# LAB ASSINMENT 2

by: Shahad Alshamrani 2111855, Lara Alofi 2110886

## Code:

```
 # Lab 2 Lara and Shahad

from PIL import Image # Import the Image module from the PIL library for image processing

import cv2 # Import the OpenCV library for image processing

import numpy as np # Import the NumPy library for numerical operations

from imwatermark import WatermarkEncoder # Import WatermarkEncoder from imwatermark module for watermark encoding

from imwatermark import WatermarkDecoder# Import WatermarkDecoder from imwatermark module for watermark decoding


# Watermark encoding

bgr = cv2.imread('lab2.png') # Read the input image in BGR format using OpenCV

wm = 'lab2' # Watermark text


encoder = WatermarkEncoder() # Create a WatermarkEncoder object

encoder.set_watermark('bytes', wm.encode('utf-8')) # Set the watermark text

bgr_encoded = encoder.encode(bgr, 'dwtDct') # Encode the watermark onto the image


cv2.imwrite('Watermarked.png', bgr_encoded) # Save the watermarked image

image = Image.open("Watermarked.png") # Open the resized image

print('image size=', image.size, ', image format=', image.format) # Print the size and format of the image
```

```
# Watermark decoding

bgr = cv2.imread('Watermarked.png') # Read the watermarked image

decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object

watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image


decoded_message = watermark.decode('utf-8') # Decode the watermark message


# Check if the decoded watermark matches the expected watermark

if decoded_message == wm:

 print("Watermark decoding passed.") # Print message indicating decoding success

 print("Expected Watermark:", wm) # Print the expected watermark message

 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message

else:

 print("Watermark decoding failed.") # Print message indicating decoding failure

 print("Expected Watermark:", wm) # Print the expected watermark message

 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty


#Attack1: Crop operation

print("\nAttack1: crop operation")

image = cv2.imread("Watermarked.png") # Read the input image

cropped = image[65:292, 37:564] # Crop the image using specified coordinates

cv2.imwrite("cropped.png", cropped) # Save the cropped image


# Watermark decoding

bgr = cv2.imread('cropped.png') # Read the watermarked image

decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object

watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image


decoded_message = watermark.decode('utf-8') # Decode the watermark message
```

```
# Check if the decoded watermark matches the expected watermark
if decoded_message == wm:
 print("Watermark decoding after cropping passed.") # Print message indicating decoding success
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message
else:
 print("Watermark decoding after cropping failed.") # Print message indicating decoding failure
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty

#Attack2 Rotation
print("\nAttack2: Rotation")
ROTATION = 60 # Define the rotation angle
image4 = Image.open("Watermarked.png") # Open the image
image4 = image4.rotate(ROTATION, expand=True) # Rotate the image
output_file = "Rotation.png" # Define the output file name
image4.save(output_file) # Save the rotated image

# Watermark decoding
bgr = cv2.imread('Rotation.png') # Read the watermarked image
decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object
watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image

decoded_message = watermark.decode('utf-8') # Decode the watermark message

# Check if the decoded watermark matches the expected watermark
if decoded_message == wm:
 print("Watermark decoding after Rotation passed.") # Print message indicating decoding success
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message
```

```
else:
 print("Watermark decoding after Rotation failed.") # Print message indicating decoding failure
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty

#Attack3: Resize
print("\nAttack3: Resize")
image3 = Image.open("Watermarked.png") # Open the image
DESIRED_WIDTH = 600 # Define the desired width
height = 600 # Define the height
image3 = image3.resize((DESIRED_WIDTH, height)) # Resize the image
output_file = "Resizing.png" # Define the output file name
image3.save(output_file) # Save the resized image
image = Image.open("Resizing.png") # Open the resized image
print('image size=', image.size, ', image format=', image.format) # Print the size and format of the image

# Watermark decoding
bgr = cv2.imread('Resizing.png') # Read the watermarked image
decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object
watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image

decoded_message = watermark.decode('utf-8') # Decode the watermark message

# Check if the decoded watermark matches the expected watermark
if decoded_message == wm:
 print("Watermark decoding after Resize passed.") # Print message indicating decoding success
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message
else:
 print("Watermark decoding after Resize failed.") # Print message indicating decoding failure
```

```
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty


#Attack4: Masking
print("\nAttack4: Masking")
image = cv2.imread('Watermarked.png') # Read the input image
mask = np.zeros(image.shape, dtype=np.uint8) # Create a binary mask
mask = cv2.circle(mask, (260, 300), 225, (255, 255, 255), -1) # Draw a white circle on the mask
result = cv2.bitwise_and(image, mask) # Apply the mask to the input image
result[mask == 0] = 255 # Color the background white
cv2.imwrite('Masking.png', result) # Save the masked image


# Watermark decoding
bgr = cv2.imread('Masking.png') # Read the watermarked image
decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object
watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image


decoded_message = watermark.decode('utf-8') # Decode the watermark message


# Check if the decoded watermark matches the expected watermark
if decoded_message == wm:
 print("Watermark decoding after Masking passed.") # Print message indicating decoding success
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message
else:
 print("Watermark decoding after Masking failed.") # Print message indicating decoding failure
 print("Expected Watermark:", wm) # Print the expected watermark message
 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty
```

