Website: IvLabs
Follow Us: LinkedIn,Instagram,Youtube

Technoseason 2025

General Instructions:

- This task consists of two topics: Python and OpenCV
- The points you'll get will be judged based on your logic, approach, implementation, etc.
- Partial Completed task is also acceptable. The Points will be given based on the extent of completion
- While you may take help from the internet for the errors you are facing or understand any algorithm, directly copying solutions is strictly prohibited.
- Some tasks have Bonus Points. This will be awarded if someone finds a unique way to solve the problem or do something out of the box.
- You must submit all your code files as a Zip File or as a Colab file preferably, also, a PDF file that includes how your code works, and Pictures/Videos of your Output.
- After completion of tasks, create a ZIP folder with all your files and upload them on this Google form Technoseason Submission
- The last submission date is 25 February 2025 before 11:59 pm. The final decision lies with the event coordinators and is not subject to change.

Python

Instructions:

- This part consists of 5 questions along with one Bonus part for some question
- You can submit either .ipynb files (Jupyter Notebooks, Google Collab file not the link) or python .py files (in a ZIP folder). You are free to use multiple files for a single problem.
- The questions are categorized by difficulty levels, ranging from easy to hard, with points assigned accordingly.
- You're not supposed to hardcode the inputs. The inputs should be user-driven
- A Well Commented Code is expected so everyone can understand it properly.

1. Guess the Date

Difficulty Level: Easy

Points: 10

Problem Statement:
In a world where time seems to flow like a river, you are a time traveller who can travel both forward and backwards in time. Your task is to make a magic calendar that determines what day of the week it will be on any date in the future or past. But there's a catch: You don't know today's date explicitly! (Do not input today's date and day).

Example:

On July 20, 1969, Apollo 11 astronauts landed on the Moon when Neil Armstrong became the first human to walk on its surface. It was Sunday on that day, so your magic calendar should return Sunday once given the date. This is an example, it should work for all dates.

2. Maximum Happiness Allocation

Difficulty Level: Medium

Points: 15

Problem Statement:
You are organizing a charity event and have K volunteers and N tasks to assign. Each task i has a difficulty level D[i] and a happiness value H[i] if completed. Each volunteer has a skill level S[j]. You can assign a volunteer to a task if their skill level is greater than or equal to the task's difficulty. Each volunteer can complete only one task, and each task can be assigned to only one volunteer. Your goal is to maximize the total happiness.

Inputs:

- N: Number of tasks ($1 \leq N \leq 100$)
- D[i]: Difficulty of the i-th task ($1 \leq D[i] \leq 100$)
- H[i]: Happiness value of the i-th task ($1 \leq H[i] \leq 1000$)

- K: Number of volunteers ($1 \leq K \leq 100$)
- S[j]: Skill level of the j-th volunteer ($1 \leq S[j] \leq 100$)

Example:

Input:
N = 5
D = [3, 8, 2, 5, 6]
H = [10, 50, 20, 40, 30]
K = 5
S = [2, 6, 8, 3, 7]

Output:
140

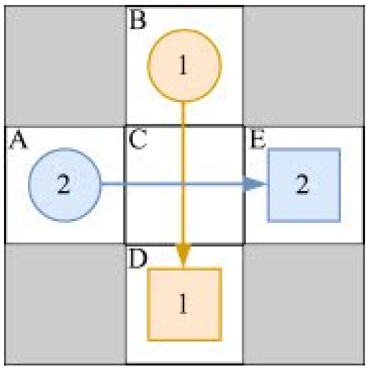
3. Multi-Agent Pathfinding with Collaborative Avoidance

Difficulty Level: Medium

Points: 10

Bonus: 05

Problem Statement:
Design a simulation for Multi-Agent Pathfinding with Collaborative Avoidance in a grid environment. The grid has obstacles, and multiple agents must navigate towards their respective goals without colliding with each other or the obstacles. The agents should be able to collaborate to avoid deadlocks and optimize their movements.



Game Components:

1. Grid: The game will be played on a grid of size N x N, where N is provided by the user. Some cells are blocked with obstacles, and the agents must avoid these during their movement.
2. Agents: There will be M agents, each with a starting position and a goal. The agents must move from their start positions to the goal positions while avoiding obstacles and other agents.
3. Obstacles: The grid will contain obstacles, which agents must navigate around. These obstacles are randomly placed at the start of the game.
4. Movement: Each agent can move one step at a time in one of four directions: up, down, left, or right. Agents cannot move diagonally or through obstacles. They must avoid collisions with other agents during their movement.
5. Collaboration: If two agents encounter a deadlock (i.e., both agents want to move into the same space), they must collaborate to resolve the deadlock by adjusting their movement strategies (e.g., moving in alternate directions).

