Computer vision in the new era of Artificial Intelligence and Deep Learning

Visión por computador en la nueva era de la Inteligencia Artificial y el Deep Learning

Rubén Usamentiaga*, Alberto Fernández°
*** University of Oviedo**
° TSK

Gijón (Spain)
5 – 16 April 2021

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

OpenCV

Image processing in OpenCV

- geometric_image_transformations_opencv.ipynb
- opencv_sliders_introduction_image_processing.ipynb
- visual_interface_image_processing_opencv.ipynb
- computational_photography_module_opencv.ipynb
- inpaint_algorithm_in_opencv.ipynb
- k_means_clustering_opencv.ipynb
- face_processing.ipynb
- references_for_main_image_processing_techniques_in_opencv.ipynb

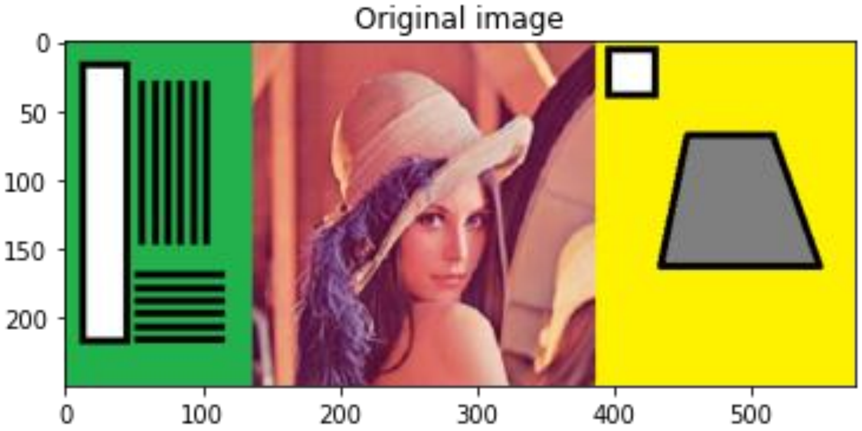
- geometric_image_transformations_opencv.ipynb
- opencv_sliders_introduction_image_processing.ipynb
- visual_interface_image_processing_opencv.ipynb
- computational_photography_module_opencv.ipynb
- inpaint_algorithm_in_opencv.ipynb
- k_means_clustering_opencv.ipynb
- face_processing.ipynb
- references_for_main_image_processing_techniques_in_opencv.ipynb

Open in Colab

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

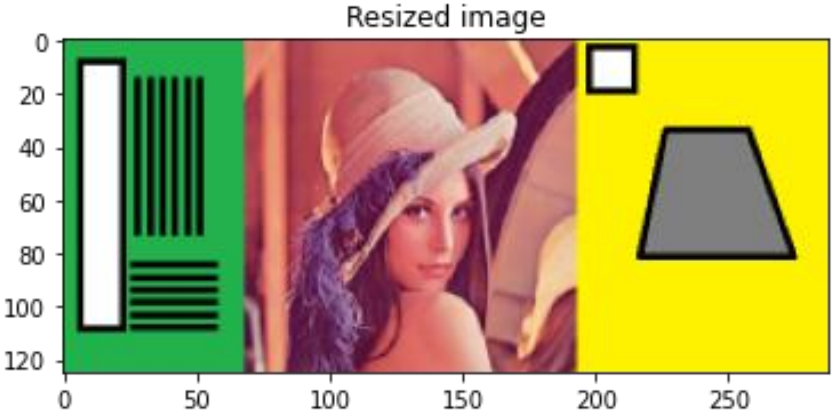
Geometric image transformations: resize

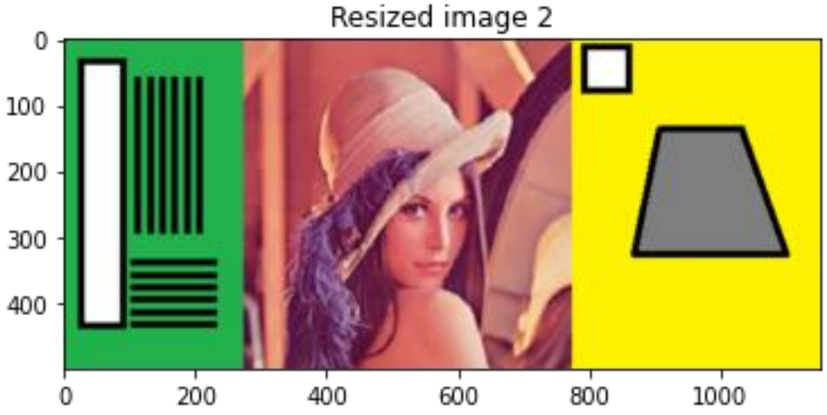
- For shrinking: cv2.INTER_AREA and cv2.INTER_CUBIC (slow)
- For zooming: cv2.INTER_LINEAR



```
cv2.resize(image, None, fx=0.5, fy=0.5,
interpolation=cv2.INTER_AREA)
```

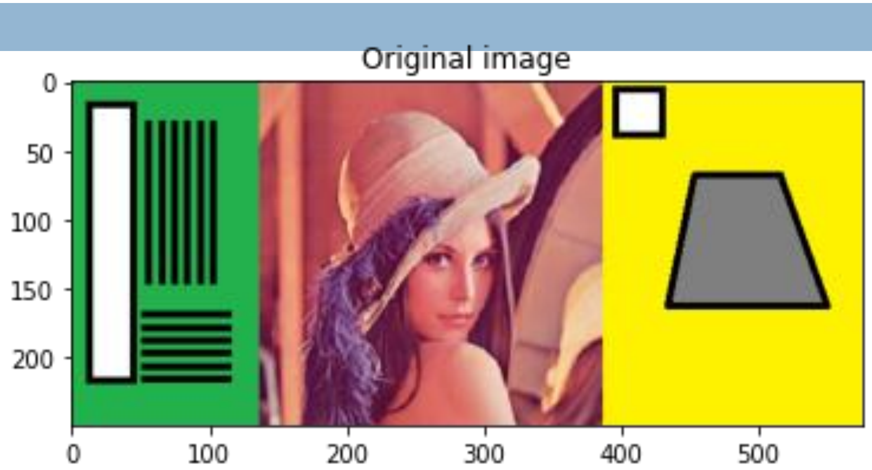
```
cv2.resize(image, (width * 2, height * 2),
interpolation=cv2.INTER_LINEAR)
```





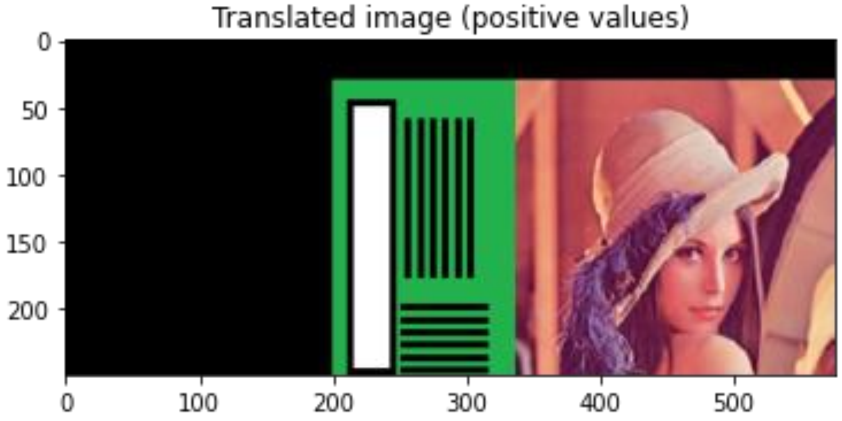
Geometric image transformations: translation

cv2.warpAffine() takes a 2x3 transformation matrix



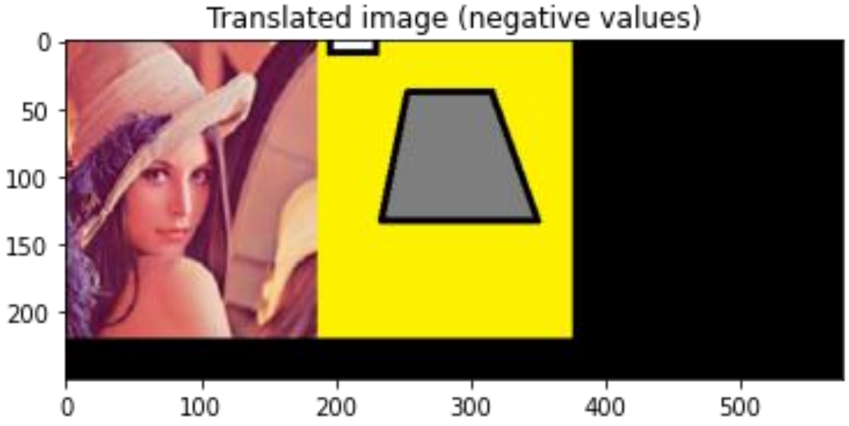
```
M = np.float32([[1, 0, 200], [0, 1, 30]])
```

```
cv2.warpAffine(image, M, (width, height))
```