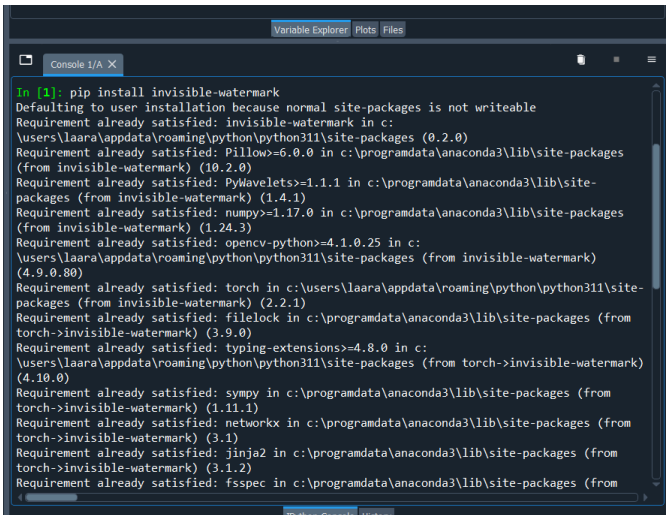
```
#Attack5: Overlaying

print("\nAttack5: Overlaying")

# Watermark encoding

bgr = cv2.imread('background.png') # Read the input image in BGR format using OpenCV

wm = 'lab2' # Watermark text


encoder = WatermarkEncoder() # Create a WatermarkEncoder object

encoder.set_watermark('bytes', wm.encode('utf-8')) # Set the watermark text

bgr_encoded = encoder.encode(bgr, 'dwtDct') # Encode the watermark onto the image

cv2.imwrite('backgroundWatermarked.png', bgr_encoded) # Save the watermarked image

img1 = Image.open(r"backgroundWatermarked.png") # Open the watermark background image

img2 = Image.open(r"lab2.png") # Open the overlay image

img1.paste(img2, (0, 0), mask=img2) # Paste the overlay image on top of the background image

img1.save("Overlaying.png") # Save the altered image


# Watermark decoding

bgr = cv2.imread('Overlaying.png') # Read the watermarked image

decoder = WatermarkDecoder('bytes', 32) # Create a WatermarkDecoder object

watermark = decoder.decode(bgr, 'dwtDct') # Decode the watermark from the image

decoded_message = watermark.decode('utf-8') # Decode the watermark message


# Check if the decoded watermark matches the expected watermark

if decoded_message == wm:

 print("Watermark decoding after overlaying passed.") # Print message indicating decoding success

 print("Expected Watermark:", wm) # Print the expected watermark message

 print("Decoded Watermark:", decoded_message) # Print the decoded watermark message

else:

 print("Watermark decoding after overlaying failed.") # Print message indicating decoding failure

 print("Expected Watermark:", wm) # Print the expected watermark message

 print("Decoded Watermark: is empty.") # Print a message indicating that the decoded watermark is empty
```

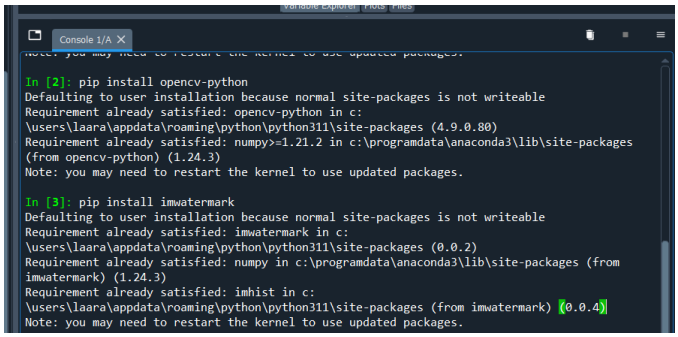
# Install Libary:

pip install invisible-watermark

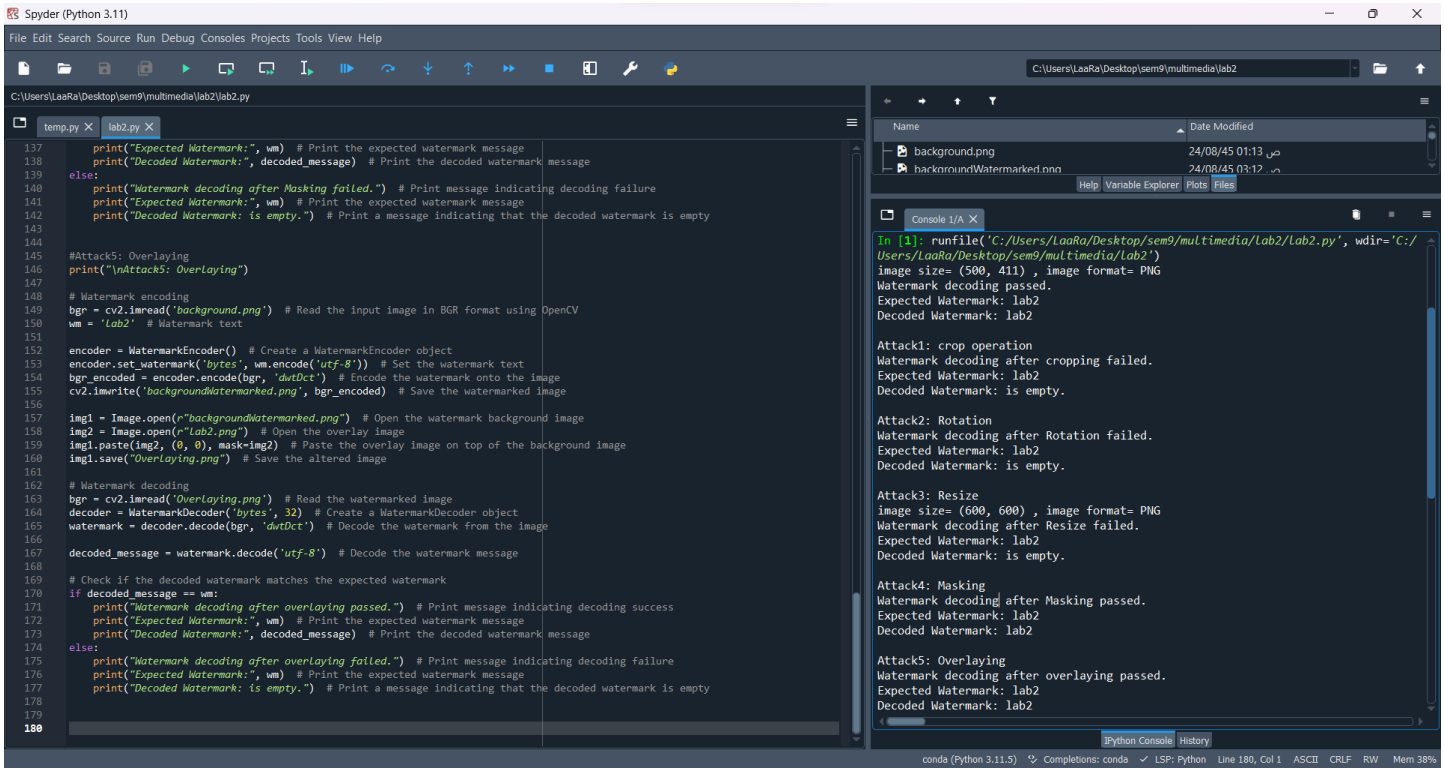
pip install opencv-python

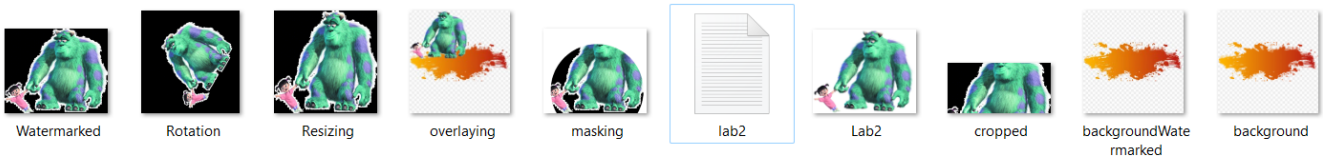
pip install imwatermark



# Output of Code:



# Description of the Watermarking Algorithm used

## Watermark Encoding:

**1- Image Input:** The algorithm begins by reading the input image lab2.png using the OpenCV library, which reads the image in BGR format (cv2.imread()).

**2- Watermark Text:** The watermark text specified as "lab2" is encoded as bytes using UTF-8 encoding.[1]

**3- Watermark Encoding Process:** An instance is initialized from the imwatermark library, facilitating the encoding process. Using the set_watermark() method, the watermark text is assigned after being converted into bytes through UTF-8 encoding. Subsequently, the encode() method of the encoder object is invoked to embed the watermark onto the input image, employing the Discrete Wavelet Transform and Discrete Cosine Transform ('dwtDct') techniques.