Gameplay:

- The agents take turns moving, and their movement will be based on a simple pathfinding algorithm such as BFS or A*.
- After each move, the grid will be printed in the terminal to show the updated positions of the agents and obstacles.
- If an agent reaches its goal, it is considered to have successfully navigated the grid.
- The game ends when all agents reach their goals or when it is impossible for the agents to reach their goals due to deadlock or blocked paths.

Objective:

Develop the game mechanics to simulate agent movement, obstacle avoidance, and deadlock resolution. The agents must use a pathfinding algorithm to reach their goals while avoiding other agents and obstacles. Additionally, the agents should collaborate to avoid deadlocks when necessary.

Challenge:

- Implement the pathfinding algorithm that handles both obstacle avoidance and collaboration.
- Implement a method for resolving deadlocks where two or more agents may attempt to occupy the same space.
- After every turn, update the grid and display it in the terminal to show the new positions of the agents.

Bonus:

- Implement the pathfinding algorithm that handles both obstacle avoidance and collaboration.
- Implement the pathfinding algorithm that handles both obstacle avoidance and collaboration.

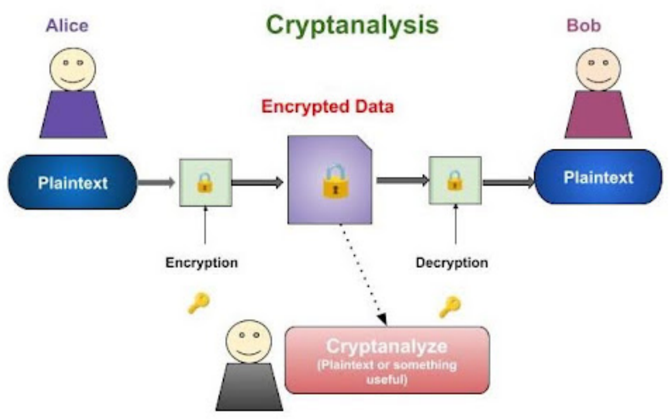
4. KenLogic Grid

Difficulty Level: Hard

Points: 20

Bonus: 10

Problem Statement:
You have intercepted a highly classified communication grid, codenamed KenLogic Grid, used by an underground organization to encrypt secret messages. The grid is composed of binary values (0 and 1) and follows a complex set of logical rules. Your task is to decode this grid by filling in the missing binary values while adhering to several constraints.
The grid is not just a simple puzzle–it's a test of your ability to apply binary logic operations (AND, OR, XOR) to solve it. The encrypted message can only be revealed once the grid is fully completed. But beware, the organization has embedded strict rules that you must follow to ensure the puzzle is solved correctly. Can you break the code and reveal the hidden message by solving the KenLogic Grid? The fate of the encrypted message rests in your hands!



Game Components:

1. Grid: A 6x6 grid where each cell contains a 0 or 1. Some cells are pre-filled, while others are empty (denoted by .), and need to be filled in. The grid is partially filled with 0 and 1 values, and your job is to complete it based on the provided constraints.
2. Cage: The grid is divided into regions known as cages. Each cage has:
   - A target value (either 0 or 1).
   - A binary operation (AND, OR, XOR).
   - A list of cells that belong to the cage (e.g., A1, A2, B1).

Constraints:

To decode the message, you must adhere to the following strict rules:

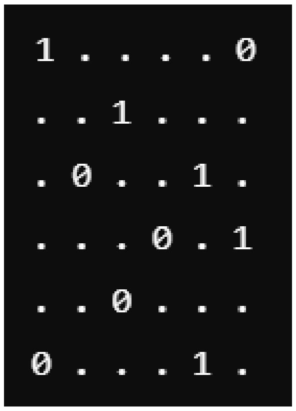
- Row and Column Constraints: Each row and column can contain any number of 1s and 0s.
- Cage Constraints: The cells within each cage must, when combined using the specified binary operation, produce the target value for that cage. The operations are:
  - AND: The result of combining all the cells in the cage using the AND operation.
  - OR: The result of combining all the cells in the cage using the OR operation.

    – XOR: The result of combining all the cells in the cage using the XOR operation.

- Uniqueness Rule: No two rows or columns can be identical.

