

NSDSYST - Final Project

Technical Documentation



GitHub Repository



<https://github.com/jm55DLSU/NSDSYST/>

System Specifications & Dependencies

System Specifications (Requirements)

1. Inputs:
 - a. Input & Output Paths
 - b. Brightness, Contrast & Sharpness Factors
2. Process: Image Processing
3. Outputs:
 - a. Processed Images
 - b. Report File: Input & Output Paths, # of Images Processed, # of Machines Used

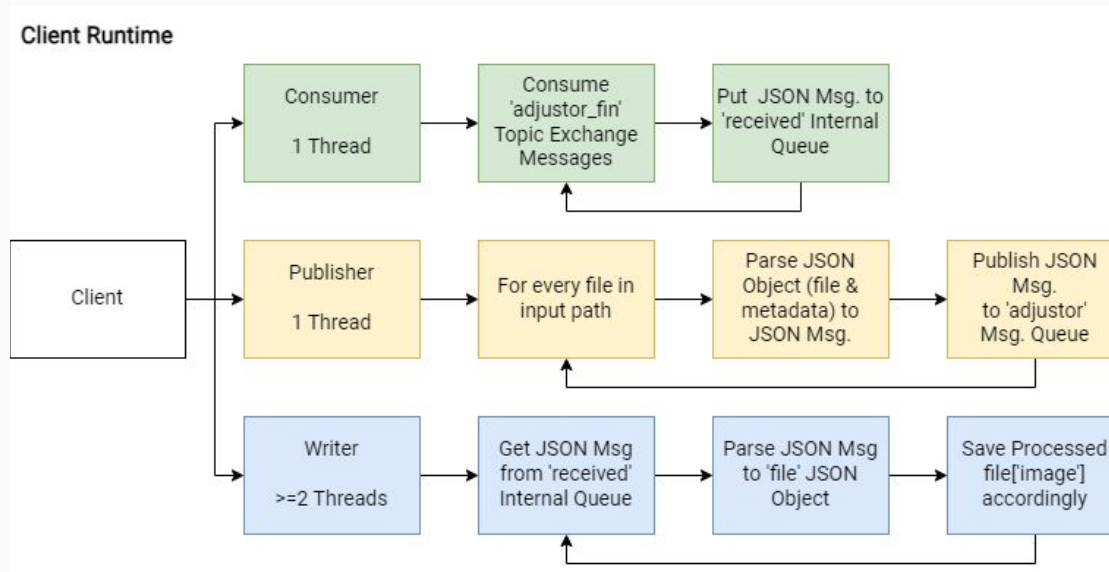
System Specifications (Implementation)

