Parallel Distributed Systems

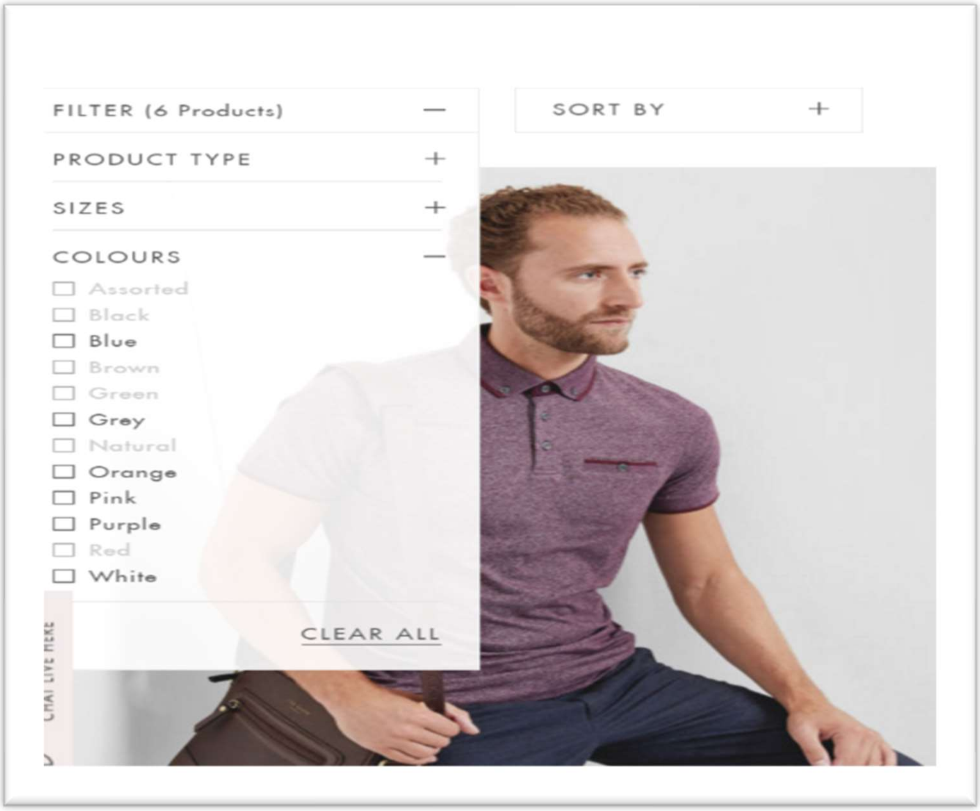
18CSC401

त्यागेनैके अमृतत्वमानशुः

MATA AMRITANANDAMAYI MATH

श्रद्धावान् लभते ज्ञानम्

MULTI-THREADING FOR IMAGE FILTERING IN E-COMMERCE WEBSITES



ISHWARYA S

CB.SC.I5DAS20113

Context

Online shopping is a form of electronic commerce which allows consumers to directly buy goods or services from a seller over the Internet using web browser or a mobile app. Filters play a crucial role in online shopping for both e-commerce platforms and customers. They provide a way to refine search results and customize the online shopping experience, offering several benefits for both customers and business.

Abstract:

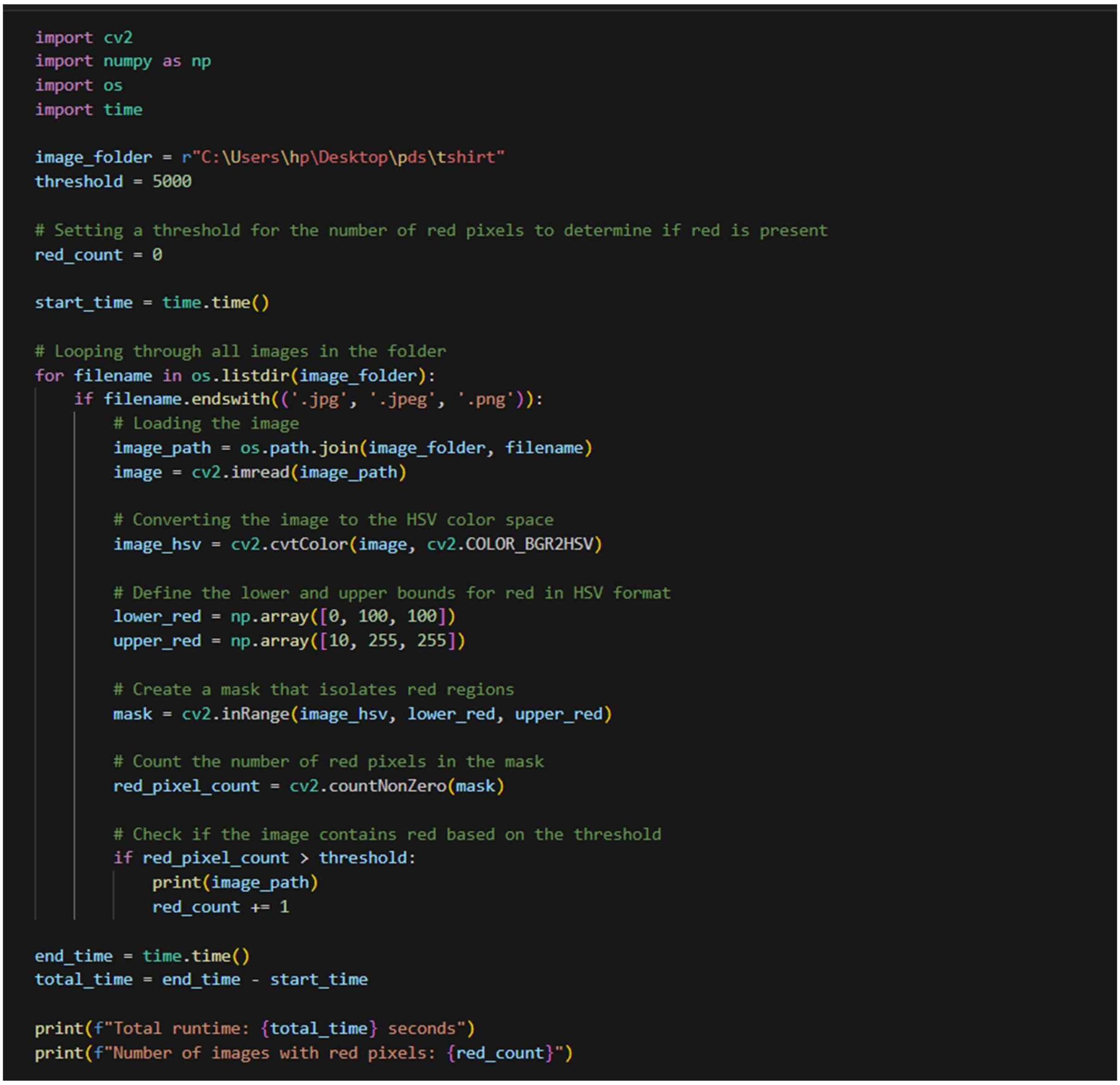
In the fast-paced world of e-commerce, delivering a seamless and efficient shopping experience is essential. An e-commerce clothing store often carries an extensive inventory, making real-time product filtering a challenging task. This project focuses on the implementation of parallel processing techniques to enhance filtering capabilities within the context of a clothing store. By harnessing the power of parallelism, the project aims to significantly reduce processing times, improve search relevance, and elevate the overall user experience.

About the Dataset:

This data set is taken from **Kaggle.** This data is collected from online e-commerce sites. It has various images of the t-shirts, with multiple colours and multiple t-shirt patterns.

