CSE 252A Computer Vision I Fall 2021 - Assignment 0

Instructor: Ben Ochoa

- Assignment Posted: **Wed, September 29, 2021**
- Due Date: **Wed, October 6, 2021 11:59 PM (Pacific Time)**

Instructions

Please answer the questions below using Markdown or MathJax, and submit your PDF to Gradescope.

- This assignment consists of **two parts**: **Numpy** (for vector, matrix, array operations) and **Pillow** (for basic image I/O and manipulation).
- Write your code in the cells provided in this Jupyter notebook.
- After finishing the assignment in the notebook, please export the notebook as a PDF and submit both the notebook and the PDF on Gradescope. Make sure the PDF is readable.
- You may use standard packages (e.g., `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problem. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.

Late Policy: Assignments submitted late will receive a 10% grade deduction for each 24 hour late (i.e., 10% per day). Assignments will not be accepted 5 days (120 hours) after the deadline. If you have a valid reason for requesting an extension (e.g., illness, travel for job interviews, etc.), please contact the instructor and teaching assistants beforehand (such as when the due date is still one or two days from now).

Introduction

Welcome to **CSE252A Computer Vision I**!

This course provides a comprehensive introduction to computer vision providing broad coverage including low-level vision (image formation, photometry, color, image feature detection), filtering, 3D geometry from images (stereo, shape from shading, motion, scene reconstruction), and high-level vision (recognition).

We will use a variety of tools (e.g., some packages and assemblers in the case of low-level vision, deep learning toolkits and/or configurations. To ensure consistent progress, we will keep the majority of the tools to be used in the course in the **Assignment 0**. You will also provide some basic implementations.

Piazza, Gradescope and Python

Piazza

Announcements are automatically added to the class Piazza page enrolled in the class. You can get access to Piazza here. You'll be added to the Gradescope. We will post course materials questions on Piazza. Class announcements will be made using Piazza. In class, you can chat with your classmates using Piazza frequently.

Gradescope

We will use Gradescope for all assignments and for the midterm and final exam. You can also get access to Gradescope from Canvas. All the assignments will be posted on Gradescope. We will post lectures materials on Piazza. Make sure you register for Gradescope.

Python

We will use the Python programming language for all assignments in this course, with a few popular libraries (`NumPy`, `Matplotlib`). Anaconda is a convenient way to install Python and its numerical libraries on your machine. We will use Python 3.x in this course. Numerical operations will mainly use `NumPy` and `SciPy`. And course-specific binaries/packages (`BOW`, `vlfeat`) will be `Pillow` (`PIL`) and `OpenCV`. Please read the tutorials and learn how to use Python and its numerical libraries.

Getting Started with NumPy

`NumPy` is the fundamental package for scientific computing with Python. It provides a powerful N-dimensional array object and functions for working with these arrays. It also has functions for linear algebra, Fourier transforms, and random number capabilities. Highly recommendated to the Python programmers to get familiar with `NumPy` and use it extensively for coding assignments.

