Computer Vision (Finals) - Jaish Khan

1. Important Topics

1.1. Image Fundamentals

> ⓘ **Image Terms**
>
> - **Image** → A digital picture or graphic made up of a grid of tiny individual colored squares or dots called pixels.
>   - **Single-channel Image** → An image that uses only one layer or "channel" of information per pixel, usually representing brightness or grayscale (like a black and white photo).
>   - **Three-channel Image** → A color image that uses three separate layers or "channels" of information per pixel, typically for the Red, Green, and Blue color components.
> - **Pixel** → The smallest individual point or unit in a digital image. Each pixel holds a single color or brightness value.
> - **Video** → A series of still images, called frames, shown one after another very quickly to create the illusion of motion.

- **Digital Image** → Composed of pixels, discrete in nature, stored in binary format.
- **Analog Image** → Continuous image signal (e.g., film-based photography).
- **Spatial Resolution** → Pixel density of an image (detail level).
- **Luminance Resolution** → Number of possible brightness levels.
- **Image Sampling** → Reduces spatial resolution by selecting certain pixels.
- **Image Quantization** → Reduces luminance resolution by mapping pixel values to a finite range.
- **Convolution** → Involves flipping the kernel before sliding it over the image.
- **Correlation** → Direct application of the kernel without flipping.
- **Image Acquisition and Role of Sensors**
  1. Involves capturing an image using a sensor array (CCD or CMOS).
  2. Sensors convert electromagnetic signals (light) into electrical signals.
  3. The analog signals are then digitized using an ADC (Analog to Digital Converter).
  4. The final digital image is composed of discrete pixels with gray levels or color values.

Pinhole Camera Model

Simple camera model where rays from a 3D scene converge through a pinhole to form a 2D image.

- *Surveillance*: Maps 3D environment to a 2D plane.
- *3D Vision*: Helps in triangulating depth using multiple views.

Feature	RGB	CMYK	LAB	HSL
Used in	Screens (TVs, monitors, cameras)	Printers	Image processing, photo editing	Color pickers, design tools
Type	Additive (light-based)	Subtractive (ink-based)	Perceptual (based on human vision)	Perceptual (user-friendly for designers)
Color Components	Red, Green, Blue	Cyan, Magenta, Yellow, Black	L* (lightness), a* (green–red), b* (blue–yellow)	Hue (0–360°), Saturation (0–100%), Lightness (0–100%)
Value Range	0–255 per channel	0–100% per ink	L*: 0–100, a*/b*: ~ -128 to 127	Hue: 0–360°, Sat/Light: 0–100%
Color Basis	Light emission	Ink absorption	Human vision	Color tone, intensity, brightness
Example Use	Web, digital images	Print media	Color correction, analysis	UI/UX design, CSS, Figma

- **Image Enhancement** → Focuses on making an image look *better* or more useful for a specific purpose (like making details clearer for a human viewer or later computer analysis). It doesn't try to reverse specific damage, just improve appearance or highlight features.
- **Image Restoration** → Aims to *fix* a degraded image by trying to reverse the cause of the degradation (like blur or noise) to get closer to the original, un-damaged image. It requires some knowledge about how the image was degraded.
- **Preprocessing Tasks** → These are initial operations applied to an image to prepare it for further analysis or processing. They help improve image quality, remove unwanted elements, or standardize the image format or appearance.

    - **Noise Removal:** Reducing unwanted random variations (noise) in the image data that make it look grainy or unclear.
        - *Gaussian Smoothing Filter*: reduces random noise while preserving edges.
        - *Median Filter*: effective for salt-and-pepper noise.
    - **Image Resizing:** Changing the dimensions (width and height) of the image.
    - **Normalization:** Adjusting the pixel values to a standard range or distribution to ensure consistency, often helpful when comparing different images.
- **Histogram Equalization** → Redistributes pixel intensity values for a uniform histogram. Enhances contrast by spreading out intensity values, especially in low-contrast images.

1.3. Feature Extraction

> The process of identifying and isolating specific patterns, points, or structures within an image that are useful for analysis, recognition, or matching.

- **Thresholding** → Converts a gray-scale image into a binary image by turning all pixels above a certain value (threshold) to white and below to black, helping in segmentation.
- **Edge Detection** → Process of identifying sharp changes in intensities in an image, representing object boundaries. Example algorithms are Sobel and Canny.
- **Sobel's Edge Detection Algorithm**
    1. Input: Grayscale image
    2. Define Sobel kernels Gx and Gy
    3. For each pixel in image:
        - Apply Gx and Gy to compute gradients
        - Compute gradient magnitude = $\sqrt{Gx^2 + Gy^2}$
    4. Output: Edge map
- **Segmentation Steps** → Preprocessing → Edge detection → Thresholding → Region growing → Post-processing.
- **Segmentation Applications** → Medical imaging (tumor detection) and Object detection in autonomous vehicles.
- **K-Means for Image Segmentation**
    1. Choose K (number of clusters).
    2. Initialize K centroids randomly.
    3. Assign each pixel to the nearest centroid based on intensity or RGB value.
    4. Update centroids based on current clusters.
    5. Repeat until convergence.
    6. Output: Each pixel labeled with a cluster, forming distinct segments.

