Data

To complete this assignment, please pick 3 images of your liking. You should use:

1) A picture from any interesting landmark from your hometown. This can be a picture taken by you or from the internet. Please clearly indicate the source on your report.

2) A picture from any interesting landmark from Chicago. This must be a picture taken by you. If you are from Chicago, please pic a different picture from the first one.

3) A random picture of your liking different from the first two.

You will have to run your code with each of these images, and then you must display and briefly discuss the results in your report.

Part 1 – Getting familiar with image manipulation

Write code that will: (a) Read an image, convert it to grayscale if it is not already, and display the converted image (b) Calculate and report the size (total number of pixels) of the image. (c) Calculate and report the maximum pixel value. (d) Calculate and report the mean pixel value.

Change the pixel values of the image in the following way: all pixels' values less than the average calculated at (d) will be equal to 0 and all the others will be equal to 1. Display your binary image. Make sure your displayed image is black and white, not black and near black.

In addition, perform your thresholding on a color image, thresholding separately in each color channel. Combine the results into a single image for display. The resulting image should be a color image.

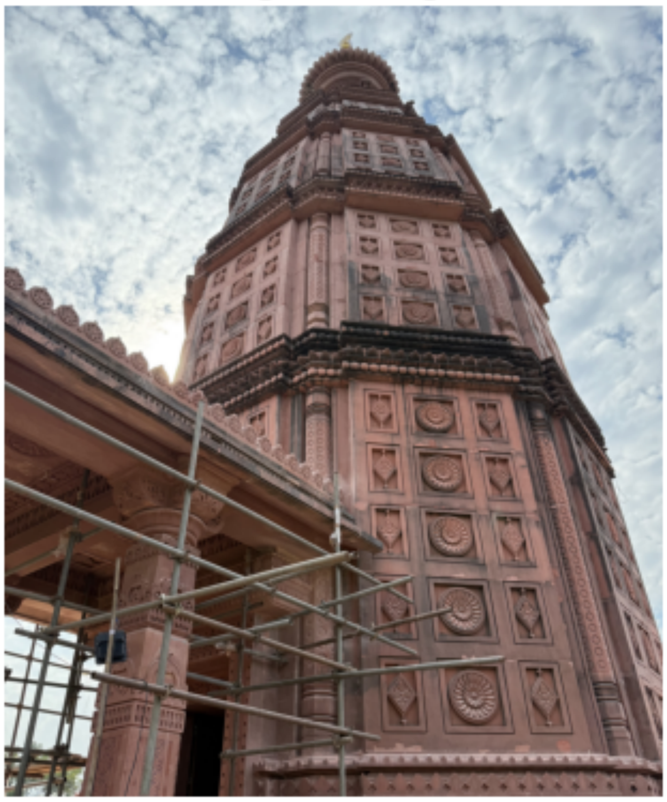
```
!pip3 install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/d
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/d
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
image_path = "/IMG_1.jpg"

image = cv2.imread(image_path)

# Use matplotlib.pyplot's imshow to display the image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)) # Convert from BGR to RGB for
plt.title("Original Image")
plt.axis("off")
plt.show()
```