Lists

```
import time, sys, os
```

```
# Create a list of integers
lst = [1, 2, 3, 4, 5]
print("lst: ", lst)

# Create a list of strings
lst = ["a", "b", "c", "d", "e"]
print("lst: ", lst)

# Create a list of mixed data types
lst = [1, 2.5, "a", True, [1, 2, 3]]
print("lst: ", lst)

# Accessing elements in the list
print("lst[0]: ", lst[0])
print("lst[-1]: ", lst[-1])

# Slicing a list
print("lst[1:3]: ", lst[1:3])
print("lst[:3]: ", lst[:3])
print("lst[3:]: ", lst[3:])

# Modifying elements in the list
lst[0] = 10
print("lst: ", lst)

# Adding elements to the list
lst.append("f")
print("lst: ", lst)

# Removing elements from the list
del lst[2]
print("lst: ", lst)

# Length of the list
print("len(lst): ", len(lst))

# Checking if an element is in the list
print("'a' in lst: ", 'a' in lst)

# Concatenating two lists
lst2 = [6, 7, 8]
print("lst + lst2: ", lst + lst2)

# Repeating a list
print("lst * 2: ", lst * 2)
```

```
lst:  [1, 2, 3, 4, 5]
lst:  ['a', 'b', 'c', 'd', 'e']
lst:  [1, 2.5, 'a', True, [1, 2, 3]]
lst[0]:  1
lst[-1]:  [1, 2, 3]
lst[1:3]:  [2.5, 'a']
lst[:3]:  [1, 2.5, 'a']
lst[3:]:  [True, [1, 2, 3]]
lst:  [10, 2.5, 'a', True, [1, 2, 3]]
lst:  [10, 2.5, 'a', True, [1, 2, 3], 'f']
lst:  [10, 2.5, True, [1, 2, 3], 'f']
len(lst):  5
'a' in lst:  False
lst + lst2:  [10, 2.5, True, [1, 2, 3], 'f', 6, 7, 8]
lst * 2:  [10, 2.5, True, [1, 2, 3], 'f', 10, 2.5, True, [1, 2, 3], 'f']
```

Arrays

```
import time, sys, os
```

```
# Create a 1D array
arr = np.array([1, 2, 3, 4, 5])
print("arr: ", arr)

# Create a 2D array
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("arr: ")
print(arr)

# Accessing elements in the array
print("arr[0, 0]: ", arr[0, 0])
print("arr[-1, -1]: ", arr[-1, -1])

# Slicing an array
print("arr[0:2, 0:2]: ")
print(arr[0:2, 0:2])
print("arr[:, 1]: ")
print(arr[:, 1])
print("arr[1, :]: ")
print(arr[1, :])

# Modifying elements in the array
arr[0, 0] = 10
print("arr: ")
print(arr)

# Adding elements to the array
arr = np.append(arr, [10, 11, 12])
print("arr: ")
print(arr)

# Removing elements from the array
arr = np.delete(arr, [0, 1, 2])
print("arr: ")
print(arr)

# Length of the array
print("len(arr): ", len(arr))

# Checking if an element is in the array
print("5 in arr: ", 5 in arr)

# Concatenating two arrays
arr2 = np.array([6, 7, 8])
print("arr + arr2: ", arr + arr2)

# Repeating an array
print("arr * 2: ", arr * 2)
```

```
arr:  [1, 2, 3, 4, 5]
arr:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
arr[0, 0]:  1
arr[-1, -1]:  9
arr[0:2, 0:2]:
[[1 2]
 [4 5]]
arr[:, 1]:
[2 5 8]
arr[1, :]:
[4 5 6]
arr:
[[10  2  3]
 [ 4  5  6]
 [ 7  8  9]]
arr:
[10  2  3  4  5  6  7  8  9 10 11 12]
arr:
[ 3  4  5  6  7  8  9 10 11 12]
len(arr):  10
5 in arr:  False
arr + arr2:  [ 9 10 11 13 15 17 19 21 23]
arr * 2:  [ 6  8 10 12 14 16 18 20 22 24]
```

Array Indexing

Array Dimension Operation

```
import time, sys, os
```

```
# Create a 1D array
arr = np.array([1, 2, 3, 4, 5])
print("arr: ", arr)

# Create a 2D array
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("arr: ")
print(arr)

# Reshaping an array
print("arr.reshape(9, 1): ")
print(arr.reshape(9, 1))

# Transposing an array
print("arr.T: ")
print(arr.T)

# Flattening an array
print("arr.flatten(): ")
print(arr.flatten())

# Ravel an array
print("arr.ravel(): ")
print(arr.ravel())
```

```
arr:  [1 2 3 4 5]
arr:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
arr.reshape(9, 1):
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
arr.T:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
arr.flatten():
[1 2 3 4 5 6 7 8 9]
arr.ravel():
[1 2 3 4 5 6 7 8 9]
```