1. Uses OpenCV as image processing library.
2. Input images limited to 5MB .

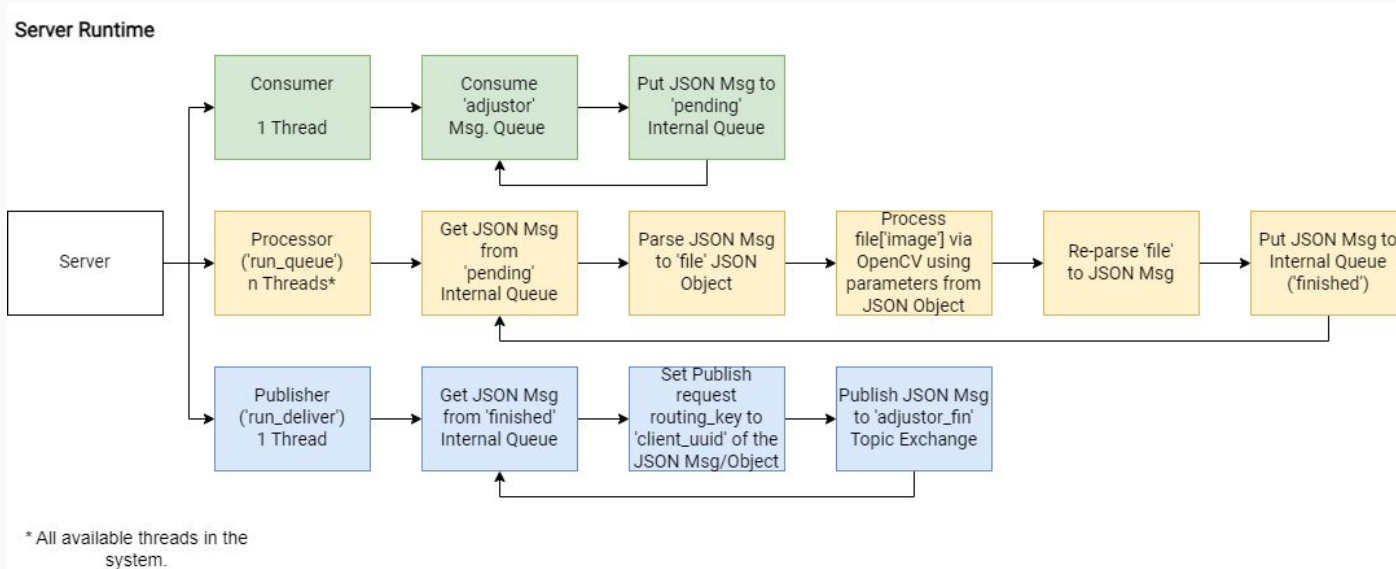
System Dependencies

1. Python 3 (at least 3.10.x)
2. Python Packages
 - a. pika
 - b. pillow (PIL)
 - c. Opencv-python

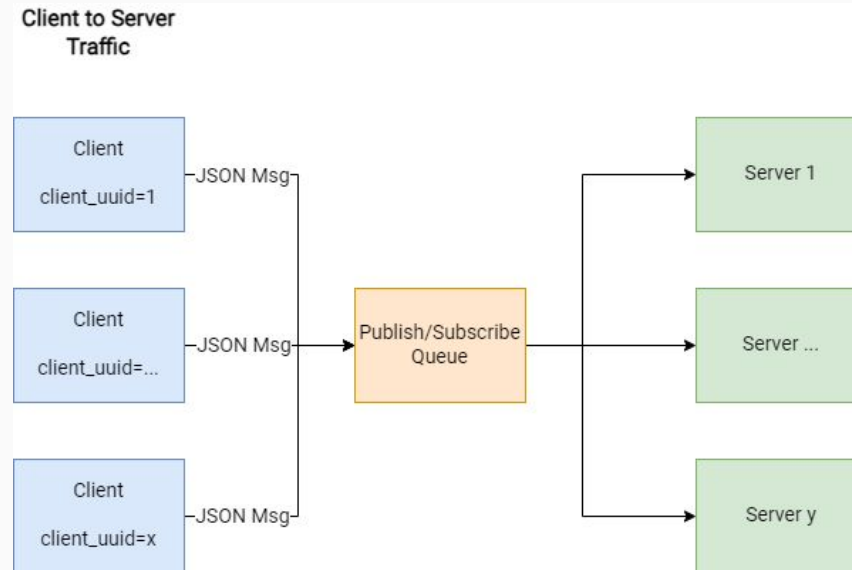
System Runtime Design (Client - MIMD)



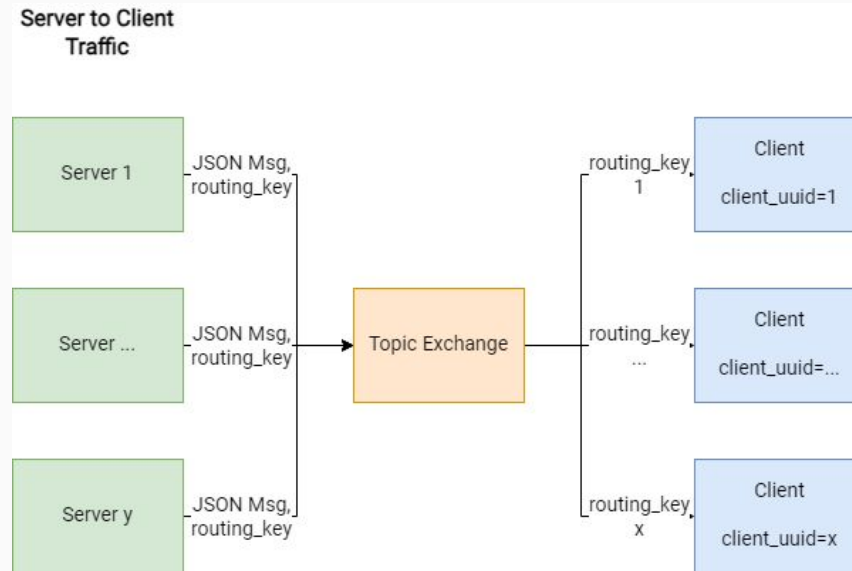
System Runtime Design (Server - MIMD)



System Traffic Design (Client to Server)



System Traffic Design (Server to Client)



Delivery Mechanisms (Client to Server)

- It uses a [Message Queue](#) mechanism to pass messages from Client to Server.
- The identity of the message origin is not monitored here, as the server should accept/consume messages regardless of who delivered them.
- The distribution of workload or message on each available server/consumer is handled by the message broker (i.e., RabbitMQ).