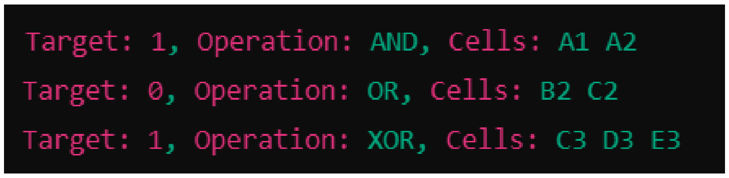
Input:

- **Grid:** A 6x6 grid with some cells pre-filled with 0 or 1 and others left as . (to be filled).
  Example:



```
1 . . . . 0
. . 1 . . .
. 0 . . 1 .
. . . 0 . 1
. . 0 . . .
0 . . . 1 .
```

- **Cages:** Each cage is described by:
    - Target value (0 or 1).
    - Binary operation (AND, OR, XOR).
    - List of cells in the cage (e.g., A1, A2, B1).



```
Target: 1, Operation: AND, Cells: A1 A2
Target: 0, Operation: OR, Cells: B2 C2
Target: 1, Operation: XOR, Cells: C3 D3 E3
```

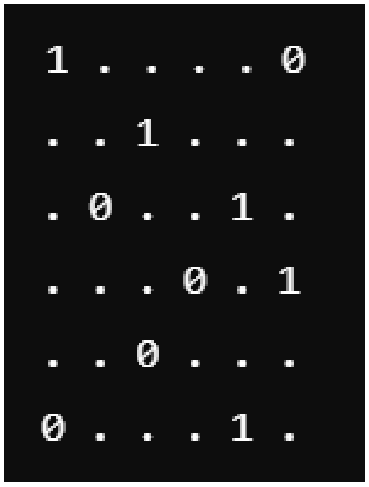
Output Requirements:

- If solving the puzzle:
    - Output the completed grid if a solution exists.
    - Output NO SOLUTION if the puzzle cannot be solved.

- If verifying a custom input
  - Output VALID if the grid satisfies all rules (row/column constraints, diagonal rule, cage constraints).
  - Output INVALID otherwise.

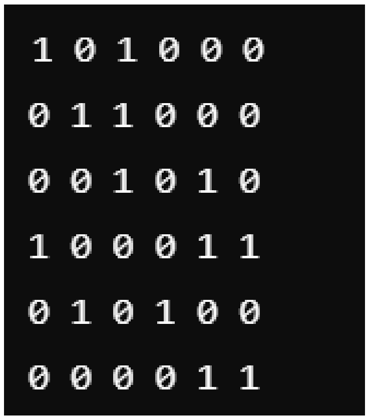
Example:

- Actual Problem Grid



```
1 . . . . 0
. . 1 . . .
. 0 . . 1 .
. . . 0 . 1
. . 0 . . .
0 . . . 1 .
```

- Cages:
  - Target: 1, Operation: AND, Cells: A1 A2
  - Target: 0, Operation: OR, Cells: B2 C2
  - Target: 1, Operation: XOR, Cells: C3 D3 E3
  - Target: 1, Operation: AND, Cells: E6 F6
  - Target: 0, Operation: XOR, Cells: F1 F2 F3
- Answer Grid as Outputted by your code:



```
1 0 1 0 0 0
0 1 1 0 0 0
0 0 1 0 1 0
1 0 0 0 1 1
0 1 0 1 0 0
0 0 0 0 1 1
```

Bonus:

- Accept a custom grid and cage constraints as input.
- Verify whether the custom grid is a valid KenLogic Grid by checking all the constraints programmatically:
  - Ensure each row and column has exactly three 1s and three 0s.
  - Check if both diagonals contain three 1s and three 0s.
  - Ensure that the cage constraints are met (i.e., the binary operations yield the target values).

This bonus task will test your ability to handle dynamic inputs and validate the grid according to the specified rules.

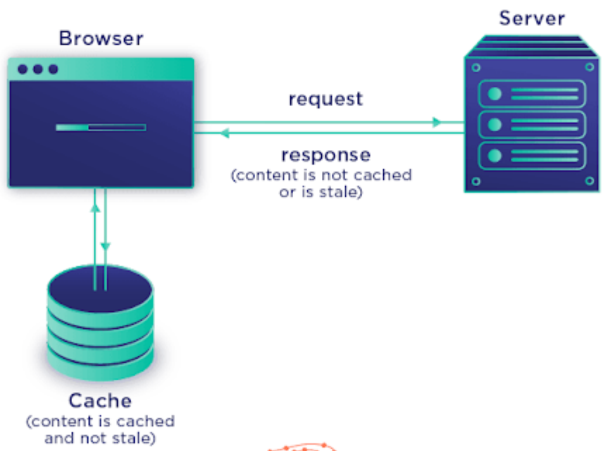
5. Hierarchically Partitioned InMemory KV store

Difficulty Level: High

Points: 20

Bonus: 10

Background:
Design a memory-efficient cache for sliding window attention mechanisms in neural network architectures, dynamically managing a fixed-size memory window for key-value representations..



