Computer Vision - Jaish Khan

1. Image Processing

> Manipulation of images using mathematical operations and algorithms to enhance, analyze, or extract information from them

- *Focuses* on the "Transformation of images at the pixel level".
- *Aims* to improve the visual appearance of images, remove noise, enhance features, and perform tasks like image compression or restoration.
- *Used* for Medical imaging (e.g., MRI or CT scans), Satellite imaging, Photography, Digital art, and more.

Image is a collection of pixels.
Pixel is the smallest unit (and contains a color value).
Video is a series of frames (images).

Images can be **single-channel** (Black and White) or **three-channel** (Red, Green, Blue).

1.1. Image Processing Techniques

not important

1. **Image Filtering/Convolution** → Use a filter on an image to get effects like blurring.

2. **Image Transformation** → Change image geometry like resizing or rotating.
3. **Color Manipulation** → Enhance or Analyze colors.
4. **Thresholding** → Simplify image to binary using a threshold value. Pixels above the threshold are set to white, while pixels below are set to black.
5. **Morphological Operations** → Filter binary images based on shapes.
6. **Histogram Equalization** → Improve image contrast.
7. **Feature Detection/Extraction** → Detecting key regions in an image for matching and tracking.
8. **Image Blending** → Combine images by blending their pixel values.
9. **Geometric Transformation** → Distort an image for perspective correction and object alignment.
10. **Image Segmentation** → Divide an image into meaningful segments based on color, intensity, or texture.
11. **Noise Reduction** → Remove/Reduce noise from an image.
12. **Image Registration** → Aligning multiple images for comparison or fusion.

2. Computer Vision

> Enabling computers to interpret and understand visual information from the real world.

- *Aims* to replicate human vision by extracting high-level information from images or video. This involves tasks like object detection, recognition, tracking, scene understanding, and 3D reconstruction.
- *Used* for autonomous vehicles, surveillance systems, augmented reality, robotics, medical diagnostics, and industrial quality control.

2.1. Image Processing vs Computer Vision

Image Processing	Computer Vision
Enhancing or modifying images.	Extract and understand meaningful information from images.
Operates at the pixel level, manipulating individual pixels or small groups of pixels.	Operates at a higher level of abstraction, dealing with objects, shapes, and scenes.
Tasks include noise reduction, sharpening, edge detection, and image segmentation.	Tasks include object detection, recognition, tracking, and scene understanding.

2.2. IDE Installation

Integrated Development Environment (IDE)

- **Anaconda** → A platform that provides multiple services for Data Science and Machine Learning including IDEs and Packages.
    1. Open browser and search "Download Anaconda python".
    2. Download the .exe file and Install it.
- **Spyder** → An IDE for scientific computing in Python.
- **OpenCV** → Install by typing `pip install opencv-python` in the terminal.

2.3. OpenCV

> A free and open-source image processing library, available on Windows/Linux/MacOS and works on C, C++ and Python.
> Made by Intel, Supported by Willow Garage, Maintained by Itseez

We use a 2D *array* to store an image. `Numpy` is used to handle the *array*.

2.3.1. OpenCV Functions

Function	Description
`cv2.imread(filepath, flags)`	Reads an image from a file.
`cv2.imwrite(filepath, image)`	Saves an image to a file.
`cv2.imshow(windowname, image)`	Displays an image in a window.
`cv2.waitKey(delay)`	Waits for a key event.
`cv2.destroyAllWindows()`	Closes all OpenCV windows.
`cv2.resize(image, (width, height))`	Resizes the image window.

1. The second parameter (flags) of *imread* function can take many values:
    - `cv2.IMREAD_COLOR` (1) → Loads a color image (default). Transparency is ignored.
    - `cv2.IMREAD_GRAYSCALE` (0) → Loads image in grayscale mode.
    - `cv2.IMREAD_UNCHANGED` (-1) → Loads image as-is (including alpha channel).
2. The *waitKey* function can take number values in milliseconds:
    - `waitKey(3000)` → Waits for 3secs.
    - `waitKey(0)` → Closes immediately.

2.3.2. Reading an Image

```
import cv2
img = cv2.imread("Image-Path")
cv2.imshow("Window-Name", img)
cv2.waitKey()
cv2.destroyAllWindows()
```

2.3.3. Resizing an Image

```
# This line (before imshow) to resize the image window.
img = cv2.resize(img,(640,350))
```

2.3.4. Printing the Image

```
# This line prints the pixel values of the image on the console.
print(img)
```

2.3.5. Binding Mouse Events

Mouse Event	Description
`cv2.EVENT_LBUTTONDOWN`	Left mouse button pressed
`cv2.EVENT_LBUTTONUP`	Left mouse button released
`cv2.EVENT_RBUTTONDOWN`	Right mouse button pressed
`cv2.EVENT_MOUSEMOVE`	Mouse moved
`cv2.EVENT_LBUTTONDBLCLK`	Double click with left button

```
import cv2

def callback(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(f"Left Click at ({x}, {y})")

image = cv2.imread(filepath)
cv2.namedWindow("Mouse Event Window")
cv2.setMouseCallback("Mouse Event Window", callback)

while True: if cv2.waitKey(1) & 0xFF == 27: break
cv2.destroyAllWindows()
```