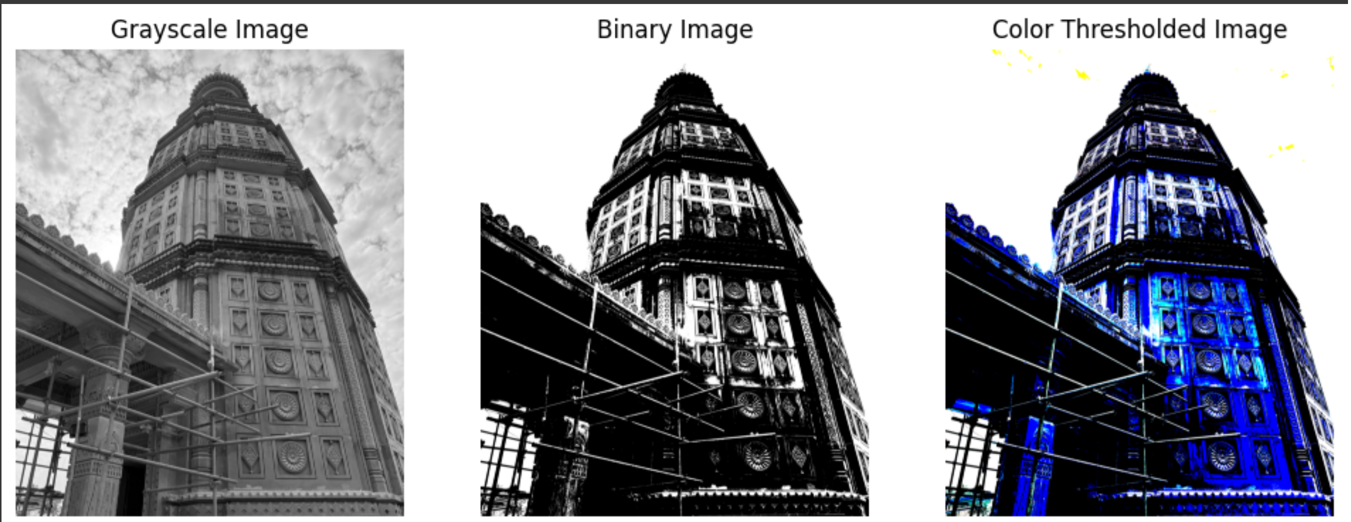
Original Image



```
def reduce_gray_levels(image, levels):
    """
    Reduces the number of gray levels in an image.
    :param image: Input grayscale image.
    :param levels: Target number of gray levels (must be power of 2).
    :return: Processed image with reduced gray levels.
    """
    factor = 256 // levels
    reduced_image = (image // factor) * factor
    return reduced_image
```

```
def process_image(image_path):
    """
    Reads an image, converts to grayscale, and performs various analyses.
    """
    # Read the image
    image = cv2.imread(image_path)

    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate required properties
    total_pixels = grayscale_image.size
    max_pixel_value = np.max(grayscale_image)
    mean_pixel_value = np.mean(grayscale_image)

    # Binary thresholding
    binary_image = np.where(grayscale_image < mean_pixel_value, 0, 255).astype(

    # Color thresholding (Separate each channel)
    channels = cv2.split(image)
    thresholded_channels = [np.where(ch < mean_pixel_value, 0, 255).astype(np.
    combined_threshold = cv2.merge(thresholded_channels)

    # Display processed images
    plt.figure(figsize=(12, 8))
    images = [grayscale_image, binary_image, combined_threshold]
    titles = ["Grayscale Image", "Binary Image", "Color Thresholded Image"]

    for i in range(3):
        plt.subplot(1, 3, i + 1)
        cmap = 'gray' if i < 2 else None
        plt.imshow(images[i], cmap=cmap)
        plt.title(titles[i])
        plt.axis("off")

    plt.show()

    # Print properties
    print(f"Image: {image_path}")
    print(f"Total pixels: {total_pixels}")
    print(f"Max pixel value: {max_pixel_value}")
    print(f"Mean pixel value: {mean_pixel_value}")
```

```
image_path = "/IMG_1.jpg"  # Change this to your image path
process_image(image_path)
```



```
Image: /IMG_1.jpg
Total pixels: 22079736
Max pixel value: 255
Mean pixel value: 128.08812012063913
```

Image Processing Summary

Part 1 – Getting Familiar with Image Manipulation

1. Read an Image and Convert to Grayscale

The image is loaded using OpenCV's `cv2.imread()` function. Since OpenCV loads images in BGR format, it is converted to grayscale using `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`. This transformation reduces the image to intensity values, making it easier to process.

2. Calculate and Report the Size of the Image

The total number of pixels in the grayscale image is determined using image.size, which is computed as the product of the image's width and height.

3. Calculate and Report the Maximum Pixel Value

The maximum intensity value in the grayscale image is found using np.max(image). This value represents the brightest pixel in the image, ranging from 0 (black) to 255 (white).

4. Calculate and Report the Mean Pixel Value

The mean pixel intensity is calculated using np.mean(image). This provides an average measure of the brightness across all pixels in the grayscale image.

5. Binary Thresholding

A thresholding technique is applied where all pixels below the mean intensity are set to 0 (black), and those above are set to 255 (white). This converts the grayscale image into a binary representation.

6. Color Thresholding

Each color channel (Red, Green, and Blue) is thresholded separately. The binary results from each channel are then combined to generate a new thresholded color image. This helps in segmenting objects based on intensity levels within each channel.