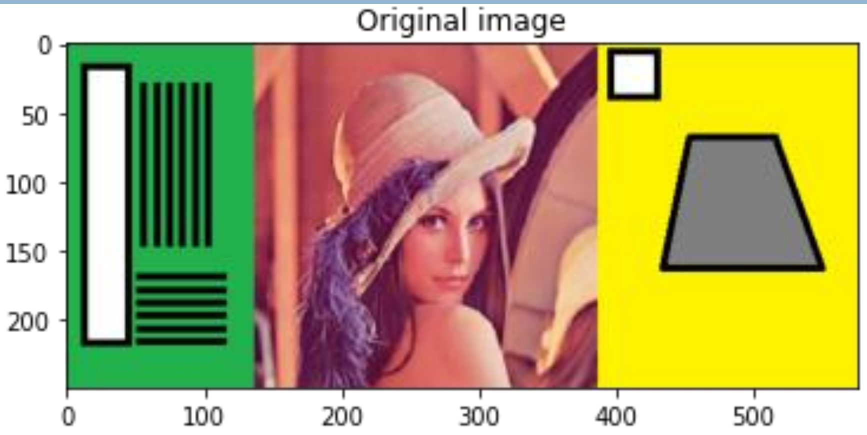
```
M = np.float32([[1, 0, -200], [0, 1, -30]])
```

```
cv2.warpAffine(image, M, (width, height))
```



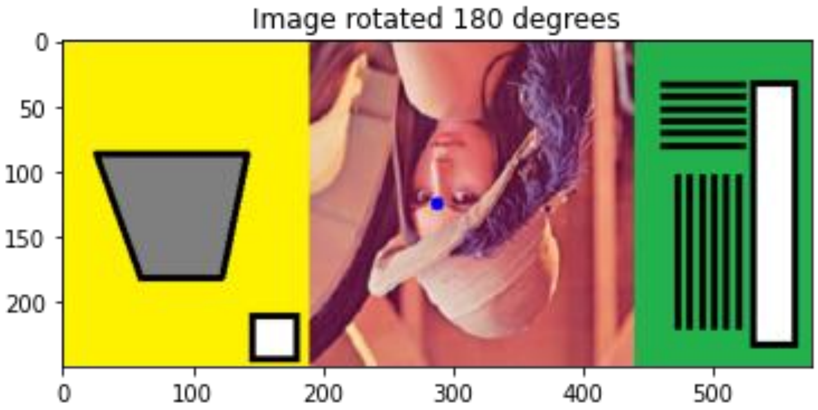
Geometric image transformations: rotation

cv2.warpAffine() takes a 2x3 transformation matrix



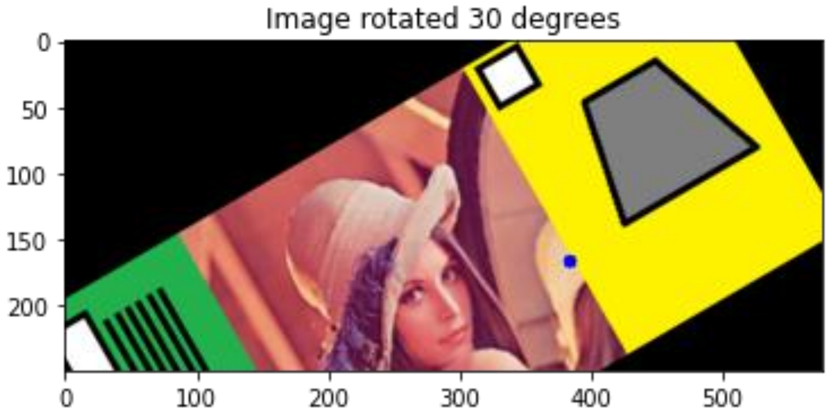
```
M = cv2.getRotationMatrix2D((width / 2.0,
height / 2.0), 180, 1)
```

```
cv2.warpAffine(image, M, (width, height))
```



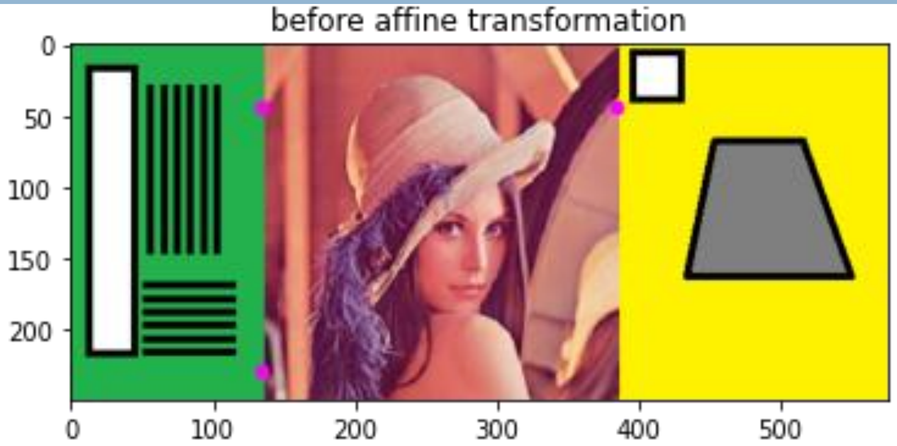
```
M = cv2.getRotationMatrix2D((width / 1.5,
height / 1.5), 30, 1)
```

```
cv2.warpAffine(image, M, (width, height))
```



Geometric image transformations: affine

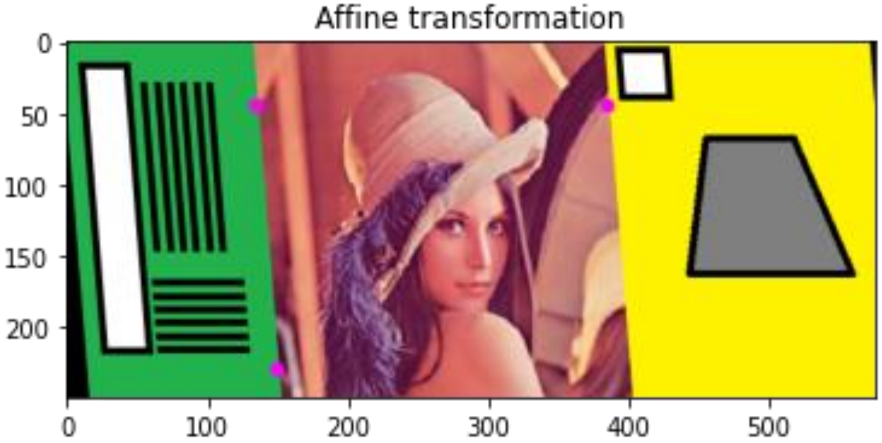
cv2.warpAffine() takes a 2x3 transformation matrix



```
pts_1 = np.float32([[135, 45], [385, 45], [135, 230]])
pts_2 = np.float32([[135, 45], [385, 45], [150, 230]])
```