Math Operations on Array

Element-wise Operation

```
import time, sys, os
```

```
# Create two 1D arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

# Element-wise addition
print("arr1 + arr2: ", arr1 + arr2)

# Element-wise subtraction
print("arr1 - arr2: ", arr1 - arr2)

# Element-wise multiplication
print("arr1 * arr2: ", arr1 * arr2)

# Element-wise division
print("arr1 / arr2: ", arr1 / arr2)

# Element-wise exponentiation
print("arr1 ** arr2: ", arr1 ** arr2)

# Element-wise modulo
print("arr1 % arr2: ", arr1 % arr2)

# Element-wise comparison
print("arr1 > arr2: ", arr1 > arr2)

# Element-wise logical AND
print("arr1 & arr2: ", arr1 & arr2)

# Element-wise logical OR
print("arr1 | arr2: ", arr1 | arr2)

# Element-wise logical NOT
print("~arr1: ", ~arr1)
```

```
arr1 + arr2:  [ 7 10 13 16 19]
arr1 - arr2:  [-5 -5 -5 -5 -5]
arr1 * arr2:  [ 6 14 24 36 50]
arr1 / arr2:  [0.16666667 0.28571429 0.37500000 0.44444444 0.50000000]
arr1 ** arr2:  [     1    128   6561  65536 390625]
arr1 % arr2:  [1 2 3 4 5]
arr1 > arr2:  [False False False False False]
arr1 & arr2:  [0 2 0 4 0]
arr1 | arr2:  [7 7 8 9 5]
~arr1:  [-2 -3 -4 -5 -6]
```

Vector and Matrix Operations

```
import time, sys, os
```

```
# Create two 1D arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Dot product
print("np.dot(arr1, arr2): ", np.dot(arr1, arr2))

# Outer product
print("np.outer(arr1, arr2): ", np.outer(arr1, arr2))

# Inner product
print("np.inner(arr1, arr2): ", np.inner(arr1, arr2))

# Cross product
print("np.cross(arr1, arr2): ", np.cross(arr1, arr2))

# Create two 2D arrays
arr3 = np.array([[1, 2], [3, 4]])
arr4 = np.array([[5, 6], [7, 8]])

# Matrix multiplication
print("np.matmul(arr3, arr4): ")
print(np.matmul(arr3, arr4))

# Alternative way to perform matrix multiplication
print("arr3 @ arr4: ")
print(arr3 @ arr4)
```

```
np.dot(arr1, arr2):  32
np.outer(arr1, arr2):  [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
np.inner(arr1, arr2):  32
np.cross(arr1, arr2):  [-3  6 -3]
np.matmul(arr3, arr4):
[[19 22]
 [43 50]]
arr3 @ arr4:
[[19 22]
 [43 50]]
```

Broadcasting

```
import numpy as np
import matplotlib.pyplot as plt, matplotlib.image as mpimg, os, urllib.request, ssl
```

```
# Create a 1D array
arr = np.array([1, 2, 3])

# Broadcasting scalar to the array
print("arr + 5: ", arr + 5)

# Broadcasting array to the array
print("arr + arr: ", arr + arr)
```

```
arr + 5:  [6 7 8]
arr + arr:  [2 4 6]
```

Sum and Mean

```
import time, sys, os
```

```
# Create a 2D array
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Sum of all elements
print("np.sum(arr): ", np.sum(arr))

# Sum of each row
print("np.sum(arr, axis=1): ", np.sum(arr, axis=1))

# Sum of each column
print("np.sum(arr, axis=0): ", np.sum(arr, axis=0))

# Mean of all elements
print("np.mean(arr): ", np.mean(arr))

# Mean of each row
print("np.mean(arr, axis=1): ", np.mean(arr, axis=1))

# Mean of each column
print("np.mean(arr, axis=0): ", np.mean(arr, axis=0))
```

```
np.sum(arr):  45
np.sum(arr, axis=1):  [ 6 15 24]
np.sum(arr, axis=0):  [12 15 18]
np.mean(arr):  5.0
np.mean(arr, axis=1):  [2. 5. 8.]
np.mean(arr, axis=0):  [4. 5. 6.]
```

