

Image Processing and Computer Vision

Image Representation, Colour Models & Compression

Sanath Thilakarathna

BSc Eng (Hons), MSc (Reading)

CINEC Campus

February 5, 2026



Digital Images and Basic Properties

What is a Digital Image?

Definition

A digital image is a numerical representation of a visual scene, stored as a two-dimensional array of pixel values.

- Each pixel represents intensity or colour information
- Images are stored as matrices in memory
- Pixel values depend on bit depth and data type

Important

An image is not just a picture; it is structured numerical data.

Basic Image Properties

- Spatial resolution (width $\times$ height)
- Number of channels (grayscale or colour)
- Bit depth (8-bit, 16-bit, etc.)
- Data type (uint8, uint16, float)

Example

A 1920×1080 RGB image with 8-bit depth uses:

$$1920 \times 1080 \times 3 \times 8 \text{ bits}$$

Image Formats

- BMP – uncompressed, large size
- PNG – lossless compression
- JPEG – lossy compression
- TIFF – high-quality storage
- WebP – modern web-optimised format

Note

The same image stored in different formats can have very different file sizes and quality.

Accessing Image Properties in Python

Loading and Inspecting Images

- Use OpenCV (cv2), PIL/Pillow, or scikit-image
- Access shape, dtype, and metadata easily

Example

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('image.jpg')
5 print(img.shape)      # (height, width, channels)
6 print(img.dtype)      # data type
7 print(img.size)       # total pixels
```

Working with PIL/Pillow

- More intuitive for beginners
- Good support for various image formats

Example

```
1 from PIL import Image
2
3 img = Image.open('image.png')
4 print(img.size)           # (width, height)
5 print(img.mode)           # 'RGB', 'RGBA', 'L'
6 print(img.format)         # file format
```

Jupyter Notebook Workflow

- Interactive environment for exploration
- Direct visualization of images
- Quick iteration and testing

Example

```
1 import matplotlib.pyplot as plt
2
3 plt.imshow(img)
4 plt.title(f'Shape: {img.shape}')
5 plt.show()
```

Image Input and Output Operations

Reading and Displaying Images (Python)

Example

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 img = cv2.imread('image.jpg')
5 img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
6
7 plt.imshow(img_rgb)
8 plt.axis('off')
9 plt.show()
```

Note

OpenCV reads images in BGR format by default.

Saving Images

Example

```
1 cv2.imwrite('output.png', img)
2 cv2.imwrite('compressed.jpg', img,
3             [cv2.IMWRITE_JPEG_QUALITY, 50])
```

Important

Saving format and quality parameters directly affect image size and visual quality.

Resolution and Pixel Characteristics

- **Spatial resolution:** Number of pixels (width $\times$ height); determines level of detail
- **Intensity resolution:** Bit depth per pixel; determines number of distinguishable tones/colours

Important

Higher resolution increases memory usage and processing time.

Colour Spaces

Why Do We Need Different Colour Spaces?

- **Task Optimization:** Different tasks require different representations (detection, segmentation, compression)
- **Perceptual Relevance:** Some spaces align better with human colour perception than others
- **Computational Efficiency:** Certain spaces reduce data redundancy and improve processing speed
- **Hardware Compatibility:** RGB for displays and cameras; YCbCr for video compression
- **Feature Extraction:** Separating colour from intensity (HSV) simplifies object detection

Important

Choosing the right colour space can significantly improve algorithm performance and reduce computational overhead.

Colour Representation

- RGB – acquisition and display
- HSV – segmentation and detection
- HSI – perceptual analysis
- HSL – image editing

Note

For more detailed information on colour spaces, see <https://opencv.org/blog/color-spaces-in-opencv/>

RGB Colour Space

- **Red, Green, Blue** primary colours
- Additive colour model (light-based)
- Hardware-native format for displays and cameras
- Range: 0-255 per channel (8-bit)

Example

```
1 import cv2
2
3 img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4 r, g, b = cv2.split(img_rgb)
```

Note

RGB is intuitive for acquisition but poor for segmentation tasks.

HSV Colour Space

- **Hue:** Colour type (0-180 in OpenCV)
- **Saturation:** Colour intensity (0-255)
- **Value:** Brightness (0-255)
- Intuitive for colour-based object detection

Example

```
1 img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
2 h, s, v = cv2.split(img_hsv)
```

Important

HSV separates colour information from lighting, ideal for segmentation.

- **Hue:** Dominant wavelength
- **Saturation:** Purity of colour
- **Intensity:** Overall brightness
- More perceptually uniform than HSV

Note

HSI is closer to human colour perception but less commonly used in OpenCV.

HSL Colour Space

- **Hue:** Colour type
- **Saturation:** Colour vividness
- **Lightness:** Brightness relative to white/black
- Symmetric distribution compared to HSV

Example

PIL/Pillow supports HSL conversion for image editing workflows.

Important

HSL is preferred in design and editing applications over HSV.

Colour Space Conversion (Python)

Example

```
1 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
2 h, s, v = cv2.split(hsv)
```

Note

HSV separates colour information from illumination.

Image Histograms

Image Histogram

Definition

An image histogram shows the distribution of pixel intensity values.

Example

```
1 plt.hist(img_rgb[:, :, 0].ravel(), bins=256)
2 plt.show()
```

When to Use Histogram Equalization

- Low-contrast images
- Images with poor illumination
- Enhancing features for analysis

Important

Use histogram equalization judiciously to avoid amplifying noise.

What is Histogram Equalization?

Definition

Histogram equalization is a technique in image processing used to enhance the contrast of an image by redistributing the intensity values of the pixels.

- It transforms the intensity distribution to be more uniform.
- This process improves the visibility of features in poorly illuminated or low-contrast images.

Why is Histogram Equalization Required?

- **Improving Contrast:** Enhances the contrast, making it easier to distinguish between different features.
- **Highlighting Details:** Reveals hidden details in images that may not be visible due to poor lighting conditions.
- **Standardization:** Standardizes brightness levels across different images, making them more comparable.

Important

Use histogram equalization judiciously to avoid amplifying noise in low-quality images.

Histogram Equalisation

Example

```
1 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
2 equalised = cv2.equalizeHist(gray)
```

Warning

Histogram equalisation may amplify noise in low-quality images.

Image Compression

Image Compression Fundamentals

- Reduces storage and bandwidth
- Removes redundancy and perceptual irrelevance

Lossless vs Lossy Compression

Example

Lossless:

- PNG, TIFF
- Exact reconstruction

Example

Lossy:

- JPEG, WebP
- Higher compression ratios

Visual Comparison of Compression

Example

```
1 cv2.imwrite('q90.jpg', img,  
2             [cv2.IMWRITE_JPEG_QUALITY, 90])  
3 cv2.imwrite('q30.jpg', img,  
4             [cv2.IMWRITE_JPEG_QUALITY, 30])
```

Important

Compression is always a trade-off between quality and efficiency.

Resources and Example Notebooks

- Access example Jupyter notebooks used in class
- Hands-on code demonstrations and exercises
- Repository: https://github.com/sanath-thilakarathna/BSc_B5_EE4216

Important

Clone the repository and explore the notebooks to practice the concepts covered in this lecture.