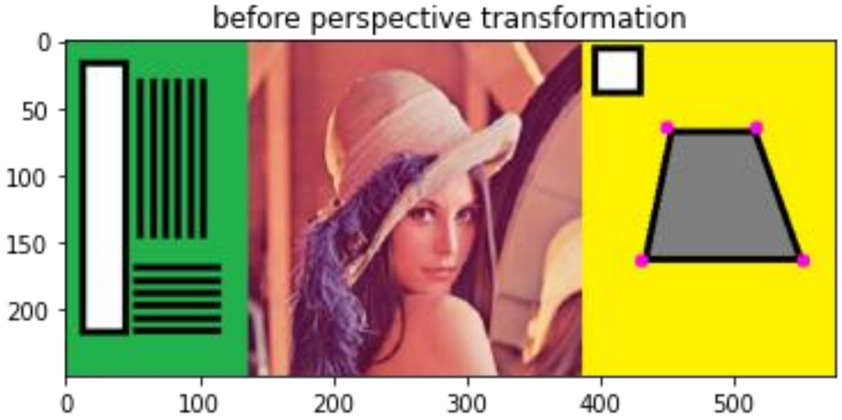
```
M = cv2.getAffineTransform(pts_1, pts_2)
```

```
cv2.warpAffine(image_points, M, (width, height))
```



Geometric image transformations: perspective correction

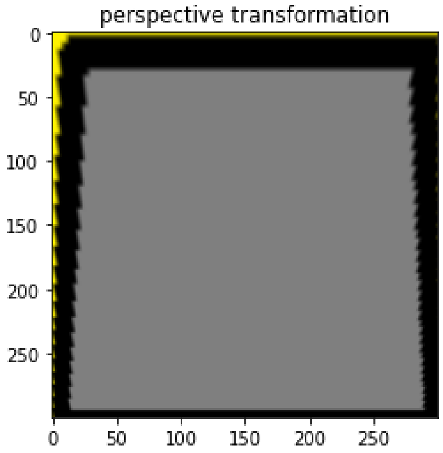
cv2.warpPerspective() takes a 3x3 transformation matrix



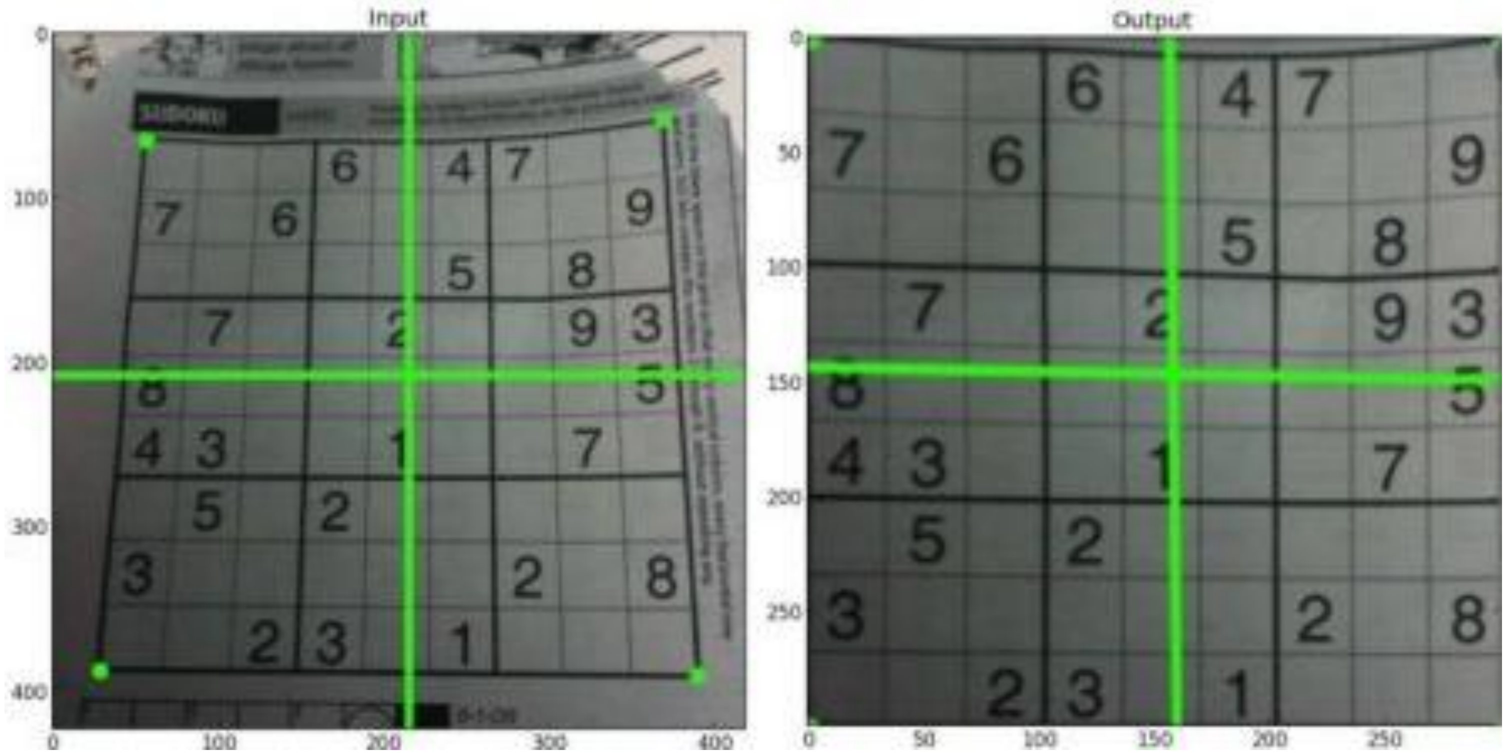
```
pts_1 = np.float32([[450, 65], [517, 65], [431, 164], [552, 164]])
pts_2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
```

```
M = cv2.getPerspectiveTransform(pts_1, pts_2)
```

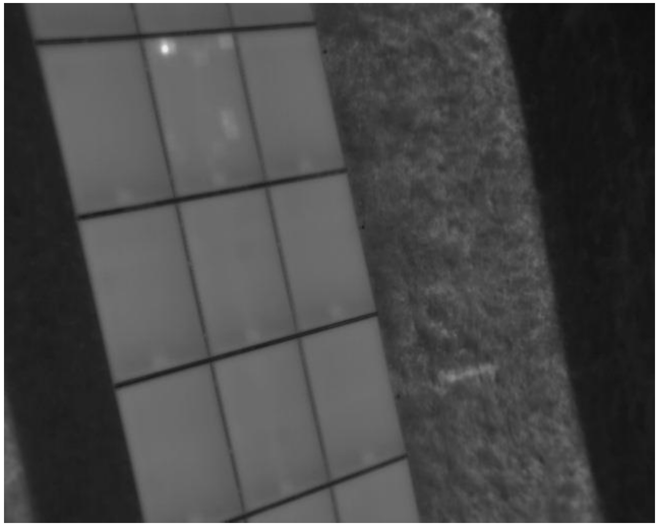
```
cv2.warpPerspective(image, M, (300, 300))
```