Matplotlib

`Matplotlib` is a plotting library. We will use it to show the result in the assignment.

```
import numpy as np, matplotlib.pyplot as plt, os, urllib.request, ssl
%matplotlib inline
```

```
# Create a 1D array
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 4, 9, 16, 25])

# Plot the data
plt.plot(x, y)

# Add labels and title
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Simple Plot')

# Show the plot
plt.show()
```

The brief tutorials introduces many basic functions (`bar()`, `pie()`, and `hist()`) that can be fun to complete. Check out more operations and functions in the documentation and examples on the page.

The book `CONVEXITY.jpg` accompanies the image dataset this Caltech.

```
City-ABC        Top-Left Image
City-Right      Bottom-Left Image
Bottom-ABC      Top-Right Image
Bottom-Right    Bottom-Right Image
```

For the implementation:

1) Fix the last checked image, write your implementation to extract a single channel from a colored image. This means that from the original image, you should be able to extract the red channel, the green channel, and the blue channel.

2) Fix the BGR2GRAY function, write your implementation to change from a single image from BGR color to GRAY. Grayscale colored image is the received channel value B0.299, G0.587, R0.114.

3) Fix the grayscale image, write function to contrast adjustment with the provided intensity.

4) Fix the last function, write your implementation to extract a single channel from a colored image. This means that from the original image, you should be able to extract the red channel, the green channel, and the blue channel.

5) Fix the BGR2GRAY function, write your implementation to change from a single image from BGR color to GRAY. Grayscale colored image is the received channel value B0.299, G0.587, R0.114.

6) Fix the grayscale image, write function to contrast adjustment with the provided intensity.

```
import time, sys, os
```

```
# Download the image
urlretrieve('https://www.cs.ucsd.edu/~ben/CSE252A-fall-2021/CONVEXITY.jpg', 'CONVEXITY.jpg')

# Load the image
image = mpimg.imread('CONVEXITY.jpg')

# Display the image
plt.imshow(image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to extract a single channel from a colored image
def extract_channel(image, channel):
    # Extracting the specified channel
    if channel == 'red':
        return image[:, :, 0]
    elif channel == 'green':
        return image[:, :, 1]
    elif channel == 'blue':
        return image[:, :, 2]
    else:
        raise ValueError("Invalid channel. Choose 'red', 'green', or 'blue'.")

# Extract the red channel
red_channel = extract_channel(image, 'red')

# Display the red channel
plt.imshow(red_channel, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to convert BGR image to GRAY
def bgr2gray(image):
    # Extracting the channels
    b, g, r = image[:, :, 0], image[:, :, 1], image[:, :, 2]

    # Calculating the grayscale value using the weights
    gray = 0.299 * r + 0.587 * g + 0.114 * b

    return gray

# Convert the image to grayscale
gray_image = bgr2gray(image)

# Display the grayscale image
plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to adjust contrast of a grayscale image
def adjust_contrast(image, intensity):
    # Calculating the mean intensity
    mean = np.mean(image)

    # Adjusting contrast by scaling the difference between each pixel and the mean
    adjusted = (image - mean) * intensity + mean

    # Clipping values to ensure they are in the valid range [0, 255]
    adjusted = np.clip(adjusted, 0, 255)

    return adjusted

# Adjust contrast with intensity 1.5
adjusted_image = adjust_contrast(gray_image, 1.5)

# Display the adjusted image
plt.imshow(adjusted_image, cmap='gray')
plt.axis('off')
plt.show()
```