Task Specifications:

1. Implement a SlidingWindowCache class with core operations:
   - Initialize with fixed maximum capacity.
   - Add new key-value pairs.
   - Retrieve the most recent N elements.
   - Automatically evict oldest elements when capacity is exceeded.
   - Support sliding window operations.
2. Cache Constraints:
   - Fixed memory size of K elements.
   - Maintain insertion order.
   - Handle cache overflow gracefully.

Example:

Python

```
# Initialize a cache with capacity 5
cache = SlidingWindowCache(capacity=5)

# Add elements
cache.add((1, 'embedding1'))
cache.add((2, 'embedding2'))
# ... continue adding elements

# Retrieve recent window
recent_window = cache.get_recent_window(3)
# Should return last 3 added elements
```

Bonus Challenge:

- Implement a memory-efficient version that can:
  - Track element access frequencies.
  - Support partial cache updates.

- Minimize memory fragmentation.
- Optimize read and write operations.

OpenCV

Instructions:

- This part consists of 5 questions along with one Bonus part for some question
- You can submit either .ipynb files (Jupyter Notebooks, Google Collab file not the link) or python .py files (in a ZIP folder).
- These questions are meant to help you learn by exploring on your own. You can look up concepts online to understand them but don't copy any code directly. Instead, understand it and create your version.
- You're not supposed to hardcode the inputs. The inputs should be user-driven. Make sure to check that the inputs meet the required conditions. For example, if a number is expected but the user enters a letter, show an error message like "Incorrect Input."
- A Well Commented Code is expected so everyone can understand it properly.

1. Chroma Cursor Explorer with Pixel Morph Mode

Difficulty Level: Easy

Points: 05

Bonus: 05

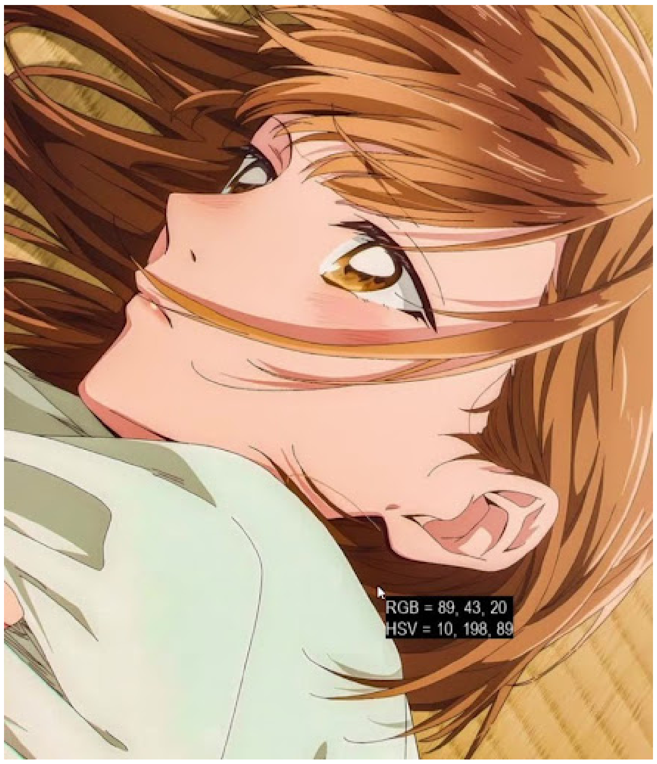
Problem Statement:
Your task is to create a program that detects the pixel of an image that the mouse pointer is on and print the RGB and HSV values of the pixel just below the cursor. You are free to use any libraries, such as OpenCV, NumPy, or Pygame.

Requirements:

- **Libraries:** You can use libraries such as OpenCV, NumPy, and Pygame to implement the solution.
- **Image Input:** The program should load an image (you can provide a sample image).
- **Mouse Interaction:** The program must detect the coordinates of the pixel where the mouse pointer is located.