**4- Saving Watermarked Image:** The watermarked image is saved as Watermarked.png using cv2.imwrite().

## Watermark Decoding:

**1- Watermarked Image Reading:** The watermarked image (Watermarked.png) is read again using OpenCV (cv2.imread()).

**2- Watermark Decoding Process:** procedure involves initializing a WatermarkDecoder instance for the decoding process. Utilizing the decode() method of the decoder object, the watermark is extracted from the watermarked image employing the identical encoding method ('dwtDct'). Following extraction, the watermark, initially in byte format, undergoes conversion back to a UTF-8 string.

**3- Verification:** includes comparing the extracted watermark with the original watermark text ('lab2') to confirm the accuracy of the decoding process. Upon matching the decoded watermark with the expected watermark, a success message is displayed; conversely, a failure message is printed if no match is found.

## Attacks:

**Crop Operation:** Part of the watermarked image is cropped and the decoding of the watermark is attempted to detect the cropping effect.[2]
**Rotate:** The watermark image is rotated 60 degrees using PIL's rotate() method, then watermark decoding is performed.[3]
**Resize:** The watermark image is resized to 600 pixels and decoded to evaluate the resizing effects.[4]
**Masking:** A circular binary mask is applied to the watermark image, and decoding is attempted to examine the effects of the mask.[5]
**Overlaying:** An overlay image (lab2.png) is applied to the background image (background.png).[6]

**Each** attack tests the algorithm's robustness against common image manipulations, which helps evaluate its effectiveness in real-world scenarios.

## Testing with attacks:

**1. Image Cropping:**

- We are trying to crop part of the image to see if we can still get our hidden message back. It's like cropping a photo on your phone.

**2. Rotate:**

- We rotate the image as if we were turning the photo sideways.

**3. Making it Bigger or Smaller (Resize):**

- We are trying to zoom in or out of the image. Will our hidden message survive these changes?

**4. Covering Part of the Picture (Masking)):**

- Imagine putting a mask on part of a photo. Can we still see our hidden message?

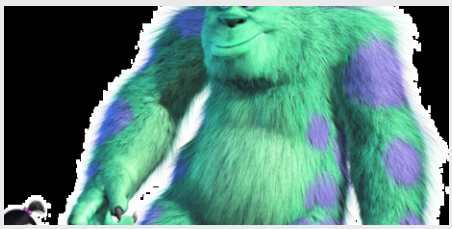
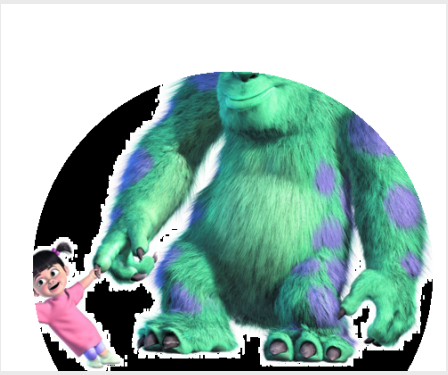
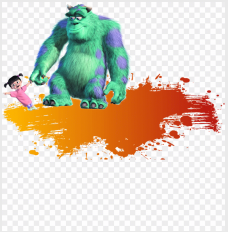
**5. Putting Pictures Together (Overlaying):**

- We stick two pictures together, one with our hidden message. Can we separate them and still find our secret code?

# Ataack Performance

**Watermarked Image**

**backgroundWatermarked Image**

| Attacks | image | Result |
| --- | --- | --- |
| *Crop operation[2]* |  | **failed** |
| Rotate 60 degress[3] |  | **failed** |
| Resize[4] |  | **failed** |
| Masking[5] |  | **passed** |
| Overlaying[6] |  | **passed** |

# References


**Watermark**:

- [1] imwatermark. (2023). invisible-watermark. [Online]. Available: https://pypi.org/project/invisible-watermark/

**Crop**:

- [2] Stack Overflow. (2021). I am using this code to crop an image using OpenCV, but this error keeps coming. [Online]. Available: https://stackoverflow.com/questions/65696961/i-am-using-this-code-to-crop-an-image-using-open-cv-but-this-error-keeps-coming

**Rotation**:

- [3] Stack Overflow. (2019). PIL image rotation and background issue. [Online]. Available: https://stackoverflow.com/questions/57833276/pil-image-rotation-and-background-issue

**Resizing**:

- [4] Cloudinary. (n.d.). Python Image Resize with Pillow and OpenCV. [Online]. Available: https://cloudinary.com/guides/bulk-image-resize/python-image-resize-with-pillow-and-opencv

**Mask**:

- [5] Stack Overflow. (2020). How to mask image with binary mask. [Online]. Available: https://stackoverflow.com/questions/59432324/how-to-mask-image-with-binary-mask

**Overlaying**:

- [6] GeeksforGeeks. (n.d.). Overlay an Image on Another Image in Python. [Online]. Available: https://www.geeksforgeeks.org/overlay-an-image-on-another-image-in-python/