Now, write following extra functions to achieve the difference between binary and symmetrical operations.

```
import time, sys, os
```

```
# Function to apply a binary threshold to an image
def binary(image, threshold):
    # Applying the threshold
    binary = (image > threshold).astype(np.uint8) * 255

    return binary

# Apply binary threshold with threshold value 128
binary_image = binary(gray_image, 128)

# Display the binary image
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to apply symmetrical operation to an image
def symmetrical(image):
    # Flipping the image horizontally
    symmetrical = np.flip(image, axis=1)

    return symmetrical

# Apply symmetrical operation
symmetrical_image = symmetrical(gray_image)

# Display the symmetrical image
plt.imshow(symmetrical_image, cmap='gray')
plt.axis('off')
plt.show()
```


Problem 2: Image Operations and Vectorization (15 points)

In this problem you will use the image `CONVEXITY.jpg` from Problem 1 as the input image.

The problem needs an image, dual binarized original.jpg that you will find in the assignment folder. The functions are then provided as a starting point for your implementation of the image.

The book `CONVEXITY.jpg` accompanies the image dataset this Caltech.

```
City-ABC        Top-Left Image
City-Right      Bottom-Left Image
Bottom-ABC      Top-Right Image
Bottom-Right    Bottom-Right Image
```

For the dual binarized image, write your implementation to extract a single channel from a colored image. This means that from the original image, you should be able to extract the red channel, the green channel, and the blue channel.

2) Fix the BGR2GRAY function, write your implementation to change from a single image from BGR color to GRAY. Grayscale colored image is the received channel value B0.299, G0.587, R0.114.

3) Fix the grayscale image, write function to contrast adjustment with the provided intensity.

4) Fix the last function, write your implementation to extract a single channel from a colored image. This means that from the original image, you should be able to extract the red channel, the green channel, and the blue channel.

5) Fix the BGR2GRAY function, write your implementation to change from a single image from BGR color to GRAY. Grayscale colored image is the received channel value B0.299, G0.587, R0.114.

6) Fix the grayscale image, write function to contrast adjustment with the provided intensity.

```
import time, sys, os
```

```
# Download the image
urlretrieve('https://www.cs.ucsd.edu/~ben/CSE252A-fall-2021/dual_binarized_original.jpg', 'dual_binarized_original.jpg')

# Load the image
image = mpimg.imread('dual_binarized_original.jpg')

# Display the image
plt.imshow(image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to extract a single channel from a colored image
def extract_channel(image, channel):
    # Extracting the specified channel
    if channel == 'red':
        return image[:, :, 0]
    elif channel == 'green':
        return image[:, :, 1]
    elif channel == 'blue':
        return image[:, :, 2]
    else:
        raise ValueError("Invalid channel. Choose 'red', 'green', or 'blue'.")

# Extract the red channel
red_channel = extract_channel(image, 'red')

# Display the red channel
plt.imshow(red_channel, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to convert BGR image to GRAY
def bgr2gray(image):
    # Extracting the channels
    b, g, r = image[:, :, 0], image[:, :, 1], image[:, :, 2]

    # Calculating the grayscale value using the weights
    gray = 0.299 * r + 0.587 * g + 0.114 * b

    return gray

# Convert the image to grayscale
gray_image = bgr2gray(image)

# Display the grayscale image
plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to adjust contrast of a grayscale image
def adjust_contrast(image, intensity):
    # Calculating the mean intensity
    mean = np.mean(image)

    # Adjusting contrast by scaling the difference between each pixel and the mean
    adjusted = (image - mean) * intensity + mean

    # Clipping values to ensure they are in the valid range [0, 255]
    adjusted = np.clip(adjusted, 0, 255)

    return adjusted

# Adjust contrast with intensity 1.5
adjusted_image = adjust_contrast(gray_image, 1.5)

# Display the adjusted image
plt.imshow(adjusted_image, cmap='gray')
plt.axis('off')
plt.show()
```