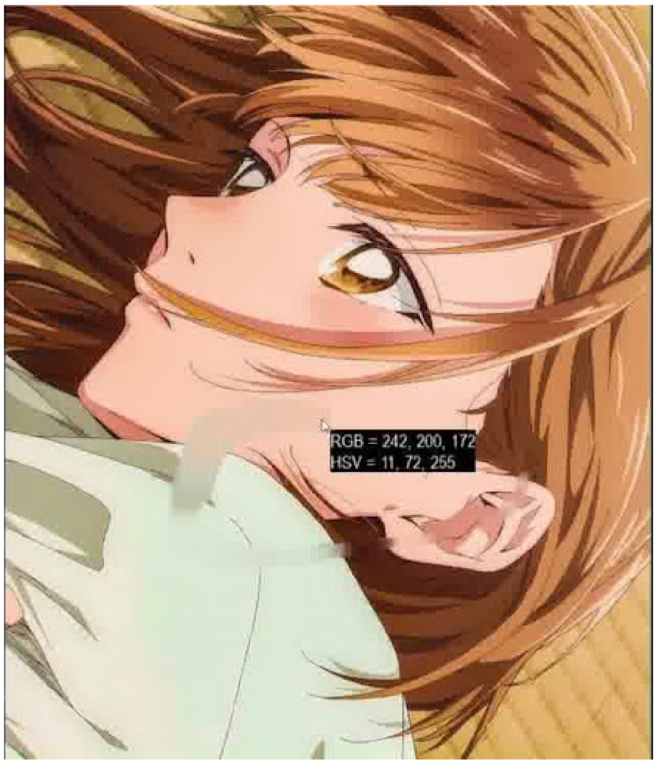
- **Output:**
  - Display the RGB and HSV color values of the pixel below the cursor.
  - The color values must be updated in real-time as the mouse moves over the image.
- Display Window:
  - The image should be displayed in a window.
  - The color values should appear near the mouse cursor in a readable format. Ensure the text does not obstruct the image or extend outside the window.

Sample Output:



```
RGB = 89, 43, 20
HSV = 10, 198, 89
```

Output



```
RGB = 242, 200, 172
HSV = 11, 72, 255
```

Output for pixelated version

You can see a Video Output Here

Bonus:
Enhance your program to include a "Random Pixel Art Generator" mode. When enabled (using a specific key, such as R), the program should randomly generate a pixelated pattern under the cursor, and we should be able to pixelate the pixels on the

cursor as we drag it as shown above. Users should be able to adjust the size of the blocks (pixel size) using the + and - keys.
Here is the Sample output for the Bonus task