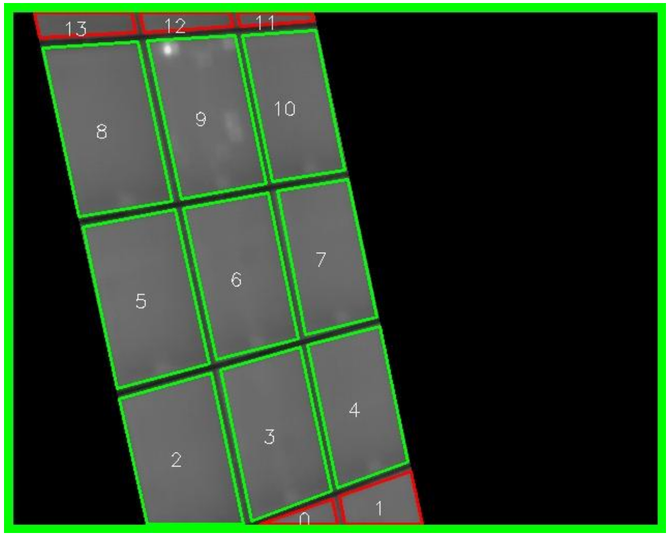


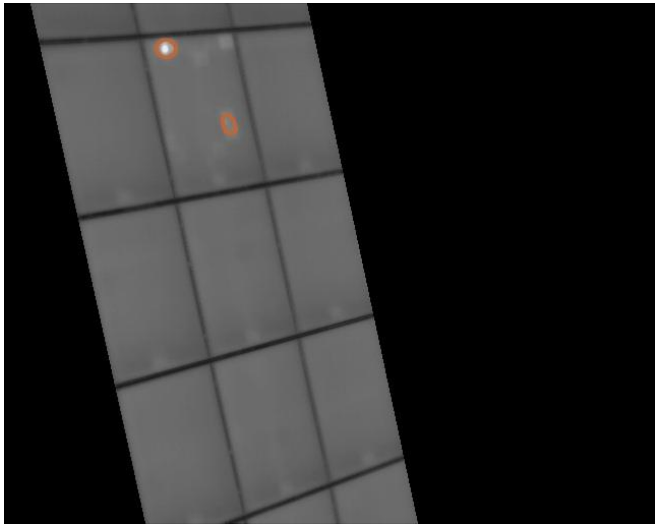
Geometric image transformations: perspective correction

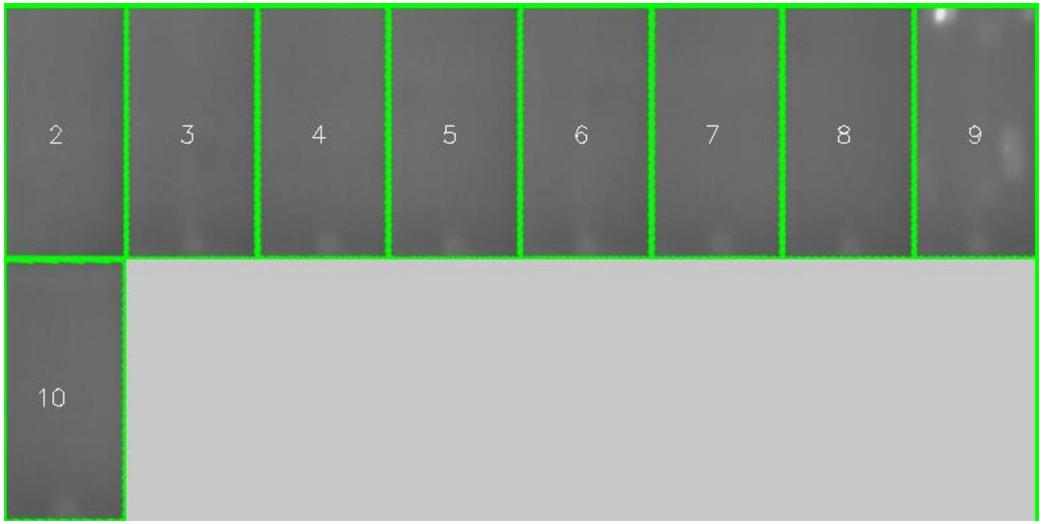


Geometric image transformations: perspective correction



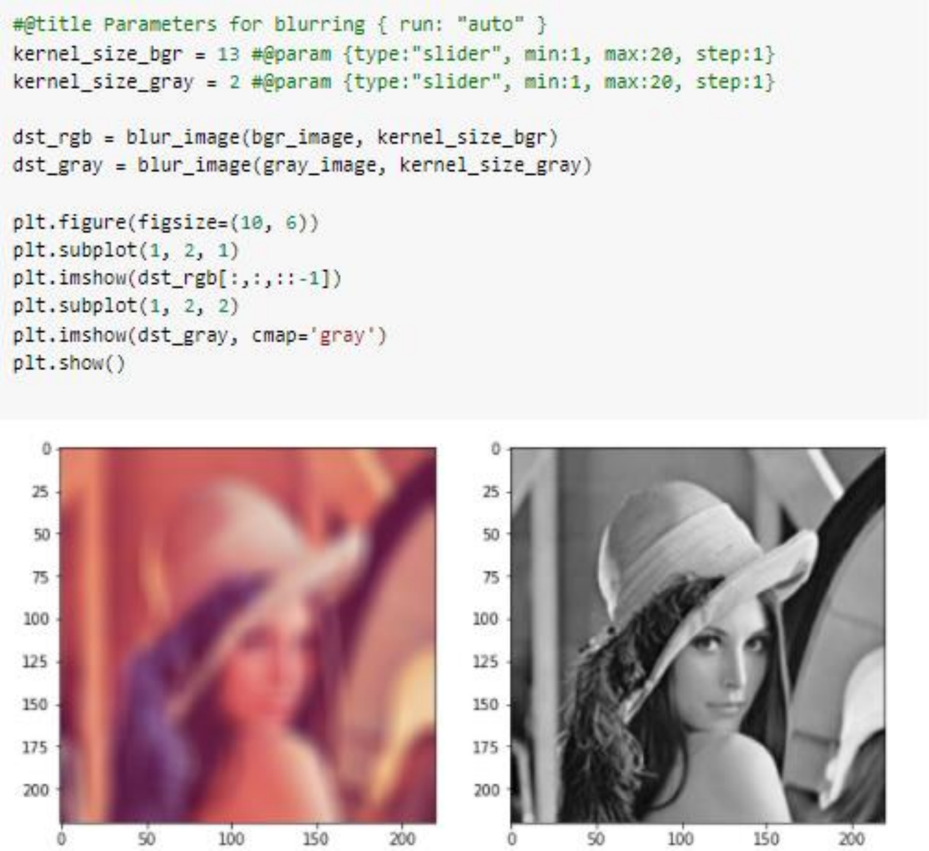






Sliders in Colab for image processing

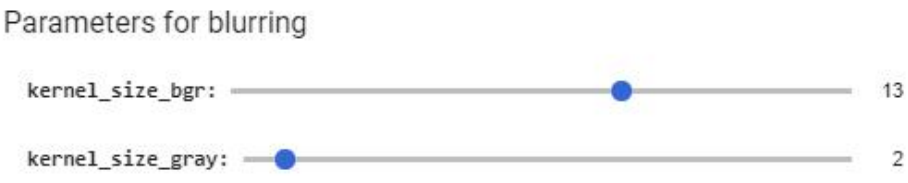
```
#@title Parameters for blurring { run: "auto" }
kernel_size_bgr = 13 #@param {type:"slider", min:1, max:20, step:1}
kernel_size_gray = 2 #@param {type:"slider", min:1, max:20, step:1}
```



```
#@title Parameters for blurring { run: "auto" }
kernel_size_bgr = 13 #@param {type:"slider", min:1, max:20, step:1}
kernel_size_gray = 2 #@param {type:"slider", min:1, max:20, step:1}

dst_rgb = blur_image(bgr_image, kernel_size_bgr)
dst_gray = blur_image(gray_image, kernel_size_gray)

plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
plt.imshow(dst_rgb[:,:,::-1])
plt.subplot(1, 2, 2)
plt.imshow(dst_gray, cmap='gray')
plt.show()
```