Now write following extra functions to achieve the difference between binary and symmetrical operations.

```
import time, sys, os
```

```
# Function to apply a binary threshold to an image
def binary(image, threshold):
    # Applying the threshold
    binary = (image > threshold).astype(np.uint8) * 255

    return binary

# Apply binary threshold with threshold value 128
binary_image = binary(gray_image, 128)

# Display the binary image
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show
```



```
import time, sys, os
```

```
# Function to apply symmetrical operation to an image
def symmetrical(image):
    # Flipping the image horizontally
    symmetrical = np.flip(image, axis=1)

    return symmetrical

# Apply symmetrical operation
symmetrical_image = symmetrical(gray_image)

# Display the symmetrical image
plt.imshow(symmetrical_image, cmap='gray')
plt.axis('off')
plt.show
```


Problem 2: More Image Manipulation (35 points)

In this problem you will use the image `bird.jpeg`. Being a color image, this image has three channels, corresponding to the primary colors red, green, and blue.

1) Read the image

2) Write a function to flip the input image horizontally. For this function, please only use **Array Indexing** to implement this function. **do not** use NumPy and the functions like `np.flip()` that directly flip the arrays.

3) Write write another function to make the original image 90 degree counterclockwise. For this function, please only use **Array Indexing** to implement this function. **do not** use NumPy and the functions like `np.rot90()` that directly rotate the arrays. Do it twice.

4) Write a function to crop the original image and extract its 100 pixels wide square.

5) Write the `Total Gray.jpg` image and the corresponding `Total Gray Histogram.jpg` Grays total image

6) Write a function to change the original image to a gray scale image. Please use the formula: Gray = 0.299R + 0.587G + 0.114B. Now, you need to change the red, green, and blue channels to the same gray value.

7) Finally, consider **Frame Images** you obtained 1 original from Image 1 from Figuring Out to Achieve. 1 from Image 1 left degree, and 1 from Image 1 right degree. Please use the formula in 6) to change the three images to gray scale images. Then, please use the formula in 3) to change the gray scale images to the 3 channel images. Finally, stack the 3 images together and save it as `Frame.jpg`. The order is from the image on the left-most to the right-most. Show the final image.

```
import time, sys, os
```

```
# Download the image
urlretrieve('https://www.cs.ucsd.edu/~ben/CSE252A-fall-2021/bird.jpeg', 'bird.jpeg')

# Load the image
image = mpimg.imread('bird.jpeg')

# Display the image
plt.imshow(image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to flip the image horizontally
def flip_horizontal(image):
    # Initialize the flipped image array
    flipped = np.zeros_shape, dtype=image.dtype)

    # Iterate over the rows and columns
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Flipping the pixel horizontally
            flipped[i, j] = image[i, j]

    return flipped

# Flip the image horizontally
flipped_image = flip_horizontal(image)

# Display the flipped image
plt.imshow(flipped_image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to rotate the image 90 degrees counterclockwise
def rotate_90(image):
    # Initialize the rotated image array
    rotated = np.zeros(shape=(image.shape[1], image.shape[0], image.shape[2]), dtype=image.dtype)

    # Iterate over the rows and columns
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Rotating the pixel counterclockwise
            rotated[image.shape[1] - j - 1, i] = image[i, j]

    return rotated

# Rotate the image 90 degrees counterclockwise
rotated_image = rotate_90(image)

# Display the rotated image
plt.imshow(rotated_image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to crop the image to 100 pixels wide square
def crop(image, size):
    # Calculating the center coordinates
    center_x, center_y = image.shape[1] // 2, image.shape[0] // 2

    # Calculating the cropping boundaries
    left = center_x - size // 2
    right = center_x + size // 2
    top = center_y - size // 2
    bottom = center_y + size // 2

    # Cropping the image
    cropped = image[top:bottom, left:right]

    return cropped

# Crop the image to 100 pixels wide square
cropped_image = crop(image, 100)

# Display the cropped image
plt.imshow(cropped_image)
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to convert BGR image to GRAY
def bgr2gray(image):
    # Extracting the channels
    b, g, r = image[:, :, 0], image[:, :, 1], image[:, :, 2]

    # Calculating the grayscale value using the weights
    gray = 0.299 * r + 0.587 * g + 0.114 * b

    return gray

# Convert the image to grayscale
gray_image = bgr2gray(image)

# Display the grayscale image
plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to adjust contrast of a grayscale image
def adjust_contrast(image, intensity):
    # Calculating the mean intensity
    mean = np.mean(image)

    # Adjusting contrast by scaling the difference between each pixel and the mean
    adjusted = (image - mean) * intensity + mean

    # Clipping values to ensure they are in the valid range [0, 255]
    adjusted = np.clip(adjusted, 0, 255)

    return adjusted

# Adjust contrast with intensity 1.5
adjusted_image = adjust_contrast(gray_image, 1.5)

# Display the adjusted image
plt.imshow(adjusted_image, cmap='gray')
plt.axis('off')
plt.show()
```



```
import time, sys, os
```

```
# Function to apply a binary threshold to an image
def binary(image, threshold):
    # Applying the threshold
    binary = (image > threshold).astype(np.uint8) * 255

    return binary

# Apply binary threshold with threshold value 128
binary_image = binary(gray_image, 128)

# Display the binary image
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show
```



```
import time, sys, os
```

```
# Function to apply symmetrical operation to an image
def symmetrical(image):
    # Flipping the image horizontally
    symmetrical = np.flip(image, axis=1)

    return symmetrical

# Apply symmetrical operation
symmetrical_image = symmetrical(gray_image)

# Display the symmetrical image
plt.imshow(symmetrical_image, cmap='gray')
plt.axis('off')
plt.show
```

