## Soft Drinks Can Detection

CSE 4127 : Image Processing and Computer Vision Laboratory

By

**Anisa Walida**

Roll: 1907087



Presented to

**Dr. Sk. Mohammad Masudul Ahsan**

Professor

CSE, KUET

**Dipannita Biswas**

Lecturer  
CSE, KUET

**Department of Computer Science and Engineering**

**Khulna University of Engineering & Technology**

**Khulna 9203, Bangladesh**

**November 2023**

**Objectives**

* To develop an automated system for detecting and identifying Pepsi and Coke cans in images.
* To improve accuracy and efficiency in the can detection process using advanced computer vision techniques.
* To provide a scalable and adaptable solution for various industrial applications.

## Introduction

The beverage industry often requires automated systems to ensure that products are correctly labeled and packaged. Manual inspection is time-consuming and prone to errors, leading to inefficiencies and increased costs. This project aims to address these challenges by developing an automated can detection system using computer vision techniques, including color segmentation and template matching.

## Theory

## Color Segmentation

Color segmentation is a fundamental technique in computer vision and image processing, used to partition an image into regions based on color properties. This process typically begins with converting an image from the RGB color space, which is sensitive to lighting and lacks separation between color and intensity, to a more suitable color space such as HSV or Lab. The HSV color space separates color information (hue) from intensity information (saturation and value), facilitating the definition of color ranges, while the Lab color space approximates human vision, making it useful for complex segmentation tasks. Once the image is converted, thresholding is applied to create binary masks by setting ranges for the color values, where pixels within the range are turned on (white) and those outside are turned off (black). This binary mask highlights the regions of interest, which can then be isolated from the original image. To refine the mask and improve segmentation quality, morphological operations such as erosion, dilation, opening, and closing are used to remove noise and fill gaps. These steps collectively enhance the accuracy and robustness of the segmentation, enabling effective isolation of regions based on color for further analysis in applications like object detection and image recognition.

## Template Matching

Template matching is a widely-used technique in computer vision for locating a sub-image or template within a larger image. The process begins by selecting a template image that represents the object or pattern to be detected. This template is then matched against different regions of the target image to find the best match. To achieve this, a method such as normalized cross-correlation (cv2.TM\_CCOEFF\_NORMED) is often employed. In this method, the template and the corresponding region in the target image are normalized by subtracting the mean and dividing by the standard deviation, enhancing the comparison's robustness to lighting changes and variations in intensity. The normalized template is then convolved with the target image using the cv2.filter2D function, which essentially slides the template over the image and computes a similarity score at each position. These scores are stored in a result matrix, where higher values indicate better matches. To account for varying intensities across the image, local means and variances of the image are computed, and the result is normalized by these local statistics. This process ensures that the similarity score reflects the true correlation between the template and the image region, rather than being influenced by local intensity variations. Finally, the positions in the result matrix that exceed a certain threshold are identified as matches, and these locations are marked on the target image, typically by drawing rectangles around the detected regions. This method is effective for various applications, such as object detection, pattern recognition, and image registration, providing a robust and accurate means of locating specific patterns within larger images.

## Methodology

1. **Load and Preprocess the Images:**

**Loading the Image:** The initial step involves loading the target image from a specified source. This image serves as the input for subsequent processing steps.

**Converting to HSV and Grayscale:** The loaded image is converted to the HSV (Hue, Saturation, Value) color space. The HSV color space separates color information (hue) from intensity information (saturation and value), which is beneficial for color-based segmentation. Additionally, the image is converted to grayscale, simplifying the image by removing color information and making it suitable for intensity-based operations.

**Histogram Equalization:** Histogram equalization is applied to the grayscale image. This technique enhances the contrast of the image by redistributing the intensity values, making it easier to detect edges and features.

1. **Detect Cans Using Color Segmentation:**

**Pepsi Can Detection:** To detect Pepsi cans, a specific blue color range is defined. The lower and upper bounds for the blue color in the HSV color space are set. A binary mask is created where pixels within the color range are set to white, and those outside the range are set to black.

**Coke Can Detection:** For Coke cans, two red color ranges are defined to account for variations in red hues. Two binary masks are created for these ranges and combined to capture all red regions.