Parameters for blurring

kernel_size_bgr: 13

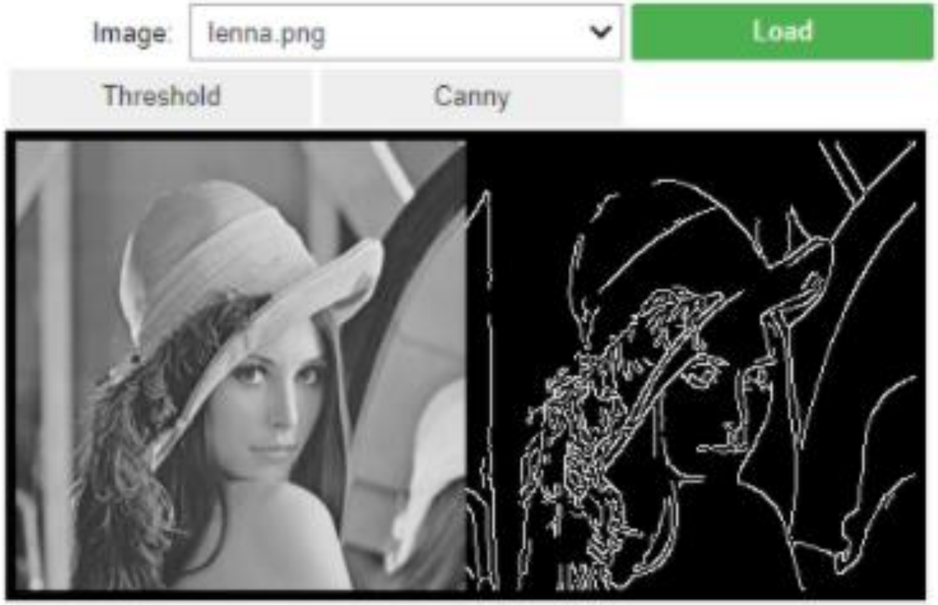
kernel_size_gray: 2

```
def blur_image(src, kernel_size):
  result = cv2.blur(src,(kernel_size
,kernel_size))
  return result
```

Visual interface for image processing

```
def display_menu():
  selector_box = widgets.HBox([file_selector, load_button])
  display(selector_box)
  button_box = widgets.HBox([thres_button, canny_button])
  display(button_box)
```

```
load_button=widgets.Button(description='Load', button_style='success')
```



```
thres_button=widgets.Button(description='Threshold')
```

```
canny_button=widgets.Button(description='Canny')
```

```
file_selector = widgets.Dropdown(options=path_images, description='Image:')
```

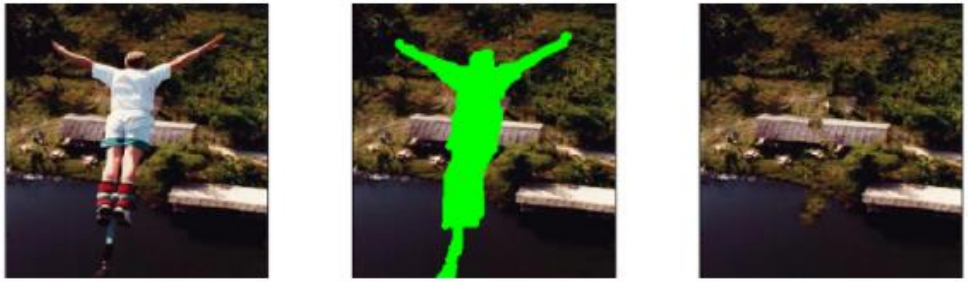
```
DIR_PATH = '/content'
included_extensions = ('.jpg','.jpeg', '.png')

path_images = [f for f in os.listdir(DIR_PATH) if any(f.endswith(ext) for ext in included_extensions)]
```

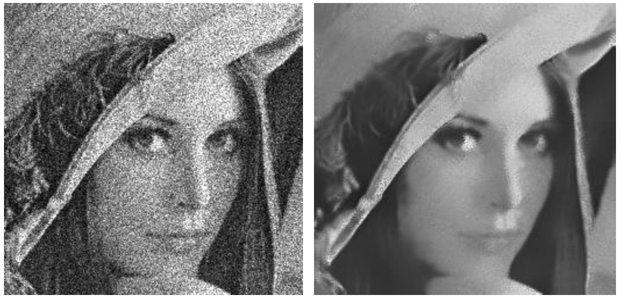
Full code in:
visual_interface_image_processing_opencv.ipynb