Delivery Mechanisms (Server to Client)

- Uses a Topic Exchange mechanism to selectively pass messages from the Server to the appropriate client via the Topic/routing_key value (i.e., Client UUID).
- The Server delivers the message to the Topic Exchange with a routing_key attached (i.e., Client UUID). The message broker then forwards the message to the appropriate Client that matches the routing_key specified.
- During its setup, the Client binds itself to the message broker's Topic Exchange, where the routing_key is set as its UUID. The Client thereafter listens and consumes any relevant messages the message broker delivers.

JSON Object Structure

Key	Raw Data Type	Description
input	String	Input folder path
filename	String	The filename of the image
output	String	Output folder path
brightness	Integer	Brightness factor (0-100)
contrast	Integer	Contrast factor (0-100)
sharpness	Integer	Sharpness (0-100)
image	OpenCV Mat Class *Stringified via pickle.dumps() → Base64 Encode → ASCII Decode	Actual image data
client_uuid	String	Client UUID

Benchmark Datasets

1. “Tokyo”

- a. File Size Range: 367 KB to 12.9 MB
- b. Total File Size: 134.2 MB
- c. Total Count: 55 Images
- d. Acceptable File Count (File Sizes \leq 5 MB): 49 Images (89.09%)
- e. Note: Represents processing heavy image files

2. “Crops”

- a. File Size Range: 2.9 KB to 1.8 MB
- b. Total File Size: 76.8 MB
- c. Total Count: 773 Images
- d. Acceptable File Count (File Sizes \leq 5 MB): 773 Images (100%)
- e. Note: Represent processing small image files

Benchmark System (Laptop)

1. Server (Guest VM):
 - a. CPU (Threads): 4
 - b. RAM: 2048 MB
2. Client (Host):
 - a. CPU (Threads): 8
 - b. RAM: 8192 MB
3. RabbitMQ: Same as Client's

Benchmark Results (Laptop)

	Benchmark ID (No. of Machines:Run)	Time (s) Tokyo Dataset	Time (s) Crops Dataset
4T	1:1	67.8779	38.1426
	1:2	45.0550	34.7252
	1:3	45.8793	33.6548
	1:4	48.2554	35.0385
8T	2:1	39.5424	29.7509
	2:2	39.7682	29.6472
	2:3	46.0564	30.4623
	2:4	40.4689	29.4173

Benchmark Results (Laptop)

No. of Machines	Average Time (s) Tokyo Dataset	Average Time (s) Crops Dataset
1 (4 Threads Total)	51.7669	35.3093
2 (8 Threads Total)	41.4590	29.8194

No. of Machines	Time Reduction Tokyo Dataset	Time Reduction Crops Dataset
1 to 2 Machines	19.91%	15.55%

Known Issues (as of Nov. 19, 2023)

- The Client does not gracefully terminate (However, it completes its task properly).
- Bottleneck was determined to be the communications between the Server and the Message Broker (i.e., reliant on its performance), which is especially noticeable for large image files. Hence, the times may not always be as what's seen in the benchmarks.

Screenshots

(During Development)

The image displays a multi-client single-server transaction system running in a virtual machine environment. The interface is split into three main sections: a code editor, a terminal, and a log viewer.

Code Editor (Client.py): The code defines a `Client_Driver` class with methods for generating JSON data, processing images, and handling client requests. The `menu` method is currently active, showing input fields for location, brightness, contrast, and sharpness.

Terminal: The terminal shows the execution of the `Client.py` script. It displays the output of the `menu` method, including the input folder path, output folder path, and the generated JSON data. The terminal also shows the output of the `process_image` method, which processes the input image and returns the processed image.

Log Viewer: The log viewer displays the output of the `Server.py` script. It shows the server's response to the client's request, including the processed image and the generated JSON data. The log viewer also shows the output of the `process_image` method, which processes the input image and returns the processed image.

