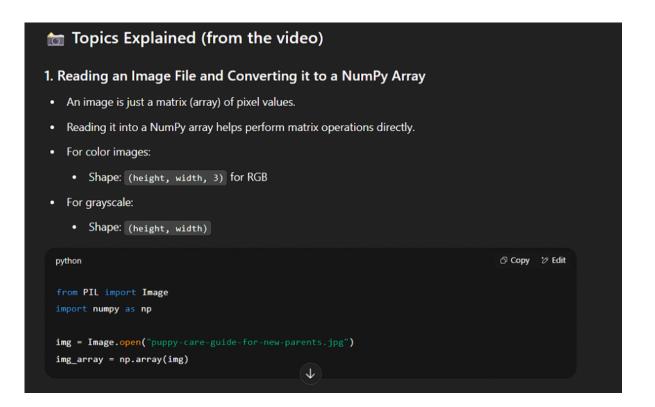
Sunday, July 27, 2025 6:43 PM

Image data processing - conceptual video: https://youtu.be/Hs5RhjpQVRo

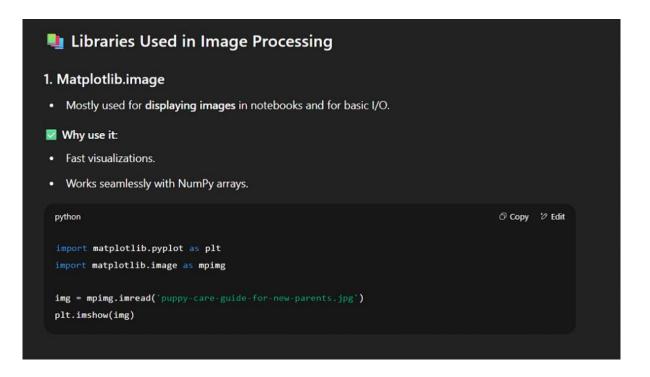
Topics:

- Reading an image file and converting it to a numpy array
- Resizing an image
- RGB to Grayscale conversion

What is Image Processing? Image Processing is the technique of performing operations on images to extract useful information, enhance image quality, or prepare them for tasks such as classification, recognition, and object detection. ✓ Goals of Image Processing: Enhance image quality (denoise, sharpen, contrast) Extract features (edges, textures, shapes) Convert formats (RGB to grayscale, etc.) Resize or transform for model input Prepare for Al/ML model predictions



2. Resizing an Image ML models expect a fixed-size input (e.g., 224x224). Resizing helps standardize input sizes. python Gopy & Edit img_resized = img.resize((224, 224))



```
2. Pillow (PIL)

• PIL = Python Imaging Library.

• Pillow is its updated fork.

• Can open, resize, rotate, crop, and convert image formats.

✓ Why use it:

• Easy for beginners.

• Lightweight for preprocessing.

• Useful for converting between formats (JPEG, PNG).

python

✓ Copy 🏖 Edit

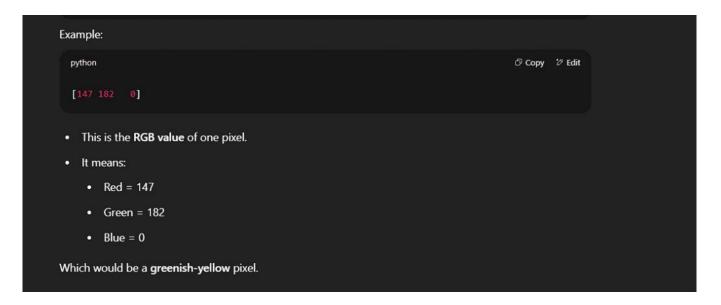
from PIL import Image

img = Image.open("puppy-care-guide-for-new-parents.jpg")
img_gray = img.convert('L')
```



```
python

img = mpimg.imread('dog.jpg')  # load the image
type(img)  # check its type
print(img.shape)  # print its shape (dimensions)
print(img)  # print raw pixel values
```



```
# 1. Display the original image using Matplotlib

python

© Copy  to Edit

img_plot = plt.imshow(img)
plt.show()

img was already loaded using matplotlib.image.imread() earlier.

plt.imshow(img) displays the image in the notebook or Python window.

plt.show() renders it.
```

```
python
python
python
python
potential image using Matplotlib

img_plot = plt.imshow(img)
plt.show()

img was already loaded using matplotlib.image.imread() earlier.

plt.imshow(img) displays the image in the notebook or Python window.

plt.show() renders it.
```

```
python

PIL import Image

img = Image.open('dog.jpg')  # Load image using PIL

img_resized = img.resize((200, 200))  # Resize to 200x200 pixels

img_resized.save('dog_image_resized.jpg')  # Save resized image

Image.open() opens the image with the Python Imaging Library (PIL).

.resize((200, 200)) scales the image to a square 200x200 pixels.

.save() stores the resized image on disk.
```

```
python

Driver Copy

Edit

img_res = mpimg.imread('dog_image_resized.jpg') # Load resized image as numpy array

img_res_plot = plt.imshow(img_res) # Display it

plt.show()

mpimg.imread() reads the resized image back into a NumPy array.

plt.imshow() visualizes the new (smaller) image.
```

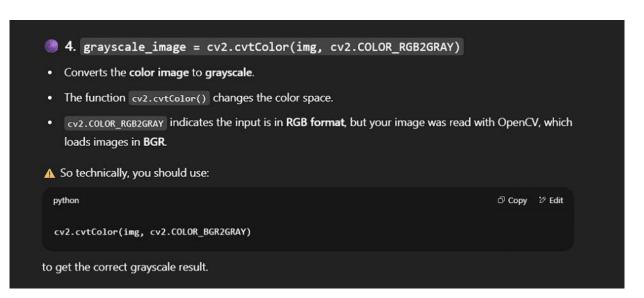
You're now using **OpenCV** (cv2) to:

- 1. Load an image.
- 2. Convert it to grayscale.
- 3. Save the grayscale image.

This is a classic image processing workflow. Let's go through it step by step with theoretical explanations:

```
I. img = cv2.imread('dog.jpg')
This reads the image file (dog.jpg) using OpenCV.
OpenCV loads the image as a NumPy array with shape:
    img.shape = (height, width, 3) → 3 represents the color channels: BGR (not RGB!)
Note: Unlike matplotlib or PIL, OpenCV uses BGR order (Blue, Green, Red) instead of RGB.
```

- 2. type(img) → numpy.ndarray
 Like before, the image is stored as a NumPy array.
 Each pixel is a vector of three values: [B, G, R].
- 3. img.shape example output → (1365, 2048, 3)
 1365 rows (height)
 2048 columns (width)
 3 channels (BGR color)



```
5. grayscale_image.shape → e.g., (1365, 2048)
Only 2 dimensions: height and width.
No color channel dimension, because it's grayscale.
6. cv2.imwrite('dog_grayscale_image.jpg', grayscale_image)
Writes the grayscale image to disk with the name dog_grayscale_image.jpg.
Now it's saved in grayscale format.
```