2.3.6. Making a Color Picker

```
import cv2
import numpy as np

# Load the image
image = np.zeros((300, 400, 3), np.uint8)
cv2.namedWindow('Color Picker')
cv2.createTrackbar('R', 'Color Picker', 0, 255, 0)
cv2.createTrackbar('G', 'Color Picker', 0, 255, 0)
```

```
cv2.createTrackbar('B', 'Color Picker', 0, 255, 0)

while True:
        cv2.imshow('Color Picker', image)
        if cv2.waitKey(1) & 0xFF == 27: break
        r = getTrackbarPos('R', 'Color Picker')
        g = getTrackbarPos('G', 'Color Picker')
        b = getTrackbarPos('B', 'Color Picker')
        img[:] = [b, g, r]

cv2.destroyAllWindows()
```

2.4. MATLAB

MATLAB (MATrix LABoratory) is a high-level programming language and interactive environment made by **MathWorks**, used for:

1. **Reading a Video** → The *VideoReader* function creates a video object.
   `video = VideoReader('your_video.mp4');`
2. **Read Frames, sequentially** → The *readFrame* function reads the current frame.
   `frame = readFrame(video);`
3. **Read Specific Frames** → Instead of reading frames one-by-one, we can directly specify the frame with the *read* function. `n` is the frame number that we want to extract.
   `frame = read(video, n);`

> ⓘ **Semicolon after `VideoReader()` and `readFrame()`**
>
> 1. We can view the Image Properties like duration, current time and number of frames by not using a semicolon after `VideoReader()`.
> 2. We can view the Matrix of all values of the Frame by not using a semicolon after `readFrame()`.

We can also type the variable name followed by a dot and use the arrow keys to see these properties like `video.FrameRate`.

4. **Navigating through the Video** → *VideoReader* keeps track of your position in the video using the `CurrentTime` property which is updated each time we use the *imread* function.
   - *Loop through the video* (using a while loop) and `hasFrame(video)` checks if more frames are available.
   - *Reset the current time* (to the beginning of the video)
     `turtleVideo.CurrentTime = 0`.

- *Skip to the last frame* (by specifying `Inf` as the second input)
  `frame = read(video,Inf);` .

5. **Displaying Frames** → Once we have a frame, it is treated as a regular image and can be displayed using the `imshow` function.
   `imshow(frame);`

> ⓘ **Overlay two images**
>
> We can overlay two images and show their differences using the `imshowpair` function.

`pause()` controls playback speed to match original frame rate.

```
video = VideoReader('your_video.mp4');

% Loop through each frame
while hasFrame(video)
    frame = readFrame(video);        % Read the current frame
    imshow(frame);                   % Display the frame
    pause(1/video.FrameRate);        % Pause to match the video frame rate
end
```

2.4.1. Cropping an Image

The *imcrop* function allows you to crop an image to a rectangle you want to keep.

```
cropped = imcrop(im,rect)
```

The `rect` parameter is a 4-element vector → [xmin, ymin, width, height].

- `rect = [0, 170, 250, 100]` starts the crop **0 pixels from the left** and **170 pixels from the top**, with a **width of 250** and **height of 100**.

```
image = imread('turtle.jpg');  % Read the original image
rect = [0, 170, 250, 100];  % Define the cropping rectangle
turtle = imcrop(image, rect); % Crop the image
imshow('Cropped Turtle', turtle); % Display the cropped image
```

2.5. Object Detection

For object detection, you need a reference image for the object you want to detect. A reference image should contain only features that are useful for detecting the object. There are 5 steps (phases) of doing this in MATLAB

1. **Import Video data** and then **Read individual frames** as images.
2. **Feature Extraction and Labeling** in images. This involves
    1. Identifying distinctive regions in an image (detection) → `detectSIFTFeatures`.
    2. Describing these features numerically (extraction) → `extractFeatures`.
    3. Finding similar features between different frames (matching) → `matchFeatures`.
    - *Label objects in a video and generate ground truth images for object detection*.
3. **Object Detection Method Selection** → Choose a detection method.
    - We can train an *aggregated channel features (ACF) object detector* to find turtles in images and in a video. Using the labeled ground truth data to train a ML model to recognize the objects in new, unseen images and video frames.
4. **Detection Implementation** → Apply the chosen detection method.
5. **Visualization and Evaluation** → See and test the results.

SIFT → Scale invariant feature transform

```
points1 = detectSIFTFeatures(frame1);
[features1, featurePoints1] = extractFeatures(frame1,points1);
points2 = detectSIFTFeatures(frame2);
[features2, featurePoints2] = extractFeatures(frame2,points2);

pairs = matchFeatures(features1,features2)

idx1 = pairs(:,1)
matchedPoints1 = featurePoints1(idx1)
idx2 = pairs(:,2)
matchedPoints2 = featurePoints2(idx2)

showMatchedFeatures(frame1,frame2, matchedPoints1,matchedPoints2,
"montage")
```