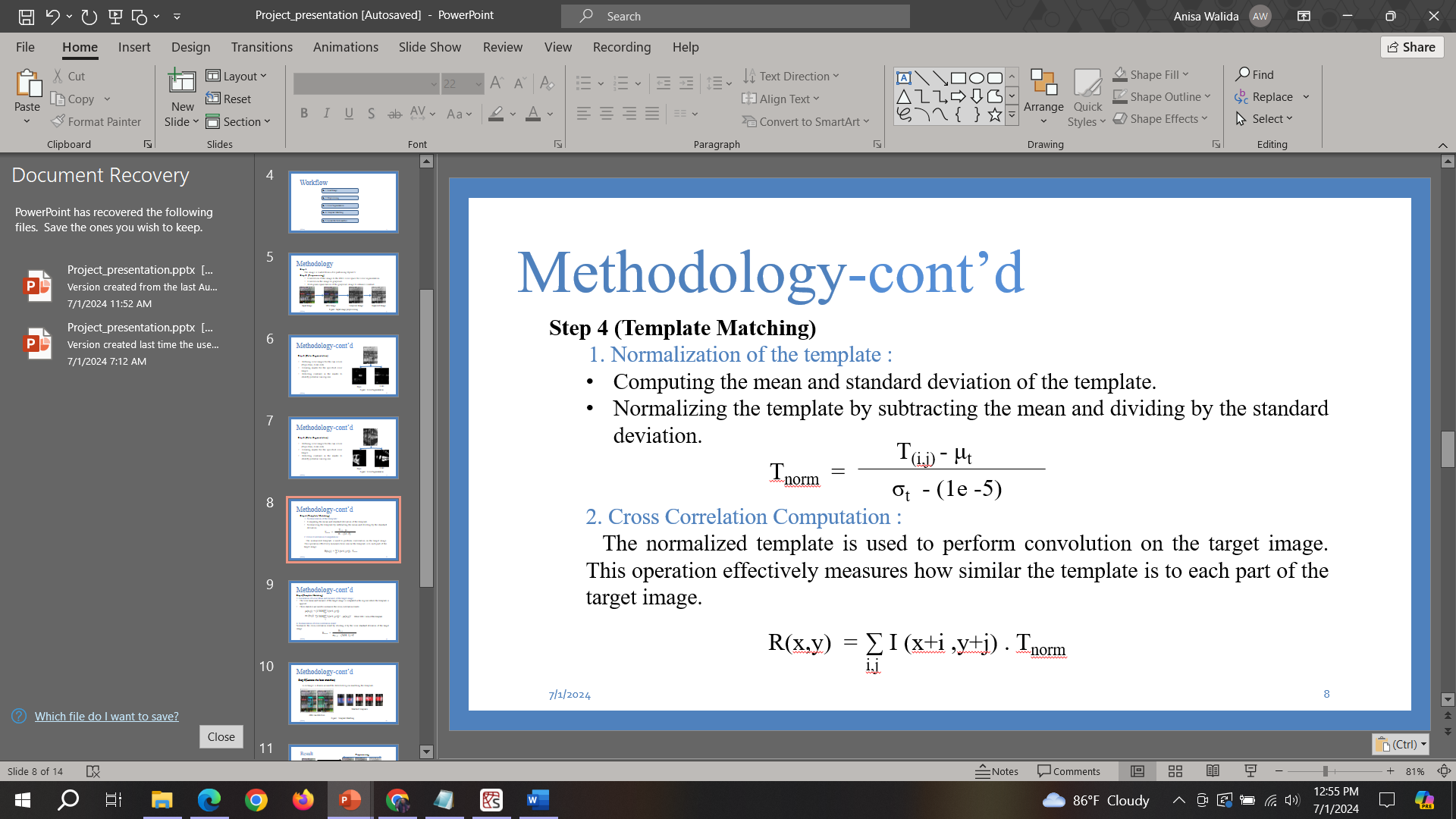
**Finding Contours:** Contours are found in the masks. Contours represent the boundaries of the detected regions, which are potential can regions. The formula for the mask is:

​

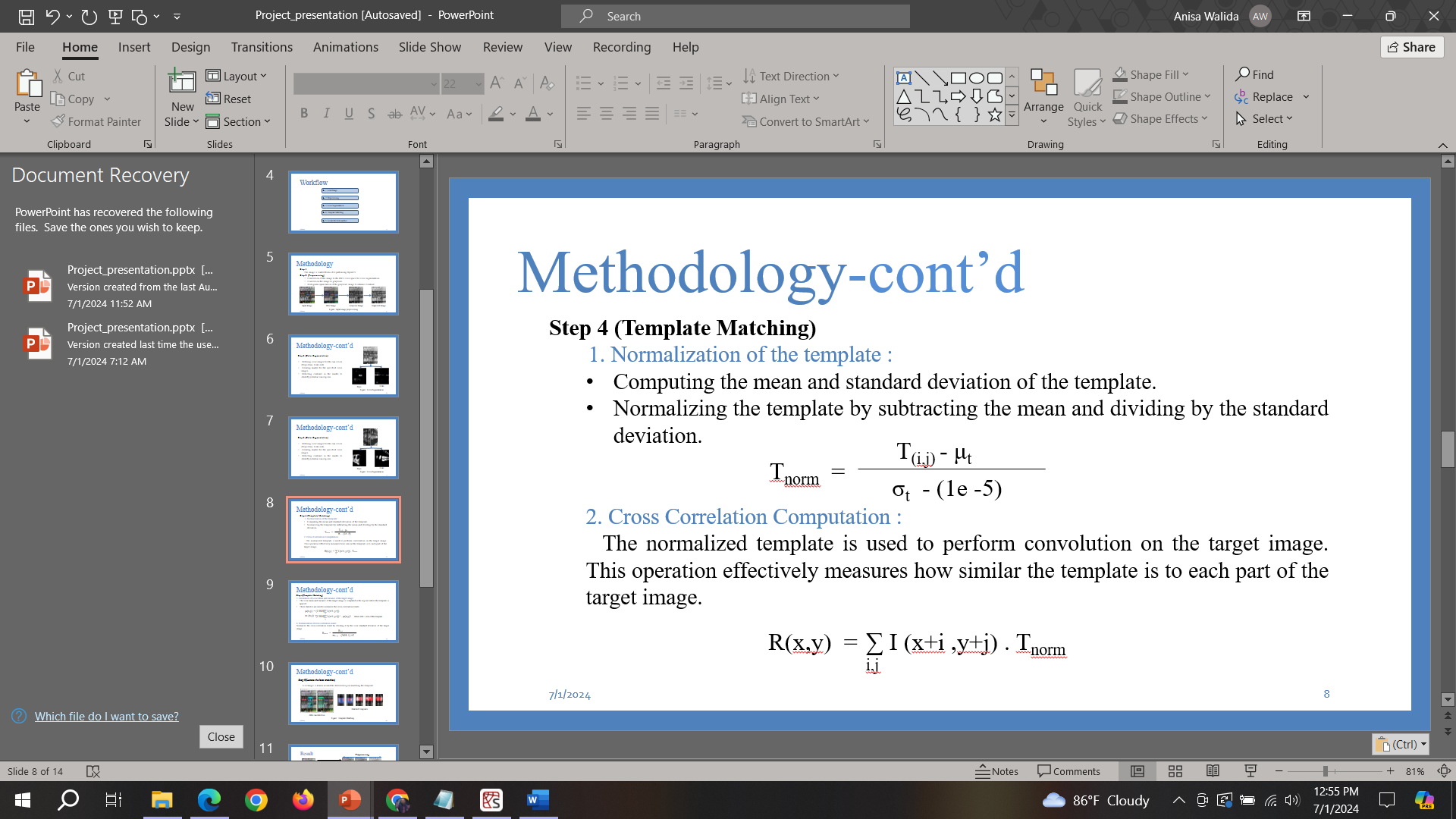
1. **Apply Template Matching Using Normalized Cross-Correlation:**

**Loading and Preprocessing Templates:** Template images for Pepsi and Coke cans are loaded and converted to grayscale if they are not already.