Computational photography module in OpenCV

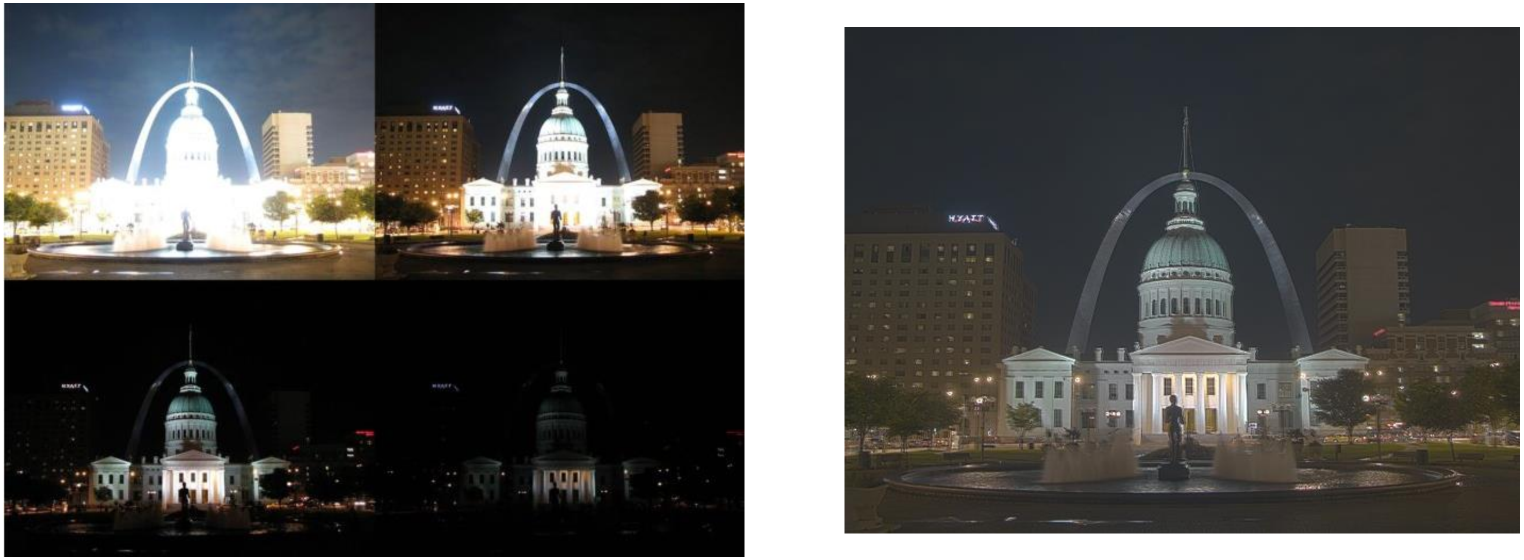
inpainting



Denoising

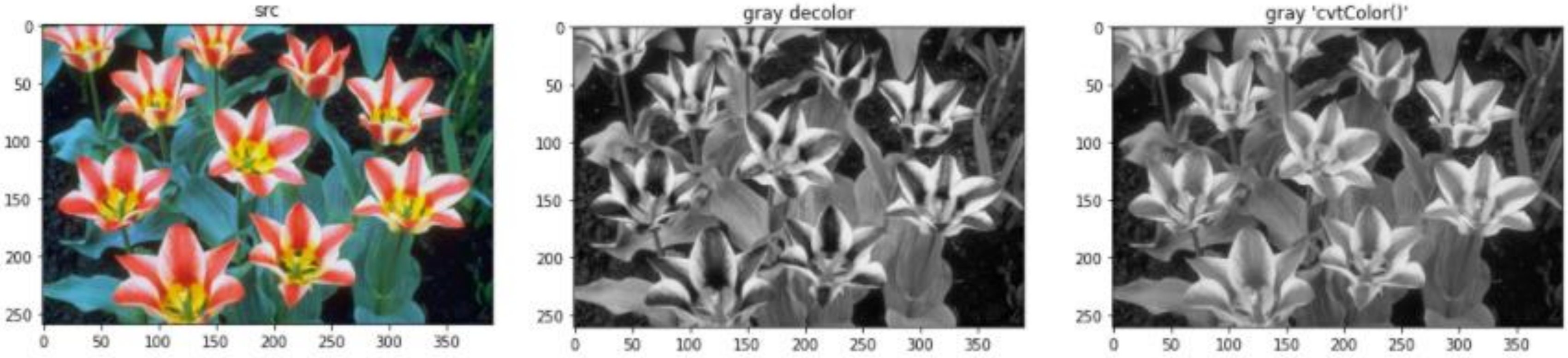


HDR imaging

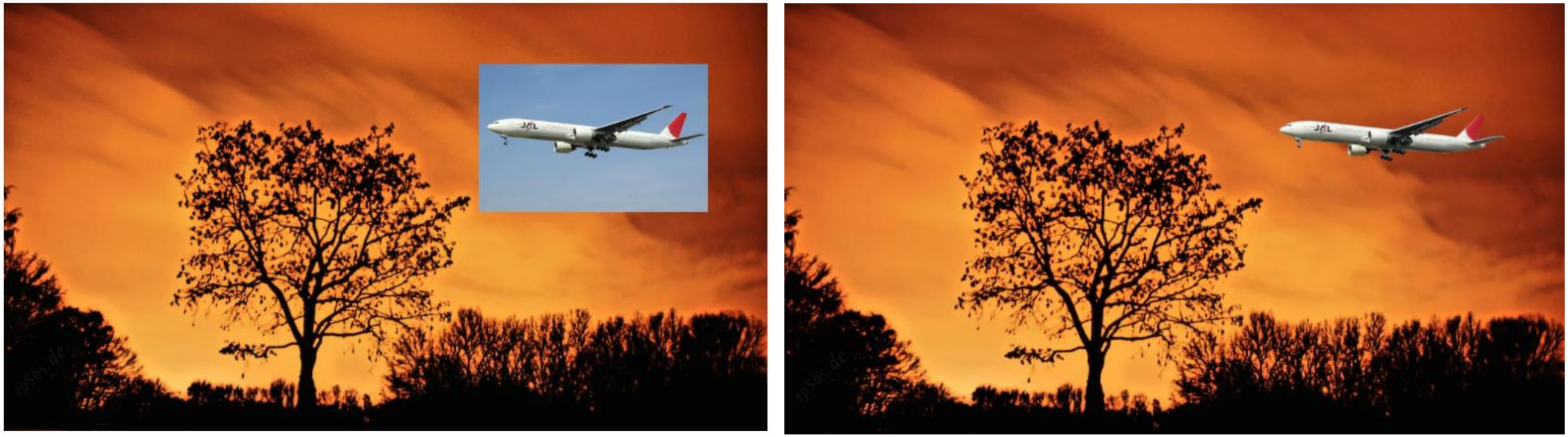


Computational photography module in OpenCV

Contrast Preserving Decolorization

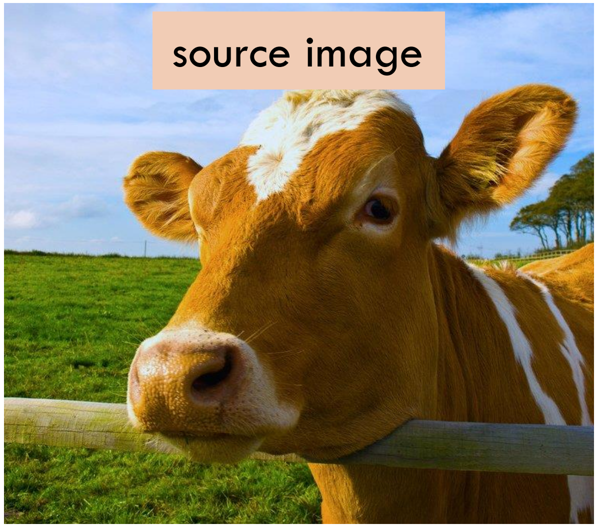


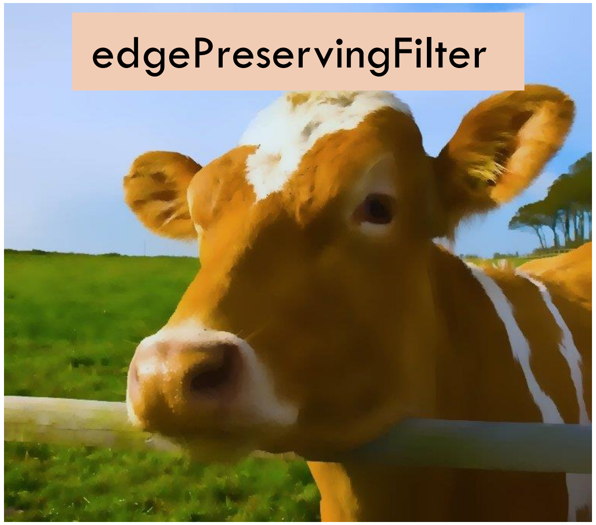
Seamless Cloning

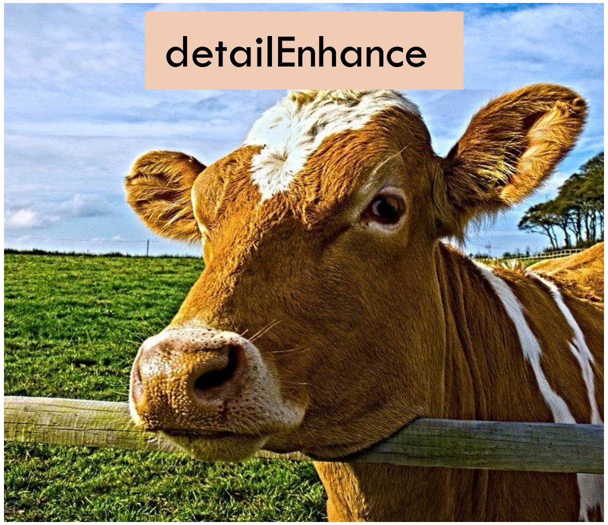


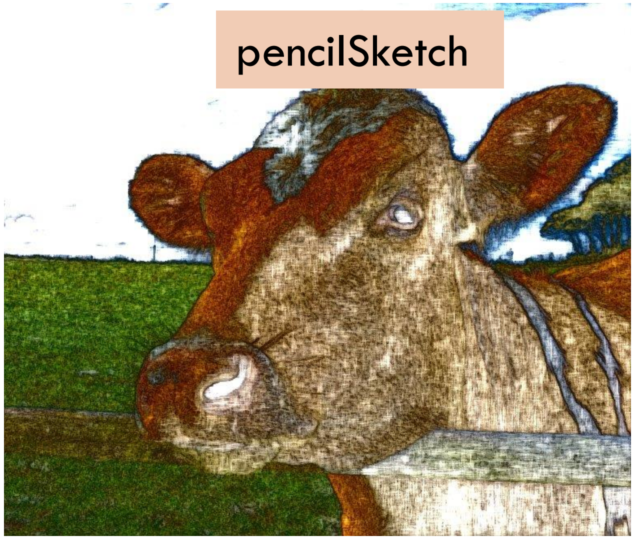
Computational photography module in OpenCV