Visualization and Output

- The original image is displayed using matplotlib after converting it from BGR to RGB.
- The processed grayscale, binary, and color-thresholded images are displayed side by side.
- Key properties such as total pixel count, maximum pixel value, and mean intensity value are printed to provide an overview of the image characteristics.

This process enables better understanding of image intensity distributions and prepares the image for further analysis or processing steps.

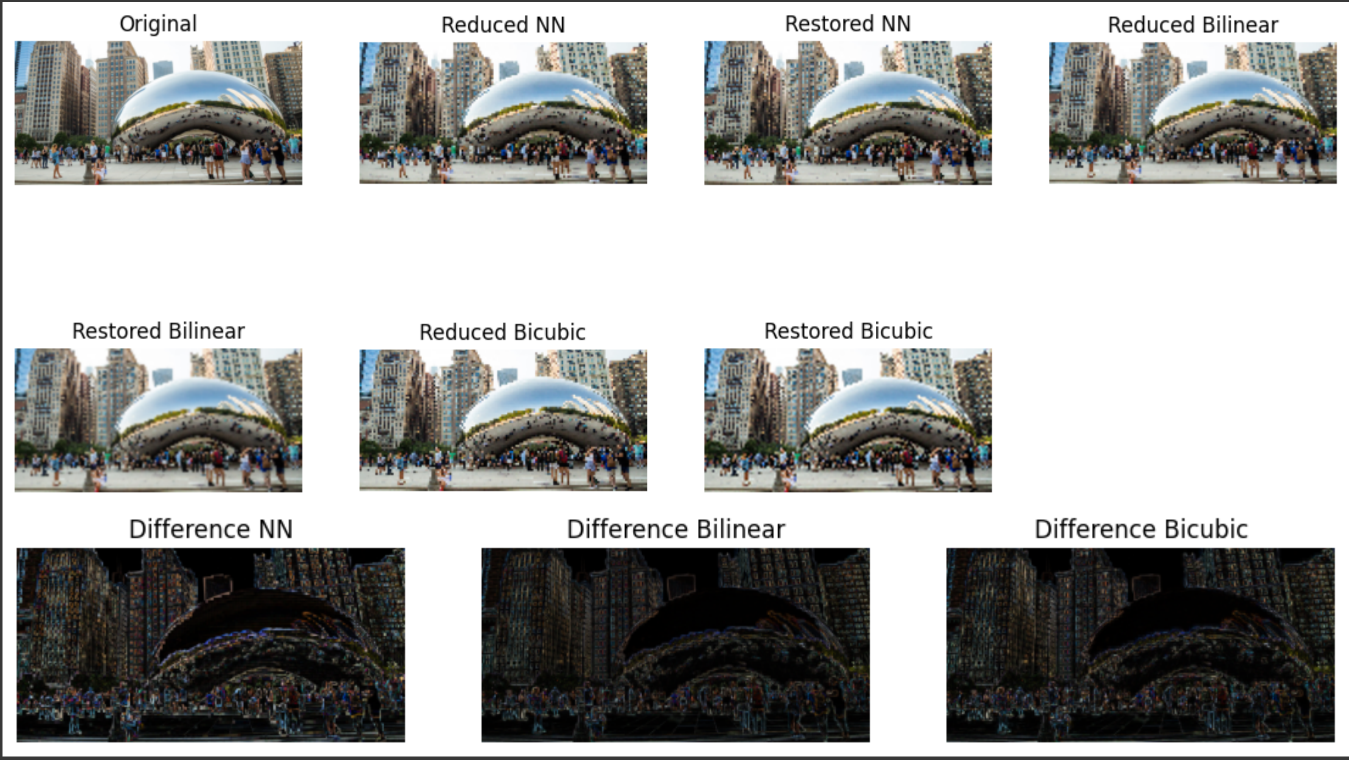
Part 2 - Image Interpolation

1. Reducing and Restoring the image resolution.

Write code that will, given an input image, reduce its spatial resolution(use a reduction of 1/10 to 1/20 in each dimension), and then return it to its original resolution. Use all of nearest neighbor, bilinear and bicubic interpolation to do this. For each input, your display should include at least 7 images: the original, three reduced images and three restored images (one pair for each interpolation). Comment on the differences you see among the three restored images.