- **SIFT** (Scale-Invariant Feature Transform) → Detects key points and creates descriptors invariant to scale, rotation. Useful in image matching.
- **HOG** (Histogram of Oriented Gradients) → Describes object shape by gradient orientation distribution. Used in pedestrian detection.
- **OCR** (Optical Character Recognition) → Converts images of text (hand-written or printed) into machine encoded text.

1.4. Machine Learning

> **Supervised Learning** → Learns from labeled data. Examples: Linear Regression(regression), Decision Trees(classification), CNN etc.
> **Unsupervised Learning** → Learns from unlabeled data. Examples: K-Means(clustering), PCA(dimensionality reduction), Autoencoders etc.

> ⚠ **Convolutional Neural Networks**
>
> CNN is a deep learning model designed to process and recognize patterns in images through convolutional layers and pooling layers.

Backpropagation in Neural Networks → Adjusts weights to reduce output error.

1. Forward pass: Calculate output.
2. Compute error (difference from target).
3. Backward pass: Distribute error to previous layers.
4. Update weights using gradient descent. Repeats over epochs to improve classification accuracy.

---

2. Other Topics

What it is Comparing	Image Processing (IP)	Computer Vision (CV)
What it Does	Changes images using computer methods to improve them or prepare them for use.	Teaches computers to 'see' and understand the content of images and videos.
Main Goal	To make images look better, remove flaws, highlight details, or change their format.	To figure out what an image or video shows, like finding objects, recognizing them, or understanding the scene.

How it Works	Works directly on the individual tiny dots (pixels) of the image or small pixel groups.	Looks at the bigger picture to understand objects, shapes, and the whole scene.
Examples	Making blurry images clearer, removing noise (grain), finding simple outlines (edges), splitting an image into parts.	Finding specific objects, recognizing faces, tracking moving things, understanding activities in a video.
Where it's Used	In medical scans (like X-rays or MRIs), making photos look better, satellite imagery, digital art.	In self-driving cars, security cameras, face recognition, robots, augmented reality apps, quality checks in factories.

2.1. Technologies used

- **Anaconda** → A platform that provides multiple services for Data Science and Machine Learning including IDEs and Packages.
- **Spyder** → An IDE for scientific computing in Python.

2.1.1. OpenCV

A free and open-source image processing library, available on all platforms and works on C, C++ and Python. We use a 2D *array* to store an image. `Numpy` is used to handle the *array*.

2.1.2. MATLAB

MATLAB (MATrix LABoratory), made by **MathWorks**, is a powerful high-level programming language and interactive environment widely used for numerical computation, data analysis, and algorithm development, including applications in image and video processing.

1. **Reading Video Files** → Use the VideoReader function to create an object that links to a video file. Omitting the semicolon `;` after `VideoReader()` or `readFrame()` will display the properties of the video object or the pixel matrix of the frame in the command window. You can also access specific properties directly, e.g., `video.FrameRate`.
2. **Reading Video Frames** → You can read frames sequentially or access specific frames:
   - *Sequentially*: Use the `readFrame` function to read the next available frame.
   - *Specific Frame*: Use the `read` function with the desired frame number `n`.
3. **Navigating and Looping** → The VideoReader object tracks the current position using the CurrentTime property.

- You can loop through all frames using a `while hasFrame(video)` loop. `hasFrame(video)` returns true if there are more frames to read.
- To reset the video position to the beginning, set `video.CurrentTime = 0`.
- To read the last frame, use `frame = read(video, Inf)`.

4. **Displaying Frames** → Once a frame is read, it's treated as a regular image (a matrix) and can be displayed using imshow.
   - Use `imshowpair` to display two images side-by-side or overlaid, useful for comparing frames or highlighting differences.
   - Use `pause(seconds)` to control display speed, often `pause(1/video.FrameRate)` to match the original video speed when looping through frames.

2.2. Object Detection

A computer vision task where the goal is for a computer to find and locate specific items (like cars, people, or animals) within an image or video frame. It draws a box around each detected object and tells you what the object is. Steps of Object Detection:

1. **Input and Preparation** → Get the image data you want to analyze. If it's a video, this involves reading individual frames sequentially. You would use functions like `VideoReader` to handle video files and `readFrame` to get each image frame.
2. **Processing and Analysis** → The computer analyzes the image to find areas that likely contain objects. This is the core of the detection method. Different object detection **methods** use different techniques for this analysis. Some look for specific image **features** (patterns or textures), while others use complex **models** trained on large datasets to recognize objects.
3. **Outputting Results** → The detector provides the locations (usually as bounding boxes) and the type or **class** (e.g., 'person', 'car') for each object it found in the image.
4. **Review and Use** → You can then visualize these results by drawing the boxes and labels on the image. The detection results can be used for further tasks like counting objects, tracking their movement, or triggering actions based on what is seen.

---

3. Programming

OpenCV Function	Description
`cv2.imread(filepath, flags)`	Reads an image from a file.
`cv2.imwrite(filepath, image)`	Saves an image to a file.