2. StreetFlow: Motion Parsing Engine

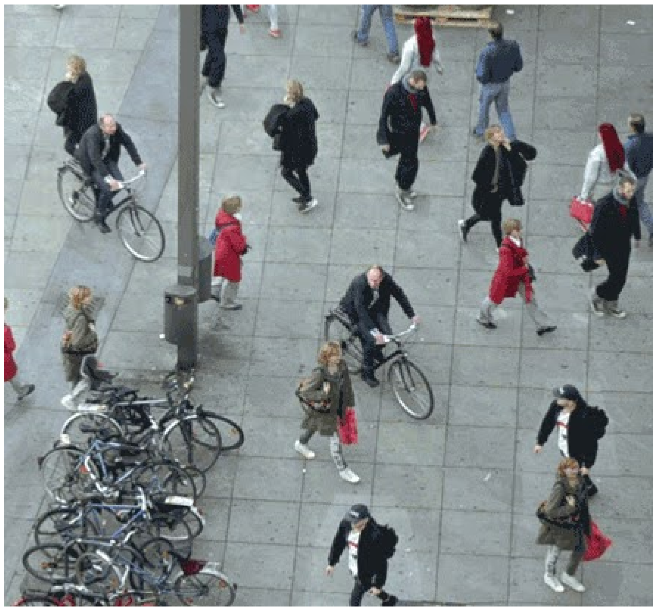
Difficulty Level: Medium

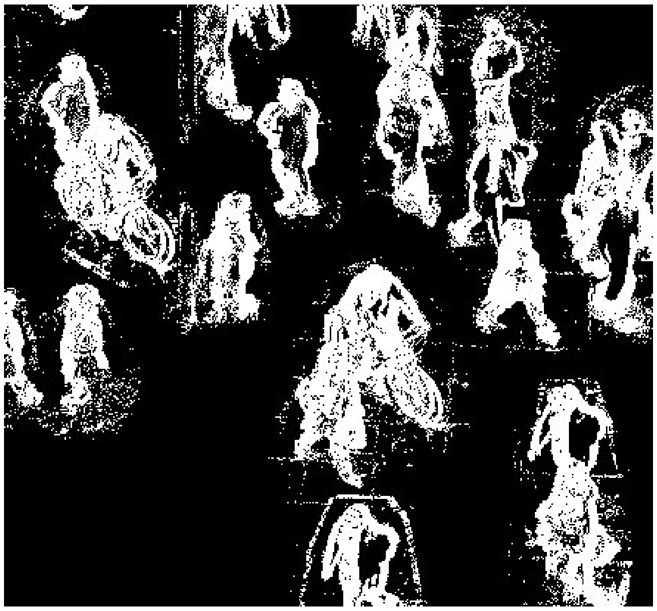
Points: 10

Bonus: 05

Problem Statement:
Your next task is to develop a segmentation model which would be used for segmenting pedestrians walking on the street by only using NumPy and OpenCV modules.
Take an input gif, then perform the segmentation.

Example: You should Segment the pedestrians like this:


Input


Output

You can see Input and Output Here

Bonus:
Try to implement or give a procedure on how you can implement fine-grain segmentation (i.e., DNN) on the given or captured video.

3. Seamless Horizon Fusion Challenge

Difficulty Level: Medium

Points: 10

Bonus: 05

Problem Statement:
Your task is to develop a program that stitches multiple images together to create a seamless panorama using stitching logic implemented from scratch. You are not allowed to use built-in stitching functions like $cv2.createStitcher()$ or $cv2.Stitchercreate()$. Instead, you must manually implement the key steps such as feature detection, matching, and homography estimation to align and blend the images into a panorama. Additionally, your program should include post-processing techniques such as cropping unwanted black borders and optionally complementing the sky. Download the sample Input Images from the link below, or you could use your own set of Images also.

Requirements:

- **Input:** A directory containing multiple overlapping images that need to be stitched into a panorama.
- **Output:** A single panoramic image saved as a file $(e.g., panoramaoutput.jpg)$.

Example:
You can find sample input Images Here

Sample Output:

Panaromic View

Bonus:

- If your program can handle stitching images in different orientations (horizontal, vertical, or mixed).
- If you implement advanced blending techniques, such as multi-band blending, for smoother transitions between images.
- If you provide an interactive interface for enabling or disabling sky complementation