In addition to your commentary on the visual differences, perform image subtraction between the original image each of the three restored images and show the subtraction results for each method of interpolation.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def resize_image(image, scale_factor, interpolation):
    new_size = (int(image.shape[1] * scale_factor), int(image.shape[0] * scale_
    return cv2.resize(image, new_size, interpolation=interpolation)

def process_interpolation(image_path):
    image = cv2.imread(image_path)

    if image is None:
        print("Error: Unable to load image. Check the file path.")
        return

    scale_factor = 0.1  # Reduce to 1/10 of original size

    # Reduced images
    reduced_nn = resize_image(image, scale_factor, cv2.INTER_NEAREST)
    reduced_bilinear = resize_image(image, scale_factor, cv2.INTER_LINEAR)
    reduced_bicubic = resize_image(image, scale_factor, cv2.INTER_CUBIC)

    # Restored images
    restored_nn = cv2.resize(reduced_nn, (image.shape[1], image.shape[0]), inte
    restored_bilinear = cv2.resize(reduced_bilinear, (image.shape[1], image.sha
    restored_bicubic = cv2.resize(reduced_bicubic, (image.shape[1], image.shape

    # Display images
    images = [image, reduced_nn, restored_nn, reduced_bilinear, restored_biline
    titles = ["Original", "Reduced NN", "Restored NN", "Reduced Bilinear", "Res

    plt.figure(figsize=(14, 6))
    for i in range(7):
        plt.subplot(2, 4, i + 1)
        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
        plt.title(titles[i])
        plt.axis("off")
    plt.show()

    # Image subtraction for analysis
    diff_nn = cv2.absdiff(image, restored_nn)
    diff_bilinear = cv2.absdiff(image, restored_bilinear)
    diff_bicubic = cv2.absdiff(image, restored_bicubic)

    plt.figure(figsize=(12, 8))
    diffs = [diff_nn, diff_bilinear, diff_bicubic]
    diff_titles = ["Difference NN", "Difference Bilinear", "Difference Bicubic'

    for i in range(3):
        plt.subplot(1, 3, i + 1)
        plt.imshow(cv2.cvtColor(diffs[i], cv2.COLOR_BGR2RGB))
        plt.title(diff_titles[i])
        plt.axis("off")
    plt.show()