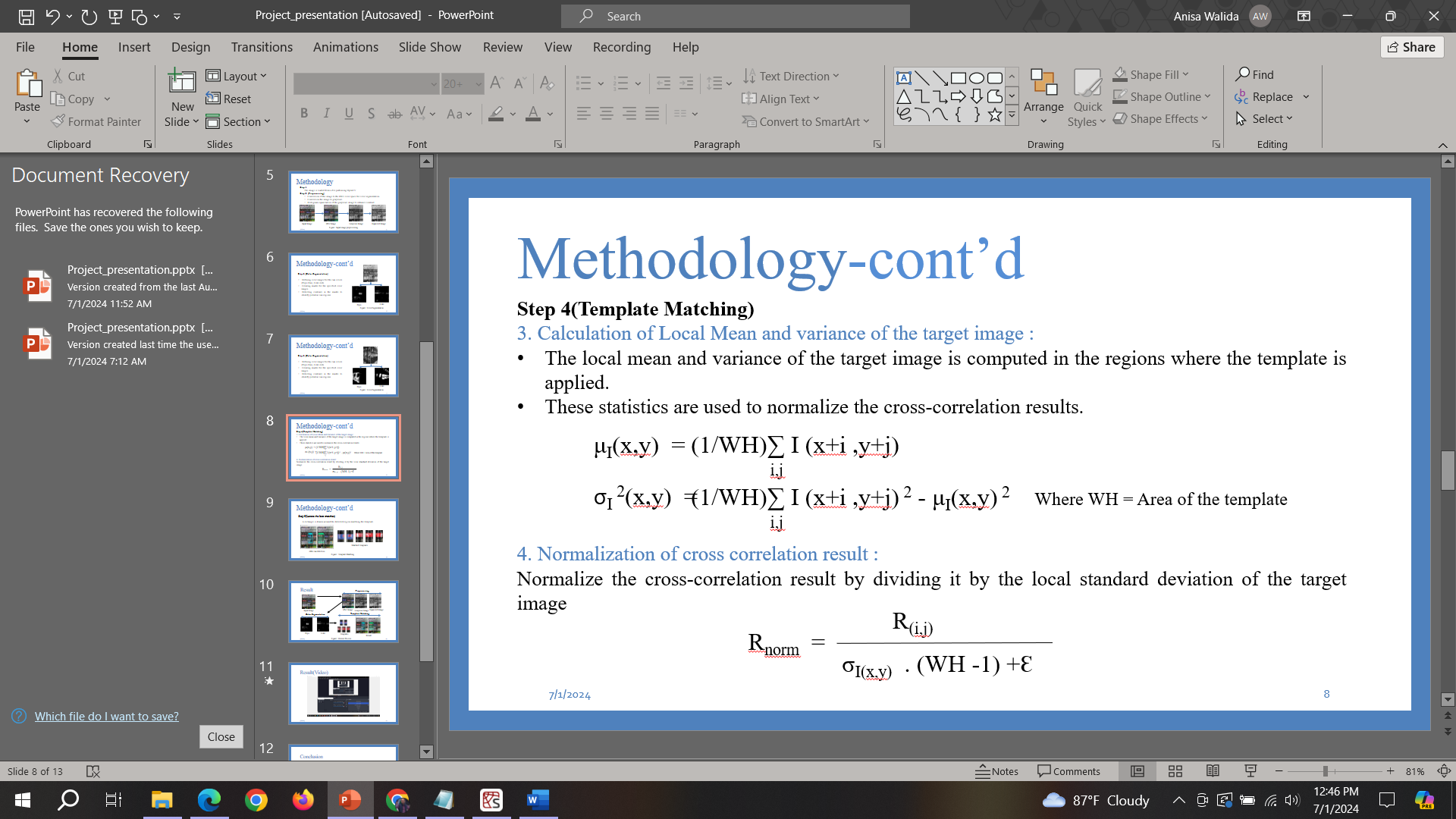
**Normalizing Templates:** The templates are normalized by subtracting the mean and dividing by the standard deviation. The normalization formula is:

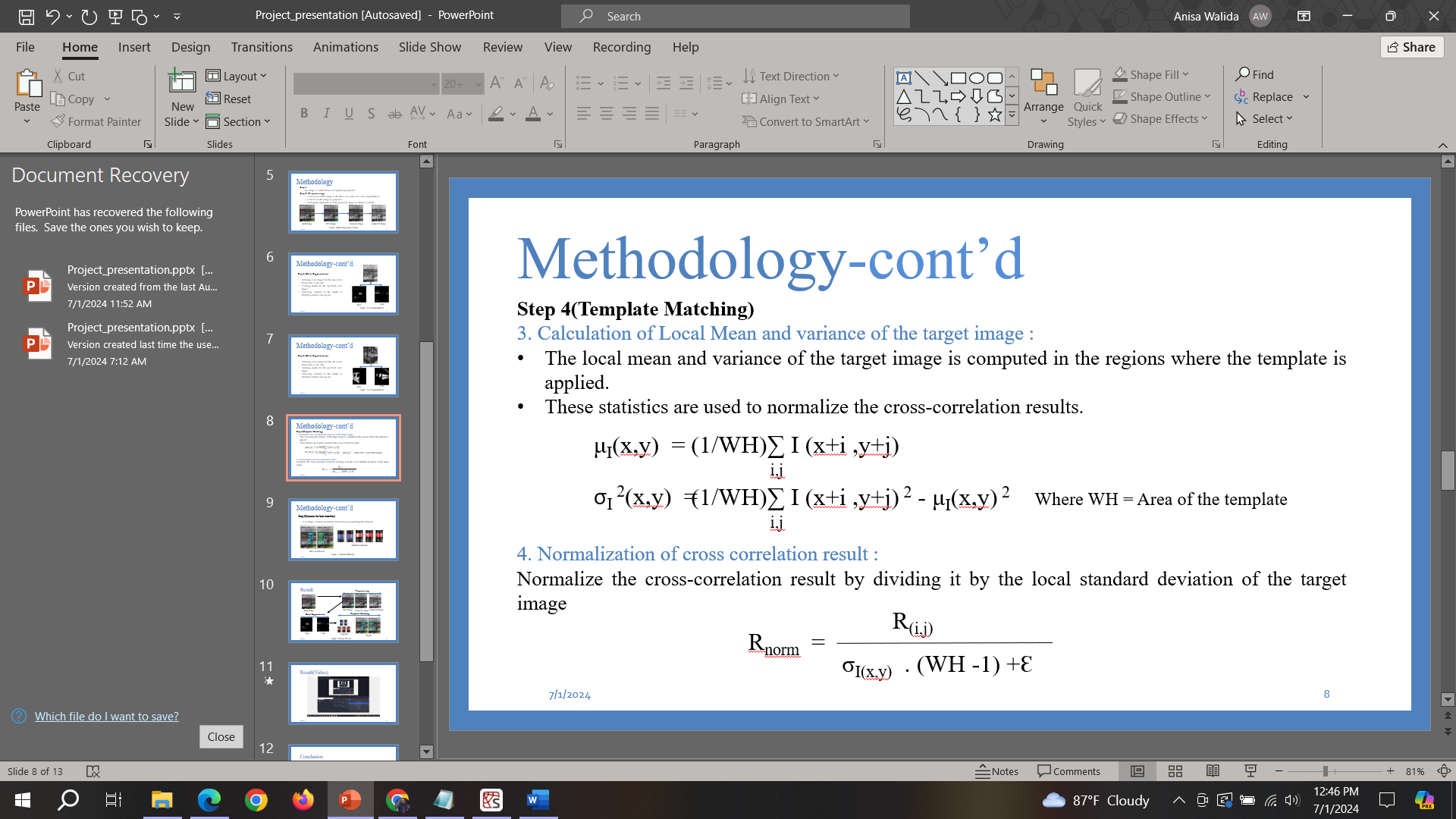


**Performing Convolution:** Convolution is performed to calculate the cross-correlation between the normalized template and the target image. The formula for the convolution is:



**Normalizing Cross-Correlation Result:** The cross-correlation result is normalized by dividing it by the local mean and variance of the target image to account for intensity variations. The normalization formula is:





**Identifying Regions Above Threshold:** Regions in the target image where the normalized cross-correlation result exceeds a predefined threshold are identified as potential matches. These regions correspond to locations where the template closely matches the image.

## Implementation

import cv2

import numpy as np

import os

from tkinter import filedialog, Tk, Label, Button, Frame, messagebox, Toplevel

from PIL import Image, ImageTk

def match\_template(image, template):

if image.ndim == 3 and template.ndim == 2:

# Convert image to grayscale if it's color and template is grayscale

image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

if image.dtype != np.float32:

# Convert image to float32 for better accuracy

image = image.astype(np.float32)

if template.dtype != np.float32:

# Convert template to float32 for better accuracy

template = template.astype(np.float32)

# Calculate result dimensions

result\_h, result\_w = image.shape[0] - template.shape[0] + 1, image.shape[1] - template.shape[1] + 1

if result\_h <= 0 or result\_w <= 0:

raise ValueError("Template size is larger than the image size.")

result = np.zeros((result\_h, result\_w), dtype=np.float32)