4. Genesis of Image Alchemy: From Pixels to Masterpieces

Difficulty Level: Medium

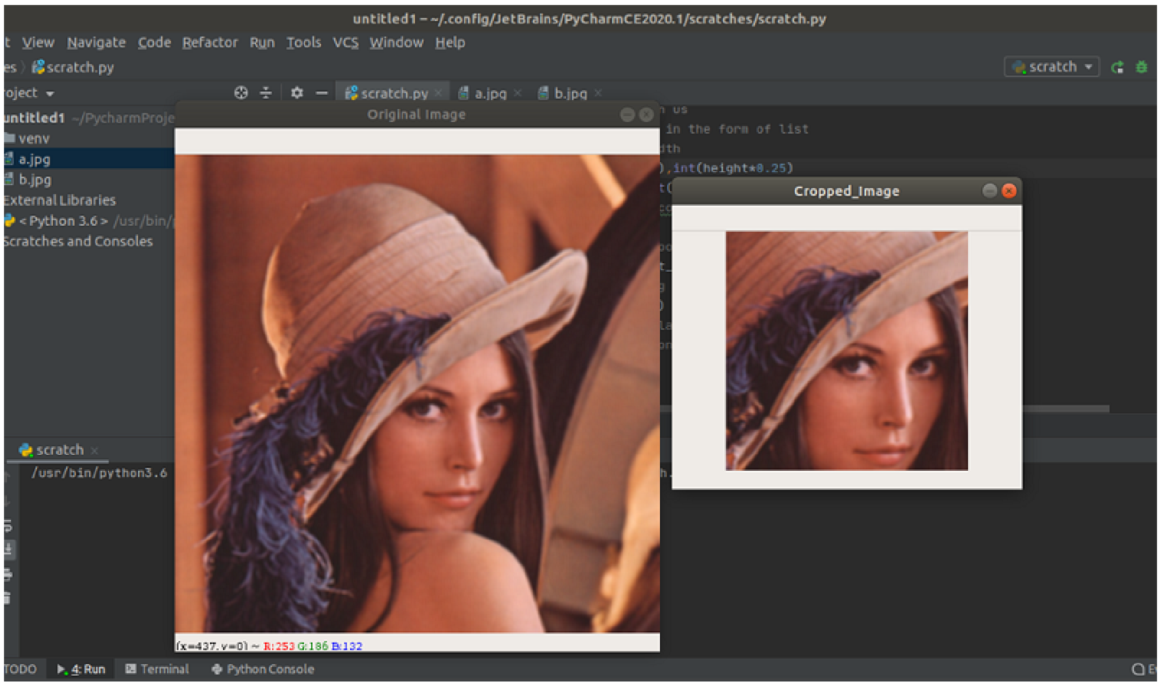
Points: 10

Bonus: 05

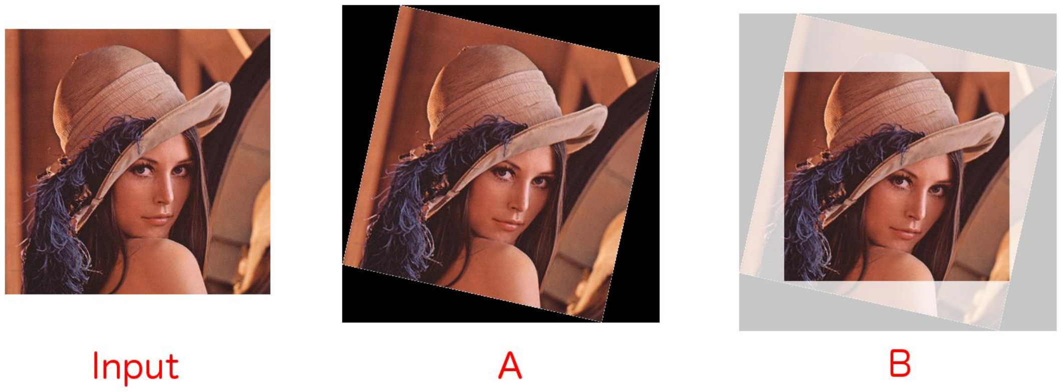
Problem Statement:
You are tasked with implementing advanced image processing operations using NumPy and OpenCV. The goal is to manipulate images programmatically by performing tasks like cropping, rotating to any angle, noise removal, and edge detection. Additionally, explore bonus tasks like pencil sketching and advanced morphology.// **Task:**

1. **Cropping an image:** Crop a specific region of interest (ROI) from the given image using basic NumPy operations.

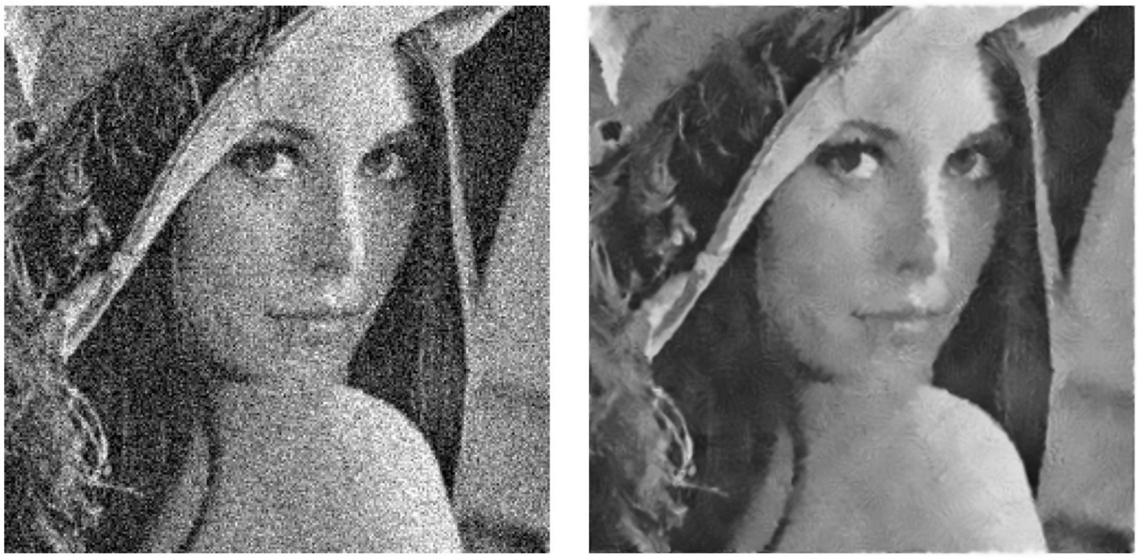
- Input: An image and the coordinates of the region (top-left and bottom-right).
- Output: Display the cropped portion of the image.