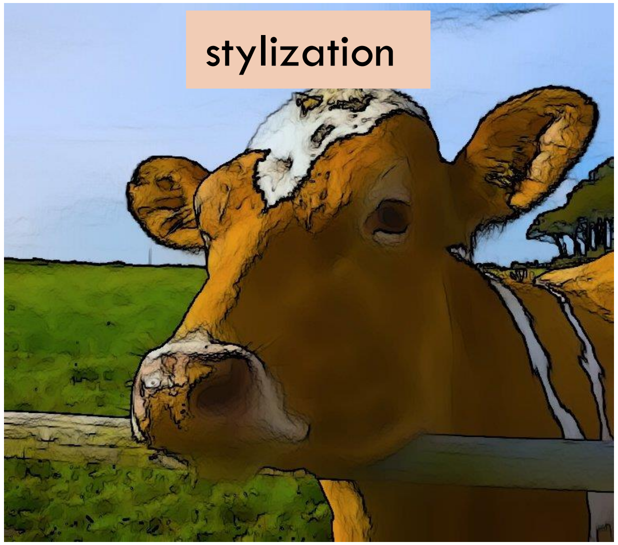
Non-Photorealistic Rendering


source image


edgePreservingFilter


detailEnhance


pencilSketch

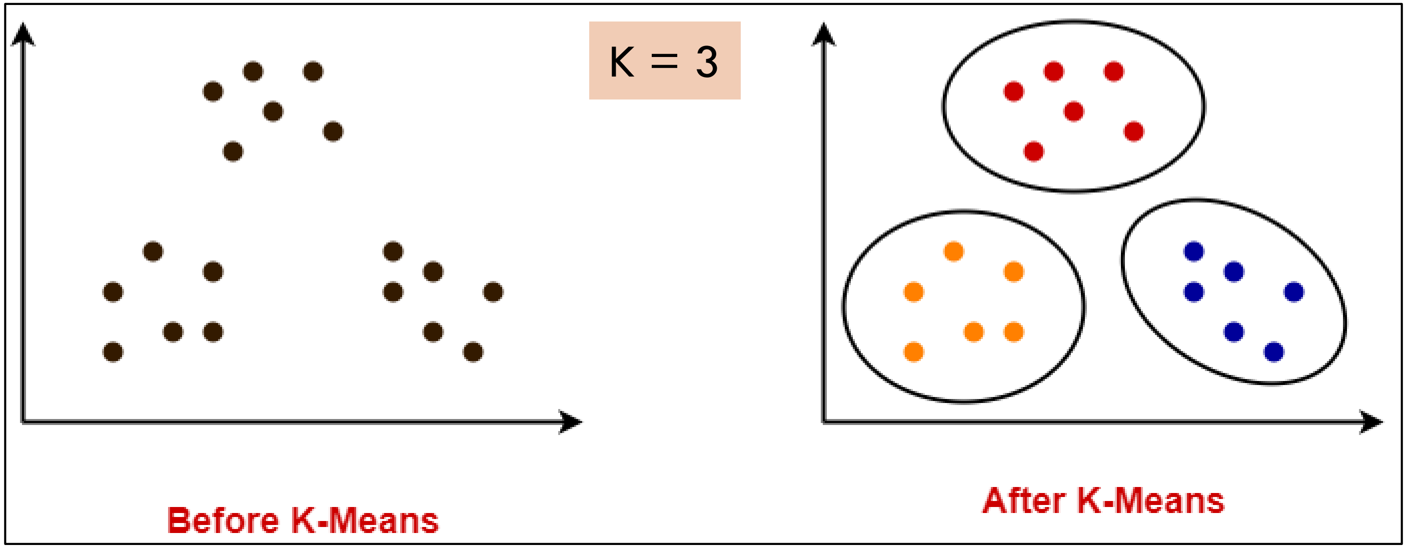

stylization

Link cow image

Images taken from here

K-means clustering OpenCV

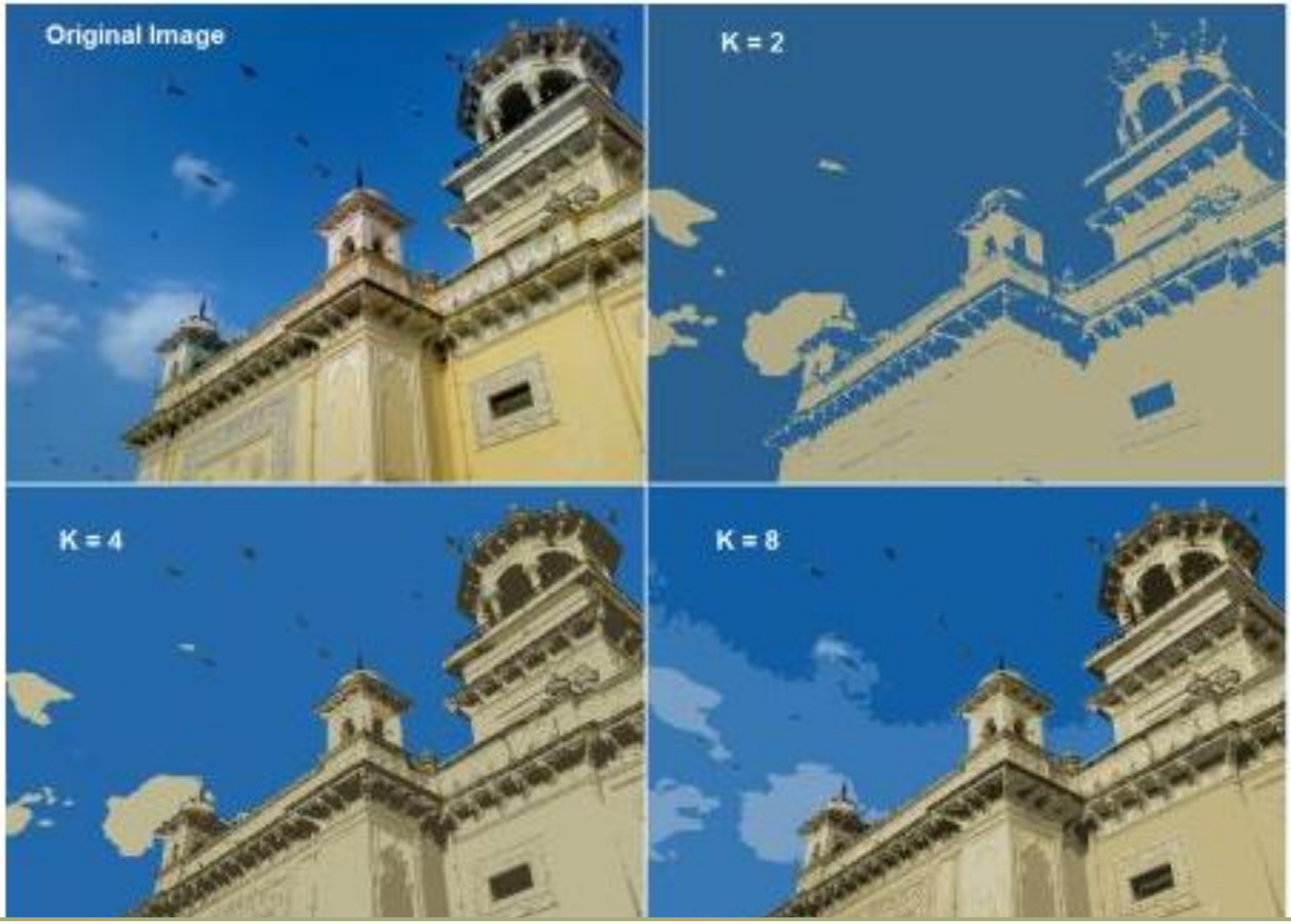
- k-means clustering is a method that aims to partition **n observations** into **k clusters** in which each observation belongs to the cluster with the nearest mean



K-means clustering OpenCV

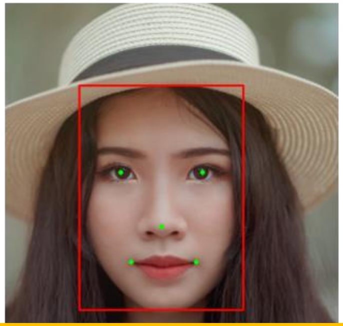
Color Quantization using K-means

Image taken from here



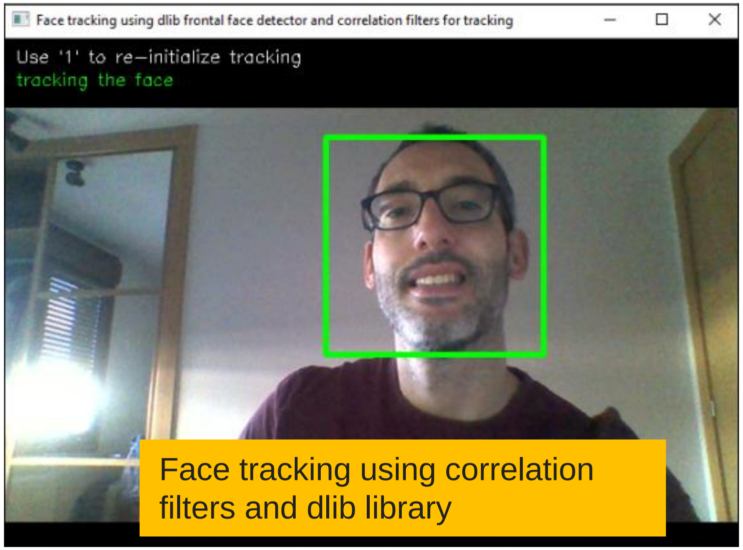
```
cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_PP_CENTERS)
```