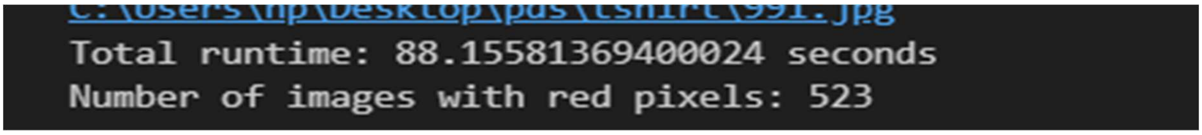
Reference to the dataset:
(https://www.kaggle.com/datasets/sunnykusawa/tshirts).

Python:



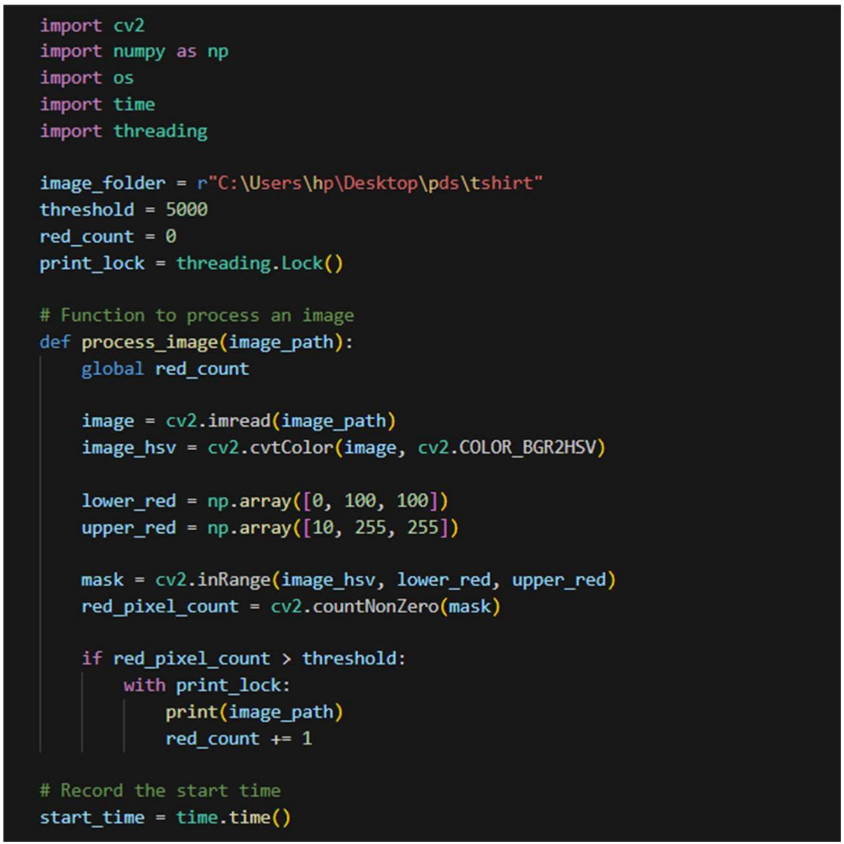
```python
import cv2
import numpy as np
import os
import time

image_folder = r"C:\Users\hp\Desktop\pds\tshirt"
threshold = 5000

# Setting a threshold for the number of red pixels to determine if red is present
red_count = 0

start_time = time.time()

# Looping through all images in the folder
for filename in os.listdir(image_folder):
    if filename.endswith(('.jpg', '.jpeg', '.png')):
        # Loading the image
        image_path = os.path.join(image_folder, filename)
        image = cv2.imread(image_path)

        # Converting the image to the HSV color space
        image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        # Define the lower and upper bounds for red in HSV format
        lower_red = np.array([0, 100, 100])
        upper_red = np.array([10, 255, 255])

        # Create a mask that isolates red regions
        mask = cv2.inRange(image_hsv, lower_red, upper_red)

        # Count the number of red pixels in the mask
        red_pixel_count = cv2.countNonZero(mask)

        # Check if the image contains red based on the threshold
        if red_pixel_count > threshold:
            print(image_path)
            red_count += 1

end_time = time.time()
total_time = end_time - start_time

print(f"Total runtime: {total_time} seconds")
print(f"Number of images with red pixels: {red_count}")
```

Run time – Sequentially:

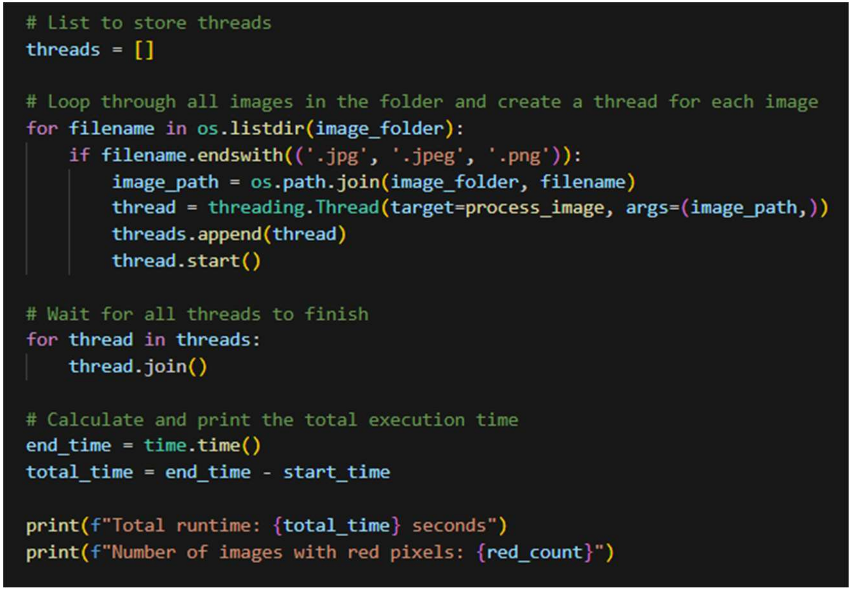


```
C:\Users\hp\Desktop\pds\tshirt\991.jpg
Total runtime: 88.15581369400024 seconds
Number of images with red pixels: 523
```

Multi-Threading in Python:

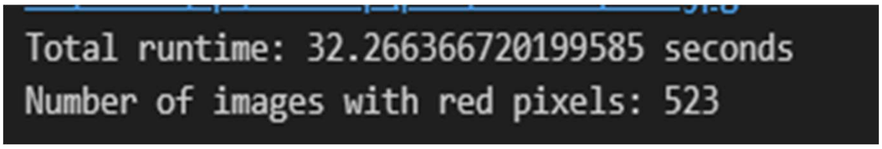


```python
import cv2
import numpy as np
import os
import time
import threading

image_folder = r"C:\Users\hp\Desktop\pds\tshirt"
threshold = 5000
red_count = 0
print_lock = threading.Lock()

# Function to process an image
def process_image(image_path):
    global red_count

    image = cv2.imread(image_path)
    image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    lower_red = np.array([0, 100, 100])
    upper_red = np.array([10, 255, 255])

    mask = cv2.inRange(image_hsv, lower_red, upper_red)
    red_pixel_count = cv2.countNonZero(mask)

    if red_pixel_count > threshold:
        with print_lock:
            print(image_path)
            red_count += 1

# Record the start time
start_time = time.time()
```



```python
# List to store threads
threads = []

# Loop through all images in the folder and create a thread for each image
for filename in os.listdir(image_folder):
    if filename.endswith(('.jpg', '.jpeg', '.png')):
        image_path = os.path.join(image_folder, filename)
        thread = threading.Thread(target=process_image, args=(image_path,))
        threads.append(thread)
        thread.start()

# Wait for all threads to finish
for thread in threads:
    thread.join()

# Calculate and print the total execution time
end_time = time.time()
total_time = end_time - start_time

print(f"Total runtime: {total_time} seconds")
print(f"Number of images with red pixels: {red_count}")
```

Run time => Multi-threading in Python:



```
Total runtime: 32.266366720199585 seconds
Number of images with red pixels: 523
```

Conclusion:

This project demonstrates the tangible benefits of incorporating parallel processing techniques, showcasing the potential to enhance the speed and efficiency of image classification tasks, with implications for broader applications in computational tasks across various domains.