# Corrected Image Path
image_path = "/content/IMG_4.jpg"  # Ensure this is the correct path
process_interpolation(image_path)
```


Original
Reduced NN
Restored NN
Reduced Bilinear
Restored Bilinear
Reduced Bicubic
Restored Bicubic
Difference NN
Difference Bilinear
Difference Bicubic

2.Observing Differences in Interpolated Images

- Nearest Neighbor: Results in visible pixelation and sharp edges due to simple nearest-pixel mapping.
- Bilinear: Produces smoother images but may blur fine details.
- Bicubic: Offers the smoothest results with better edge preservation.

3.Image Subtraction for Error Analysis

To quantify differences between the restored images and the original, image subtraction is performed:

diff_nn = cv2.absdiff(image, restored_nn)

diff_bilinear = cv2.absdiff(image, restored_bilinear)

diff_bicubic = cv2.absdiff(image, restored_bicubic)

The resulting difference images highlight areas where interpolation methods introduce distortions.

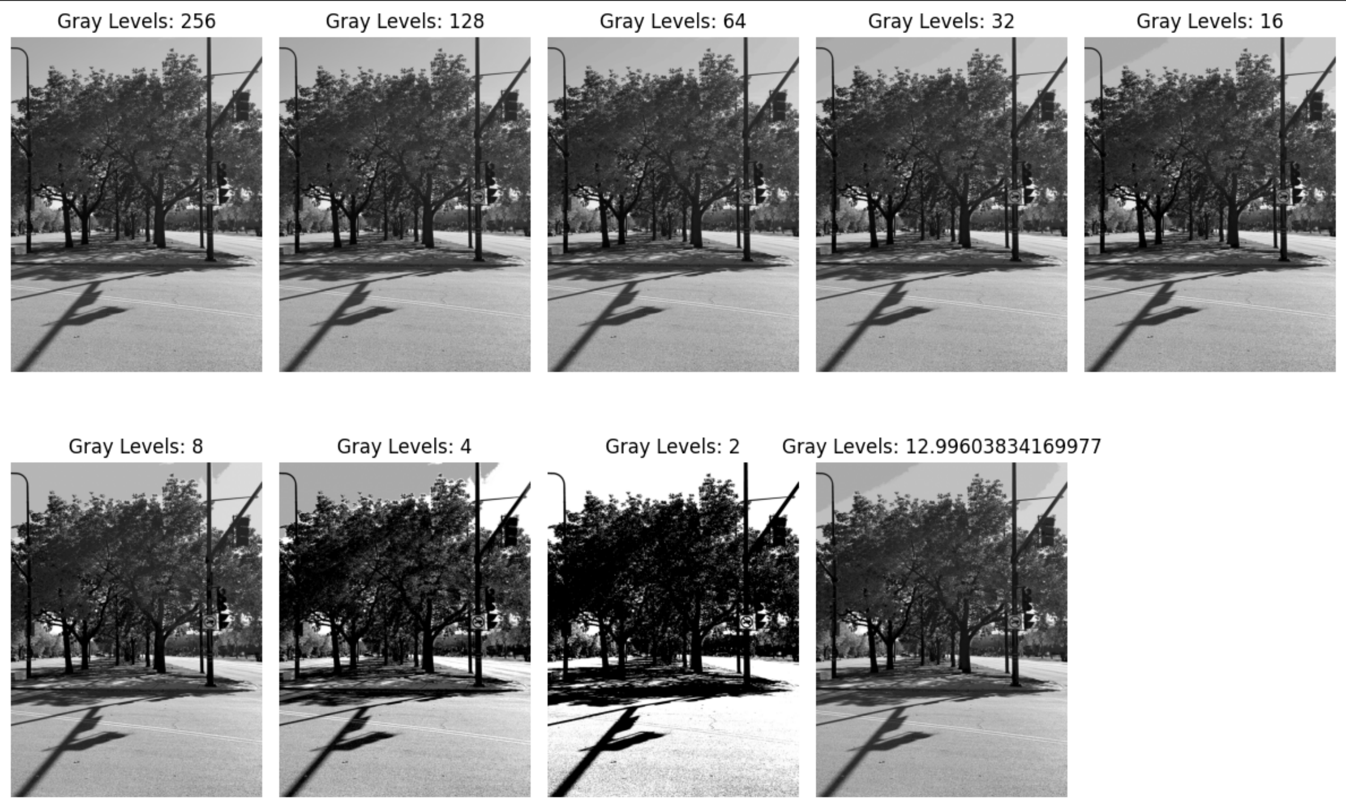
Conclusion

Image interpolation techniques vary in effectiveness depending on the use case. Nearest neighbor is suitable for simple cases where speed is essential, whereas bicubic is preferred for higher-quality restoration. The analysis through image subtraction helps visualize the artifacts introduced by different interpolation methods.

Through these results, we observe that nearest neighbor interpolation introduces blocky distortions, bilinear interpolation smooths edges but blurs details, and bicubic interpolation provides the highest quality restoration with minimal artifacts.

Part 3 – Reducing the Number of Gray Levels in an Image

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def reduce_gray_levels(image, levels):
    factor = 256 / levels  # Compute scaling factor
    reduced_image = (image / factor).astype(np.uint8) * factor  # Reduce gray l
    return reduced_image

# Load the image in grayscale
image_path = "/content/IMG_3.jpg"  # Update with your image path
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define power-of-2 levels and a non-power-of-2 level
gray_levels = [256, 128, 64, 32, 16, 8, 4, 2, float(2**3.7)]

# Plot original and processed images
plt.figure(figsize=(12, 8))
for i, levels in enumerate(gray_levels):
    processed_image = reduce_gray_levels(image, levels)
    plt.subplot(2, 5, i + 1)
    plt.imshow(processed_image, cmap='gray')
    plt.title(f"Gray Levels: {levels}")
    plt.axis("off")

plt.tight_layout()
plt.show()
```


Gray Levels: 256
Gray Levels: 128
Gray Levels: 64
Gray Levels: 32
Gray Levels: 16
Gray Levels: 8
Gray Levels: 4
Gray Levels: 2
Gray Levels: 12.99603834169977

Observations

- As the number of gray levels decreases, the image loses details, but contrast is maintained.
- The power-of-2 levels produce gradual transitions, avoiding abrupt intensity changes.
- The non-power-of-2 level (2^3.7) demonstrates how fractional values can impact the reduction process differently.

Conclusion

Reducing gray levels helps analyze how intensity information is stored in images. Using non-power-of-2 levels expands flexibility beyond traditional methods, allowing more customized visual effects.

```
Start coding or generate with AI.
```