Face processing



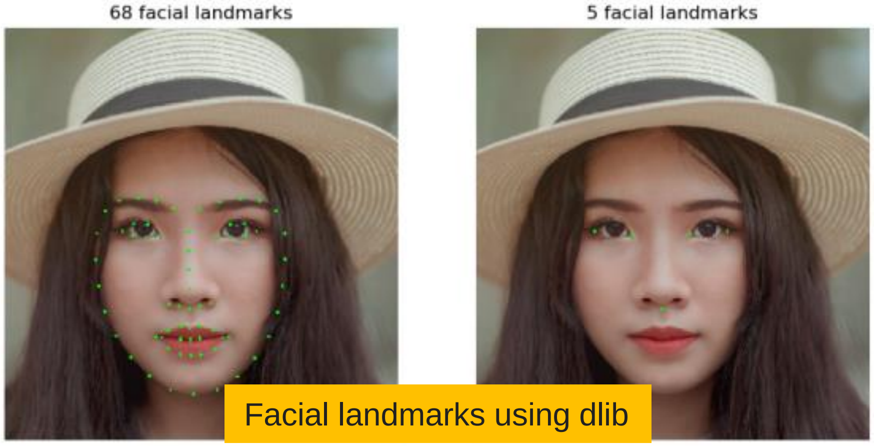
MTCNN: Face and facial landmarks detection

FER: Facial expression recognition

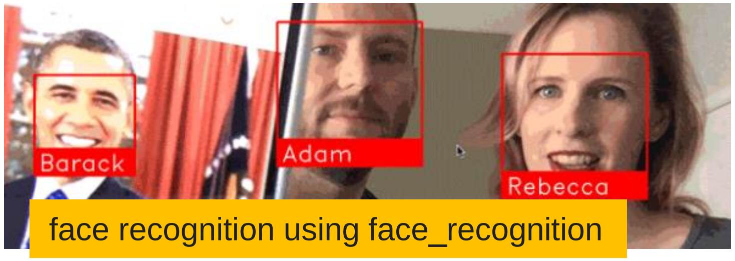

Face tracking using dlib frontal face detector and correlation filters for tracking

```
Use '1' to re-initialize tracking
tracking the face
```

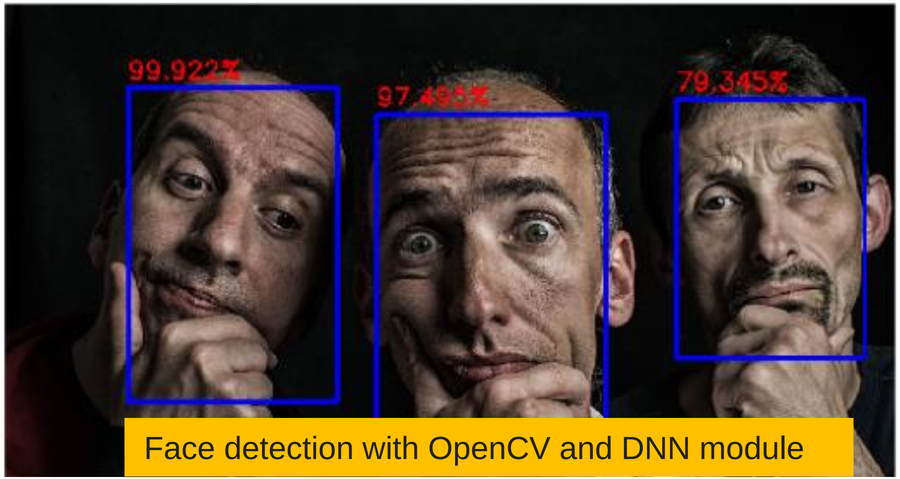
Face tracking using correlation filters and dlib library



Facial landmarks using dlib



face recognition using face_recognition



Face detection with OpenCV and DNN module

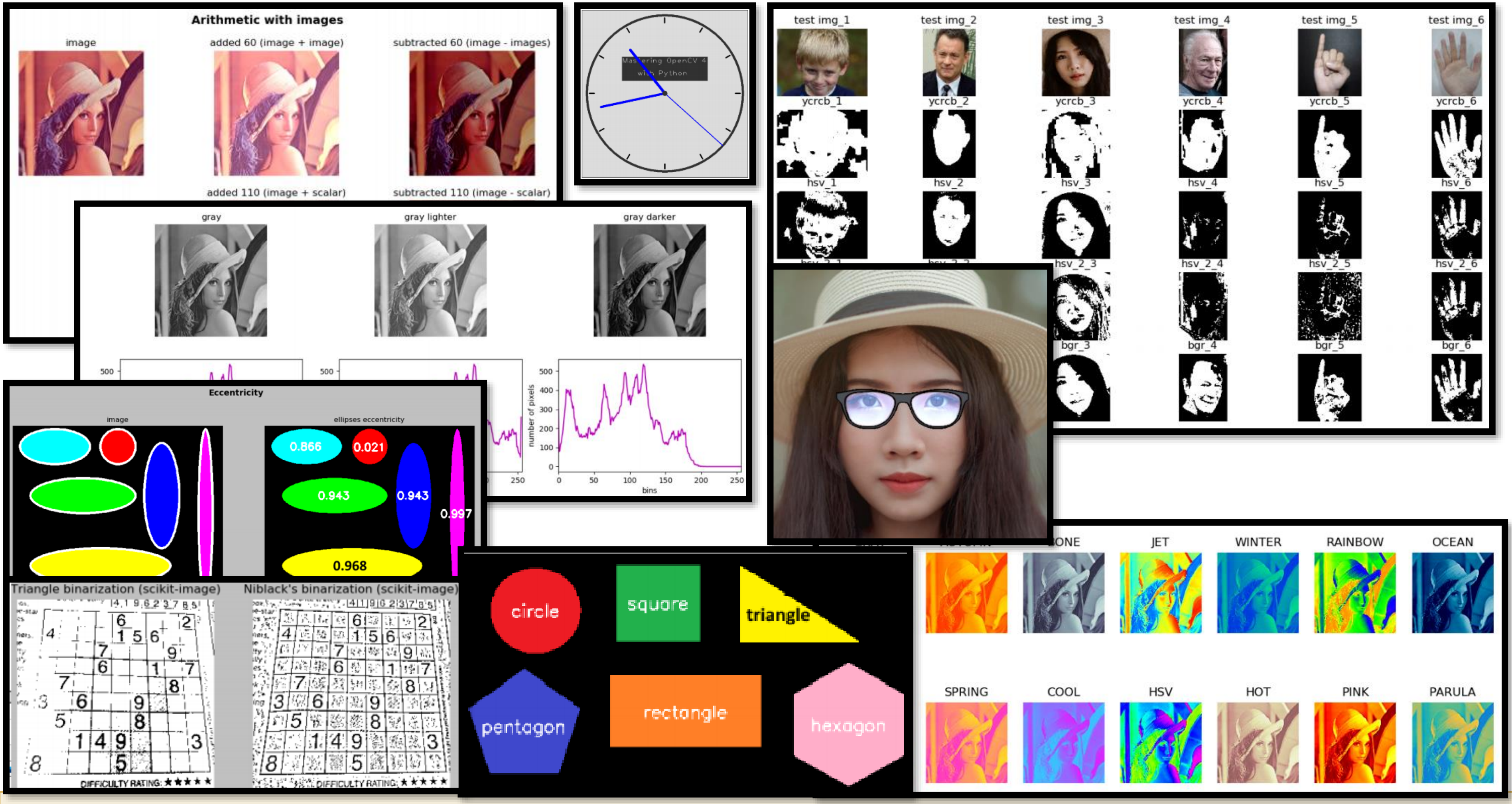
See face_processing.ipynb for implementations and references

Summary of image processing in OpenCV

references_for_main_image_processing_techniques_in_opencv.ipynb

Open in Colab

references_for_main_image_processing_techniques_in_opencv.ipynb



OpenCV

Image processing in OpenCV

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course