OpenCV Function	Description
`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.

3.1. Resizing and Region of Interest

```
import cv2

img = cv2.imread("F:\design.png")
img = cv2.resize(img, (300,460)) #Resizing
roi = img[36:117, 107:194] #Region of Interest

cv2.imshow("img", roi)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.2. Binding Mouse Events

```
import numpy as np
import cv2

def draw(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(f"Left Click at ({x}, {y})")
    if event == cv2.EVENT_LBUTTONDBLCLK:
        cv2.circle(img,(x,y),100,(125,0,255),5)
    if event == cv2.EVENT_RBUTTONDBLCLK:
        cv2.rectangle(img,(x,y),(x+100,y+75),(125,125,255),2)

cv2.namedWindow(winname = "res")
img = np.zeros((512,512,3), np.uint8)
cv2.setMouseCallback("res",draw)
```

```
while True:
    cv2.imshow("res",img)
    if cv2.waitKey(1) & 0xFF == 27: break

cv2.destroyAllWindows()
```

3.3. Color Picker

```
import cv2
import numpy as np
def none(x): pass

img = np.zeros((300,512,3),np.uint8)
cv2.namedWindow("Color Picker")

switch = '0 : OFF \n1 : ON'
cv2.createTrackbar(switch, 'Color Picker',0,1,none)
cv2.createTrackbar("R","Color Picker",0,255,none)
cv2.createTrackbar("G","Color Picker",0,255,none)
cv2.createTrackbar("B","Color Picker",0,255,none)

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

cv2.destroyAllWindows()
```

3.4. Drawing Shapes

```
import numpy as np
import cv2

img = np.ones([512, 512, 3], np.uint8) * 255

img = cv2.line(img, (0, 0), (200, 200), (245, 7, 31), 10)
img = cv2.arrowedLine(img, (0, 125), (255, 255), (255, 0, 125), 10)
img = cv2.rectangle(img, (384, 10), (510, 128), (128, 0, 255), 5)
img = cv2.circle(img, (247, 125), 63, (214, 255, 0), 5)
```

```
img = cv2.ellipse(img, (400, 400), (100, 50), 0, 0, 360, (0, 360, 125),
-1)

font = cv2.FONT_ITALIC
img = cv2.putText(img, 'Jaish', (20, 500), font, 4, (0, 125, 255), 10,
cv2.LINE_AA)

cv2.imshow("res", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.5. Reading and Playing a Video

```
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
```

3.6. Cropping Images

```
% Define the cropping rectangle with top-left at (0, 170), size 250x100
rect = [0, 170, 250, 100];
image = imread('turtle.jpg');

turtle_cropped = imcrop(image, rect); % Crop the image
imshow(turtle_cropped); % Display the cropped image
title('Cropped Turtle'); % Add a title to the displayed image
```

---

4. Numericals

4.1. Depth-Disparity Problems

FORMULA

$$\text{Depth} = \frac{\text{Baseline} \times \text{Focal Length}}{\text{Disparity}}$$

Question 1. A stereo camera system has a baseline of 0.1 meters and a focal length of 800 pixels. If the disparity for a particular point is 20 pixels, what is the depth of that point?

Data: `Baseline = 0.1m`, `Focal Length = 800p`, `Disparity = 20p`, `Depth = ?`
Solution: (put values in the formula)

$$\text{Depth} = \frac{0.1 \times 800}{20}$$

$$\boxed{\text{Depth} = 4 \text{ meters}}$$

Question 2. If the depth of an object is 4 meters, the baseline is 0.05 meters and focal length is 1000 pixels, what is the disparity?
Data: `Baseline = 0.05m`, `Focal Length = 1000p`, `Disparity = ?`, `Depth = 4m`
Solution: (put values in the formula)

$$4 = \frac{0.05 \times 1000}{\text{Disparity}}$$

$$\text{Disparity} = \frac{0.05 \times 1000}{4}$$

$$\boxed{\text{Disparity} = 12.5 \text{ pixels}}$$

Question 3. Given a focal length of 700 pixels and depth of 3 meters, what baseline is required if disparity is 35 pixels?
Data: `Baseline = ?`, `Focal Length = 700p`, `Disparity = 35p`, `Depth = 3m`
Solution: (put values in the formula)

$$3 = \frac{\text{Baseline} \times 700}{35}$$

$$\text{Baseline} = \frac{3 \times 35}{700}$$

$$\boxed{\text{Baseline} = 0.15 \text{ meters}}$$

4.2. Image Storage Calculations

Question. Calculate total bits in a 256 × 256 pixels 8-bit image.
Solution:

1. Calculate the total number of pixels:
   `256 pixels (width) * 256 pixels (height) = 65536 pixels`
2. Calculate the total number of bits:
   `65536 pixels * 8 bits/pixel = 524288 bits`
3. Convert bits to bytes (since 1 byte = 8 bits):
   `524288 bits / 8 bits/byte = 65536 bytes`
4. Convert bytes to Kilobytes (since 1 KB = 1024 bytes):
   `65536 bytes / 1024 bytes/KB = 64 KB`

- Total size: **64 KB** (Kilobytes)`