The interface is running on an Ubuntu VM VirtualBox. The top bar shows the file explorer, machine view, input devices, and help. The bottom bar shows the terminal, output, debug console, and ports. The status bar at the bottom indicates the current file, line, column, and encoding.

Multi-Client Single-Server Transaction

The image shows a dual-monitor setup. The left monitor displays the Visual Studio Code editor with the 'Client.py' script. The script is a Python program that interacts with a server. It includes a 'write_report' function that logs the number of images processed, time elapsed, and file names. The 'main' function prompts the user to enter a run command (R) or quit (Q). The terminal window shows the execution of the script, with output indicating the receipt and processing of various images (e.g., coffee_images48.jpg, cucumber_image (27).jpg, coconute_image (8).jpg, etc.).

The right monitor displays two Oracle VM VirtualBox windows, each running an Ubuntu VM. The top VM window shows a log of server activity, with entries indicating the receipt and processing of images (e.g., 'Returning coffee_images59.jpg...', 'Returning coffee_images62.jpg...', 'Returning coconute_image (6).jpg...', etc.). The bottom VM window shows a log of client activity, with entries indicating the receipt and processing of images (e.g., 'Received cucumber_image (15).jpg...', 'Thread 4 Processed cucumber_image (15).jpg @ 0.02s', 'Queued cucumber_image (17).jpg, Queue = 1', etc.).

Single-Client Multi-Server Transaction

The image displays a Kali Linux terminal window and two Oracle VM VirtualBox windows, all showing the results of a network throughput test. The terminal window, titled 'kali@ESCALONA-LTP02: ~/Documents/GitHub/NSDSYST/FinalProj', shows a list of transactions and their completion times. The two VirtualBox windows, titled 'Ubuntu1 [Running] - Oracle VM VirtualBox' and 'Ubuntu2 [Running] - Oracle VM VirtualBox', show the results of the test on the respective virtual machines. The terminal window also displays a summary of the test results, including the number of transactions, the total time, and the average throughput.

Kali Linux Terminal Output:

```
2023-11-18 21:47:47.758937: Client - Thread 0 finished writing wheat_image (40).jpg
2023-11-18 21:47:47.778039: Client - Thread 0 finished writing wheat_image (50).jpg
2023-11-18 21:47:47.882425: Client - Thread 1 finished writing tomato_image (9).jpg
2023-11-18 21:47:48.087770: Client - Received wheat_image (42).jpg
2023-11-18 21:47:48.088052: Client - Queueing File wheat_image (42).jpg for writing...
2023-11-18 21:47:48.207427: Client - Thread 0 finished writing wheat_image (42).jpg
2023-11-18 21:47:49.090878: Client - Received wheat_image (43).jpg
2023-11-18 21:47:49.091275: Client - Queueing File wheat_image (43).jpg for writing...
2023-11-18 21:47:49.315062: Client - Thread 1 finished writing wheat_image (43).jpg
2023-11-18 21:47:51.284286: Client - Received wheat_image (48).jpg
2023-11-18 21:47:51.287715: Client - Queueing File wheat_image (48).jpg for writing...
2023-11-18 21:47:51.307532: Client - Received wheat_image (6).jpg
2023-11-18 21:47:51.307599: Client - Queueing File wheat_image (6).jpg for writing...
2023-11-18 21:47:51.312809: Client - Thread 1 finished writing wheat_image (6).jpg
2023-11-18 21:47:51.377311: Client - Thread 0 finished writing wheat_image (48).jpg
2023-11-18 21:47:51.599562: Client - Received wheat_image (49).jpg
2023-11-18 21:47:51.599926: Client - Queueing File wheat_image (49).jpg for writing...
2023-11-18 21:47:51.600589: Client - Received wheat_image (8).jpg
2023-11-18 21:47:51.600889: Client - Queueing File wheat_image (8).jpg for writing...
2023-11-18 21:47:51.624823: Client - Thread 0 finished writing wheat_image (8).jpg
2023-11-18 21:47:51.817175: Client - Thread 1 finished writing wheat_image (49).jpg
2023-11-18 21:47:52.145459: Client - Received wheat_image (53).jpg
2023-11-18 21:47:52.145553: Client - Queueing File wheat_image (53).jpg for writing...
2023-11-18 21:47:52.215556: Client - Thread 0 finished writing wheat_image (53).jpg
```

