Processing image data for Deep Learning

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 AI/ML model predictions

📸 Topics Explained (from the video)

1. Reading an Image File and Converting it to a NumPy Array

- An image is just a matrix (array) of pixel values.
- Reading it into a NumPy array helps perform matrix operations directly.
- For color images:
  - Shape: `(height, width, 3)` for RGB
- For grayscale:
  - Shape: `(height, width)`

python Copy Edit

```python
from PIL import Image
import numpy as np

img = Image.open("puppy-care-guide-for-new-parents.jpg")
img_array = np.array(img)
```

2. Resizing an Image

- ML models expect a fixed-size input (e.g., 224x224).
- Resizing helps standardize input sizes.

python Copy Edit

```python
img_resized = img.resize((224, 224))
```

3. RGB to Grayscale Conversion

- Removes color information, reducing complexity.
- Useful in edge detection, document scanning, etc.

python Copy Edit

```python
img_gray = img.convert("L")  # 'L' mode for grayscale
```

📚 Libraries Used in Image Processing

1. Matplotlib.image

- Mostly used for **displaying images** in notebooks and for basic I/O.

✅ **Why use it:**

- Fast visualizations.
- Works seamlessly with NumPy arrays.

python Copy Edit

```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('puppy-care-guide-for-new-parents.jpg')
plt.imshow(img)
```

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

```python
from PIL import Image

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

3. OpenCV (`cv2`)

- Powerful and fast C++ based library with Python bindings.
- Can do everything: read, write, resize, filter, edge detection, object tracking, etc.

✅ **Why use it:**

- Industrial-grade performance.
- Real-time computer vision applications.
- Broad support for image/video operations and AI integrations.

python Copy Edit

```python
import cv2

img = cv2.imread('puppy-care-guide-for-new-parents.jpg')
img_resized = cv2.resize(img, (224, 224))
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

> ⚠️ Note: OpenCV reads images as **BGR** (not RGB), so you often need to convert it using `cv2.cvtColor`.

python Copy Edit

```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
```

📌 Output Explained:

✅ `type(img)` → `numpy.ndarray`

- The image is stored as a **NumPy array**.
- This means you can manipulate the image like any matrix — useful for image processing.

✅ `img.shape` → `(1365, 2048, 3)`

This tells us:

- `1365` → height (rows) of the image (pixels)
- `2048` → width (columns) of the image (pixels)
- `3` → the number of **color channels** (Red, Green, Blue = RGB)

So it's a **color image** of size **2048×1365 pixels**.

✅ `print(img)` → **the pixel matrix**

You're printing the entire image array. Each pixel is represented as:

```python
[ [R, G, B], ... ]
```

Example:

```python
[147 182   0]
```

- This is the **RGB value** of one pixel.
- It means:
  - Red = 147
  - Green = 182
  - Blue = 0

Which would be a **greenish-yellow** pixel.

📌 1. Display the original image using Matplotlib

```python
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.

📌 1. Display the original image using Matplotlib

python Copy Edit

```python
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.

📌 2. Resize the image using `PIL`

python Copy Edit

```python
from 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.

📌 3. Load and display resized image using `matplotlib`

python Copy Edit

```python
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**:

🖼️ 1. `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)

🟣 4. `grayscale_image = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)`

- Converts the **color image** to **grayscale**.
- The function `cv2.cvtColor()` changes the color space.
- `cv2.COLOR_RGB2GRAY` indicates the input is in **RGB format**, but your image was read with OpenCV, which loads images in **BGR**.

⚠️ So technically, you should use:

python Copy Edit

```python
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

to get the correct grayscale result.

🎨 What is grayscale?

Grayscale means:

- One channel instead of three.
- Each pixel is a single value from 0 to 255.
  - 0 = black
  - 255 = white
  - In between = shades of gray

The formula used:

```ini
Y = 0.299 * R + 0.587 * G + 0.114 * B
```

This preserves brightness and contrast in the grayscale image.

📏 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.

👁️ 7. Displaying in Google Colab:

```python
from google.colab.patches import cv2_imshow
cv2_imshow(grayscale_image)
```

- `cv2.imshow()` doesn't work in Colab, so `cv2_imshow()` is used instead.
- It displays the image inline in Colab.