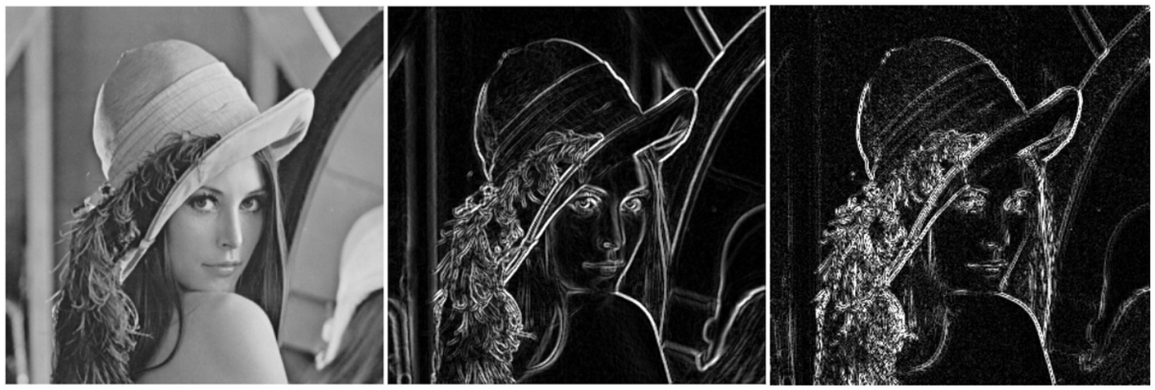
2. **Rotating an Image to Any Angle:**Rotate the image to a user-defined angle without using built-in functions.
    - Input: An image (as a NumPy array) and the desired rotation angle (in degrees).
    - Output: Display the rotated image.
    - Additional Requirement: Handle resizing or cropping due to rotation.



3. **Removing Noise Using a Mean Filter:**Apply a mean filter to remove noise from the image programmatically.
    - Input: A grayscale image and a kernel size ($e.g., 3 * 33\, times\, 33 * 3$).
    - Output: Display the denoised image.
    - Restriction: Do not use built-in filtering functions.



4. **Edge Detection:**Detects the edges of objects in an image programmatically.
    - Input: A grayscale image.
    - Output: Display the edge-detected image.
    - Restriction: Do not use built-in edge-detection functions.



Edge Detection

Bonus:

1. **Pencil Sketch Transformation:**Convert an image into a pencil sketch.
   - Input: A color image.
   - Output: A grayscale pencil-sketch-like image.
2. **Image Morphology: Dilation and Erosion:**Implement dilation and erosion for a binary image.
   - Input: A binary image and kernel size $k * kk \times kk * k$.
   - Output: Perform and display the results of both dilation and erosion.

Feel free to experiment various Morphology Techniques and create your own unique outputs.

5. MotionMath - Computing via Motions

Difficulty Level: Hard

Points: 20

Bonus: 10

Problem Statement:
Welcome to MotionMath, a next-generation computational experience where the boundaries between motion and computation are blurred. Using the power of OpenCV and advanced gesture recognition, your task is to create an interactive environment where hand movements become the conduit for real-time arithmetic execution. This is not merely a calculator — it's a dynamic, gesture-powered system that empowers the user to control operations seamlessly through intuitive motions.

Requirements:

1. Dynamic Hand Segmentation:
   - Implement a precise segmentation system to isolate and track hand gestures in a live video stream, using background subtraction, motion detection, and contour extraction.

    - Ensure robust segmentation under diverse environmental conditions, including lighting variations and complex backgrounds, for a flawless user experience.

2. Real-Time Gesture Decoding:
    - Leverage the provided pre-trained model weights to recognize and classify hand gestures as numeric digits and operators.
    - Enable real-time feedback by displaying the decoded gestures live, ensuring seamless interaction between user input and system output.
3. Arithmetic Magic:
    - Decode the sequence of gestures (first number, operator, second number) and execute the corresponding arithmetic operation with minimal delay.
    - Implement the basic operations (addition, subtraction, multiplication, division), ensuring fluid transitions between gestures for intuitive control.
    - Display the result with precision, ensuring an instantaneous response.

Bonus:

- Introduce a dynamic gesture validation feedback loop, allowing users to receive immediate confirmation of input gestures via visual or auditory cues.
- Expand the system to support complex mathematical operations, such as exponentiation, percentages, or multi-digit number input, to enhance the functionality.

Note:

In MotionMath, computing via motions transforms every gesture into an intuitive interaction with numbers. The challenge lies in blending gesture recognition with dynamic, responsive logic to create an experience that feels effortless and powerful. Use the provided pre-trained model weights for gesture recognition and focus on building a seamless, gesture-driven interface.

You can get pretrained weights Here