Ubuntu1 [Running] - Oracle VM VirtualBox Output:

```
2023-11-18 21:48:08.628071: Server - Thread 4 Processed tokyo (48).jpg @ 4.83s
2023-11-18 21:48:08.712343: Server - Thread 2 Processed tokyo (50).jpg @ 0.59s
2023-11-18 21:48:09.747498: Server - Thread 0 Processing tokyo (51).jpg...
2023-11-18 21:48:09.854167: Server - Thread 1 Processed tokyo (5).jpg @ 1.63s
2023-11-18 21:48:10.216519: Server - Returning tokyo (50).jpg...
2023-11-18 21:48:10.433975: Server - Queued tokyo (52).jpg, Queue = 1
2023-11-18 21:48:11.140853: Server - Queued tokyo (53).jpg, Queue = 2
2023-11-18 21:48:11.675601: Server - Thread 0 Processed tokyo (51).jpg @ 1.93s
2023-11-18 21:48:12.508383: Server - Thread 3 Processed tokyo (49).jpg @ 5.88s
2023-11-18 21:48:12.723437: Server - Queued tokyo (54).jpg, Queue = 3
2023-11-18 21:48:12.849610: Server - Returning tokyo (48).jpg...
2023-11-18 21:48:13.143602: Server - Queued tokyo (55).jpg, Queue = 4
2023-11-18 21:48:13.989597: Server - Queued tokyo (6).jpg, Queue = 5
2023-11-18 21:48:15.008588: Server - Queued tokyo (7).jpg, Queue = 5
2023-11-18 21:48:15.631390: Server - Thread 4 Processing tokyo (53).jpg...
2023-11-18 21:48:15.838744: Server - Thread 2 Processing tokyo (52).jpg...
2023-11-18 21:48:16.065648: Server - Thread 4 Processed tokyo (53).jpg @ 0.43s
2023-11-18 21:48:16.267276: Server - Returning tokyo (51).jpg...
2023-11-18 21:48:16.903871: Server - Queued tokyo (8).jpg, Queue = 5
2023-11-18 21:48:17.115862: Server - Queued tokyo (9).jpg, Queue = 6
2023-11-18 21:48:17.291827: Server - Returning tokyo (5).jpg...
```

Ubuntu2 [Running] - Oracle VM VirtualBox Output:

```
2023-11-18 21:47:51.753732: Client - Received tokyo (44).jpg
2023-11-18 21:47:51.753848: Client - Queueing File tokyo (44).jpg for writing...
2023-11-18 21:47:52.146481: Client - Thread 1 finished writing tokyo (44).jpg
2023-11-18 21:47:52.733105: Client - Thread 0 finished writing tokyo (42).jpg
2023-11-18 21:47:53.985827: Client - Received tokyo (45).jpg
2023-11-18 21:47:53.985935: Client - Queueing File tokyo (45).jpg for writing...
2023-11-18 21:47:55.323952: Client - Thread 1 finished writing tokyo (45).jpg
2023-11-18 21:47:56.940915: Client - Received tokyo (40).jpg
2023-11-18 21:47:56.941058: Client - Queueing File tokyo (40).jpg for writing...
2023-11-18 21:47:58.106691: Client - Thread 0 finished writing tokyo (40).jpg
2023-11-18 21:48:07.897784: Client - Received tokyo (46).jpg
2023-11-18 21:48:07.898755: Client - Queueing File tokyo (46).jpg for writing...
2023-11-18 21:48:11.135426: Client - Received tokyo (47).jpg
2023-11-18 21:48:11.135528: Client - Queueing File tokyo (47).jpg for writing...
2023-11-18 21:48:11.859396: Client - Received tokyo (50).jpg
2023-11-18 21:48:11.859502: Client - Queueing File tokyo (50).jpg for writing...
2023-11-18 21:48:12.908440: Client - Thread 1 finished writing tokyo (46).jpg
2023-11-18 21:48:13.380761: Client - Thread 1 finished writing tokyo (50).jpg
2023-11-18 21:48:14.350664: Client - Thread 0 finished writing tokyo (47).jpg
2023-11-18 21:48:17.101616: Client - Received tokyo (48).jpg
2023-11-18 21:48:17.102564: Client - Queueing File tokyo (48).jpg for writing...
```