# Perform template matching using cv2.TM\_CCOEFF\_NORMED

template\_mean = np.mean(template)

template\_std = np.std(template)

template = (template - template\_mean) / (template\_std + 1e-5)

result = cv2.filter2D(image, -1, template)

image\_sqmean = cv2.filter2D(image\*\*2, -1, np.ones(template.shape) / np.prod(template.shape))

image\_mean = cv2.filter2D(image, -1, np.ones(template.shape) / np.prod(template.shape))

image\_var = image\_sqmean - image\_mean\*\*2

epsilon = 1e-10

image\_var = np.maximum(image\_var, 0) # Ensure non-negative variance

image\_std = np.sqrt(image\_var + epsilon)

result /= (image\_std \* (np.prod(template.shape) - 1) + epsilon)

return result

class CanDetector:

def \_\_init\_\_(self, image\_path):

self.image = cv2.imread(image\_path)

if self.image is None:

raise ValueError("Image not loaded correctly.")

self.hsv\_image = cv2.cvtColor(self.image, cv2.COLOR\_BGR2HSV)

self.gray\_image = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)

self.equalized\_image = cv2.equalizeHist(self.gray\_image)

def detect\_can(self, lower\_color, upper\_color):

mask = cv2.inRange(self.hsv\_image, lower\_color, upper\_color)

contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

return contours, mask

def apply\_template\_matching(self, template\_folder, can\_type, threshold=0.7,scale\_factor=2,rect\_color=(0, 255, 0)):

matched\_templates = []

for template\_name in os.listdir(template\_folder):

template\_path = os.path.join(template\_folder, template\_name)

template = cv2.imread(template\_path, 0)

if template is None:

continue

template = cv2.equalizeHist(template)

w, h = template.shape[::-1]

try:

res = match\_template(self.equalized\_image, template)

loc = np.where(res >= threshold)

for pt in zip(\*loc[::-1]):

expanded\_w = int(w \* scale\_factor)

expanded\_h = int(h \* scale\_factor)

x\_start = max(0, pt[0] - (expanded\_w - w) // 2)

y\_start = max(0, pt[1] - (expanded\_h - h) // 2)

cv2.rectangle(self.image, (x\_start, y\_start), (x\_start + expanded\_w, y\_start + expanded\_h), rect\_color, 2)

cv2.putText(self.image, can\_type, (x\_start, y\_start - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, rect\_color, 2)

if template\_path not in matched\_templates:

matched\_templates.append(template\_path)

except Exception as e:

print(f"Error matching template {template\_path}: {e}")

return self.image, matched\_templates

class PepsiDetector(CanDetector):

def \_\_init\_\_(self, image\_path, template\_folder):

super().\_\_init\_\_(image\_path)

self.template\_folder = template\_folder

def detect(self):

lower\_blue = np.array([110, 50, 50])

upper\_blue = np.array([130, 255, 255])

contours, mask = self.detect\_can(lower\_blue, upper\_blue)

if contours:

return self.apply\_template\_matching(self.template\_folder, "Pepsi", rect\_color=(249, 236, 6)), mask

return self.image, [], mask

class CokeDetector(CanDetector):

def \_\_init\_\_(self, image\_path, template\_folder):

super().\_\_init\_\_(image\_path)

self.template\_folder = template\_folder

def detect(self):

lower\_red1 = np.array([0, 120, 70])

upper\_red1 = np.array([10, 255, 255])

lower\_red2 = np.array([170, 120, 70])

upper\_red2 = np.array([180, 255, 255])

contours1, mask1 = self.detect\_can(lower\_red1, upper\_red1)

contours2, mask2 = self.detect\_can(lower\_red2, upper\_red2)

if contours1 or contours2:

mask = cv2.bitwise\_or(mask1, mask2)

return self.apply\_template\_matching(self.template\_folder, "Coke", rect\_color=(152, 244, 20)), mask

return self.image, [], mask

def select\_image():

file\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png")])

if file\_path:

status\_label.config(text="Processing...")

root.update()

try:

pepsi\_detector = PepsiDetector(file\_path, 'C://Users//USER//Downloads//Image Project Data//Pepsi\_logo')

coke\_detector = CokeDetector(file\_path, 'C://Users//USER//Downloads//Image Project Data//Coke\_logo')

(pepsi\_result, pepsi\_templates), pepsi\_mask = pepsi\_detector.detect()

(coke\_result, coke\_templates), coke\_mask = coke\_detector.detect()

display\_intermediate\_results(pepsi\_detector, coke\_detector, pepsi\_mask, coke\_mask, pepsi\_templates, coke\_templates)

status\_label.config(text="Detection completed.")

