSize=(8,8))
clahe = cv2.createCLAHE(clipLimit=2.0)
image = clahe.apply(img_gry)
```

Thresholding

In [ ]:

```
threshType = cv2.THRESH_BINARY | cv2.THRESH_BINARY_INV | cv2.THRESH_TRUNC | cv2
retval, image = cv2.threshold(img, thresh, maxval, threshType)
```

In [ ]:

```
adaptMethod = cv2.ADAPTIVE_THRESH_MEAN_C | cv2.ADAPTIVE_THRESH_GAUSSIAN_C
# ADAPTIVE_THRESH_GAUSSIAN_C => cross-correlation with Gaussian window (sigma c
# blocksize: int
# threshOffs: constant subtracted from the (weighted) mean
image = adaptiveThreshold(img, maxValue, adaptMethod, threshType, blockSize, th
```