Kali Linux Terminal Summary:

```
Mem[|||||] [6.79G/9.59G] Date: 2023-11-18
Swp[|||||] [194M/976M] Tasks: 138, 682 thr, 162 kthr; 8 running
Disk IO: 11.7% read: 1.59MiB/s write: 1.75MiB Load average: 7.69 3.18 1.37
Network: rx: 74.2MiB/s tx: 96.9MiB/s (5432/S Uptime: 00:06:55
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3350	kali	20	0	5666M	2060M	2032M	S	341.7	21.0	5:49.18	/usr/lib/virtualbox/Virtua
3822	kali	20	0	5595M	1993M	1962M	S	174.2	20.3	4:00.16	/usr/lib/virtualbox/Virtua
6345	kali	20	0	1304M	109M	38016	R	107.9	1.1	1:09.58	python3 Client.py
3406	kali	21	1	5666M	2060M	2032M	R	74.8	21.0	1:13.89	/usr/lib/virtualbox/Virtua
3407	kali	21	1	5666M	2060M	2032M	R	80.0	21.0	1:14.43	/usr/lib/virtualbox/Virtua
3404	kali	21	1	5666M	2060M	2032M	R	60.9	21.0	1:18.77	/usr/lib/virtualbox/Virtua
3405	kali	21	1	5666M	2060M	2032M	S	55.6	21.0	1:12.15	/usr/lib/virtualbox/Virtua
3892	kali	17	3	5595M	1993M	1962M	S	50.3	20.3	0:12.99	/usr/lib/virtualbox/Virtua

Multi-Client Single-Server Transaction Network Throughput

References

- [1] "AMQP 0-9-1 Model Explained — RabbitMQ," RabbitMQ. <https://www.rabbitmq.com/tutorials/amqp-concepts.html> (accessed Nov. 19, 2023).
- [2] "Part 4: RabbitMQ Exchanges, routing keys and bindings," CloudAMQP, 2019. <https://www.cloudamqp.com/blog/part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html> (accessed Nov. 19, 2023).
- [3] "RabbitMQ Topic Exchange Explained," CloudAMQP, 2022. <https://www.cloudamqp.com/blog/rabbitmq-topic-exchange-explained.html> (accessed Nov. 19, 2023).
- [4] "RabbitMQ tutorial," Topics — RabbitMQ. <https://www.rabbitmq.com/tutorials/tutorial-five-python.html> (accessed Nov. 19, 2023).
- [5] Pankaj, "Python Multiprocessing Example," DigitalOcean, Aug. 03, 2022. Accessed: Nov. 19, 2023. [Online]. Available: <https://www.digitalocean.com/community/tutorials/python-multiprocessing-example>
- [6] "RabbitMQ tutorial," Work Queues — RabbitMQ. <https://www.rabbitmq.com/tutorials/tutorial-two-python.html> (accessed Nov. 19, 2023).
- [7] K41F4r, "Sending OpenCV image in JSON," Stack Overflow. <https://stackoverflow.com/questions/55892362/sending-opencv-image-in-json/55900422#55900422> (accessed Nov. 19, 2023).
- [8] pika, "pika/examples/basic_consumer_threaded.py at 1.0.1 · pika/pika," GitHub. https://github.com/pika/pika/blob/1.0.1/examples/basic_consumer_threaded.py (accessed Nov. 19, 2023).
- [9] S. A. Khan, "How to change the contrast and brightness of an image using OpenCV in Python?," Tutorialspoint, 2023. <https://www.tutorialspoint.com/how-to-change-the-contrast-and-brightness-of-an-image-using-opencv-in-python> (accessed Nov. 19, 2023).
- [10] "How to Sharpen an Image with OpenCV," How to use OpenCV, 2023. <https://www.opencvhelp.org/tutorials/image-processing/how-to-sharpen-image/> (accessed Nov. 19, 2023).
- [11] "Python OpenCV cv2.imwrite method," GeeksforGeeks, Aug. 05, 2019. Accessed: Nov. 19, 2023. [Online]. Available: <https://www.geeksforgeeks.org/python-opencv-cv2-imwrite-method/>
- [12] M. W. Azam, "Agricultural crops image classification," Kaggle. <https://www.kaggle.com/datasets/mdwaquarazam/agricultural-crops-image-classification> (accessed Nov. 19, 2023)