except Exception as e:

messagebox.showerror("Error", str(e))

status\_label.config(text="An error occurred. Try again.")

def display\_intermediate\_results(pepsi\_detector, coke\_detector, pepsi\_mask, coke\_mask, pepsi\_templates, coke\_templates):

for widget in main\_frame.winfo\_children():

if isinstance(widget, Label) and widget != status\_label and widget != open\_button:

widget.destroy()

display\_image(main\_frame, pepsi\_mask, "Color Segmentation Pepsi", 2, 0)

display\_image(main\_frame, coke\_mask, "Color Segmentation Coke", 2, 1)

display\_image(main\_frame, pepsi\_detector.equalized\_image, "Equalized Pepsi Image", 1, 0)

display\_image(main\_frame, coke\_detector.equalized\_image, "Equalized Coke Image", 1, 1)

display\_templates(main\_frame, pepsi\_templates, "Matched Templates - Pepsi", 1, 2)

display\_templates(main\_frame, coke\_templates, "Matched Templates - Coke", 2, 2)

display\_final\_result(pepsi\_detector.image, coke\_detector.image)

def display\_image(window, image, title, row, col, colspan=1):

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) if len(image.shape) == 3 else image

desired\_size = (250, 300)

image\_pil = Image.fromarray(image\_rgb).resize(desired\_size, Image.LANCZOS)

image\_photo = ImageTk.PhotoImage(image=image\_pil)

image\_label = Label(window, image=image\_photo, text=title, compound="top")

image\_label.image = image\_photo

image\_label.grid(row=row, column=col, columnspan=colspan, padx=10, pady=10, sticky="nsew")

def display\_image2(window, image, title, row, col, colspan=1):

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) if len(image.shape) == 3 else image

desired\_size = (450, 600)

image\_pil = Image.fromarray(image\_rgb).resize(desired\_size, Image.LANCZOS)

image\_photo = ImageTk.PhotoImage(image=image\_pil)

image\_label = Label(window, image=image\_photo, text=title, compound="top")

image\_label.image = image\_photo

image\_label.grid(row=row, column=col, columnspan=colspan, padx=10, pady=10, sticky="nsew")

def display\_templates(window, templates, title, row, col):

for i, template\_path in enumerate(templates):

template\_image = Image.open(template\_path).resize((150, 150), Image.LANCZOS)

template\_photo = ImageTk.PhotoImage(image=template\_image)

template\_label = Label(window, image=template\_photo, text=title, compound="top")

template\_label.image = template\_photo

template\_label.grid(row=row, column=col + i, padx=10, pady=10, sticky="nsew")

def display\_final\_result(pepsi\_image, coke\_image):

result\_window = Toplevel(root)

result\_window.title("Final Can Detection Result")

display\_image2(result\_window, pepsi\_image, "Final Detected Pepsi Cans", 0, 0)

display\_image2(result\_window, coke\_image, "Final Detected Coke Cans", 0, 1)

root = Tk()

root.title("Can Detector UI")

main\_frame = Frame(root)

main\_frame.pack(pady=20, padx=20)

open\_button = Button(main\_frame, text="Open Image", command=select\_image)

open\_button.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

status\_label = Label(main\_frame, text="Please select an image to detect cans.", relief="sunken")

status\_label.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

root.mainloop()

## Discussion

The can detection system successfully identifies and marks Pepsi and Coke cans in images. The intermediate results, including the original image, HSV image, grayscale image, color segmentation masks, histogram equalization results, and matched templates, are displayed to provide insight into each processing step. The final detected cans are displayed in a separate window for verification. Some issues were faced while implementing this. Such as the image size needed to be resized again and again so that no template in the dataset is bigger than the input image. The quality of the templates had to be good otherwise the can detection system was getting corrupted. The creation of the template matching function was the biggest challenge while creating this system.

## Conclusion

This project demonstrates the effectiveness of using computer vision techniques, such as color segmentation and template matching, for automated can detection. The system enhances efficiency and accuracy in industrial applications, offering a solution for various product detection tasks. The implementation showcases the practical application of advanced image processing methods, contributing to improved quality control and reduced manual inspection efforts.

## References

* <https://docs.opencv.org>
* [Template matching using OpenCV in Python - GeeksforGeeks](https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/)
* [Product Documentation - NI](https://www.ni.com/docs/en-US/bundle/ni-vision-concepts-help/page/color_segmentation.html)