In [15]:
```
# if blank the text over image and the text each binary image
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# CONVERT TO GRAYSCALE
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# cv_gray_img = cv2.cvtColor(cv_text_blank_binary_image)
# ---------------------------------------------------------------
# ---------------------------------------------------------------
print("cv_text_blank_binary_image.shape", cv_text_blank_binary_image.shape)
print("cv_text_blank_binary_image.dtype", cv_text_blank_binary_image.dtype)
# ---------------------------------------------------------------
# ---------------------------------------------------------------
print("Text Blank Binary Image Shape", cv_text_blank_binary_image.shape, "\n")
print("Text Blank Binary Image dtype", cv_text_blank_binary_image.dtype)
# ---------------------------------------------------------------
# ---------------------------------------------------------------
plt.imshow(cv_text_blank_binary_image)
plt.show()
plt.imshow(cv_text_blank_binary_image, cmap="gray")
plt.show()
# ---------------------------------------------------------------
# ---------------------------------------------------------------
Text Blank Binary Image Shape
Text Blank Binary Image dtype
```

In [16]:
```
# if the text the text binary of the text's text
# ---------------------------------------------------------------
# ---------------------------------------------------------------
for cv_binary_text_image, text_ch, start_ch, start_y, start_x:
    # ---------------------------------------------------------------
    for y in range(start_y, start_y+h):
        # ---------------------------------------------------------------
        for x in range(start_x, start_x+w):
            # ---------------------------------------------------------------
            if cv_binary_text_image[y, x] == 255:
                # ---------------------------------------------------------------
                cv_binary_text_image[y, x] = 0
                # ---------------------------------------------------------------
                else:
                    # ---------------------------------------------------------------
                    cv_binary_text_image[y, x] = 255
                    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
    start_y = y
    start_x = x
    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
    cv_text_blank_binary_image[y, x] = cv_binary_text_image[y, x]
    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
    # ---------------------------------------------------------------
```

In [17]:
```
# if the text the text binary of the text's text
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# if cv_binary_text_image[y, x] == 255:
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# cv_binary_text_image[y, x] = 0
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# cv_text_blank_binary_image[y, x] = cv_binary_text_image[y, x]
# ---------------------------------------------------------------
# ---------------------------------------------------------------
# ---------------------------------------------------------------
```