Wroclaw University Of Science And Technology

CANER OLCAY 276715

Artificial Intelligence and Computer Vision

20.11.2024

List 2

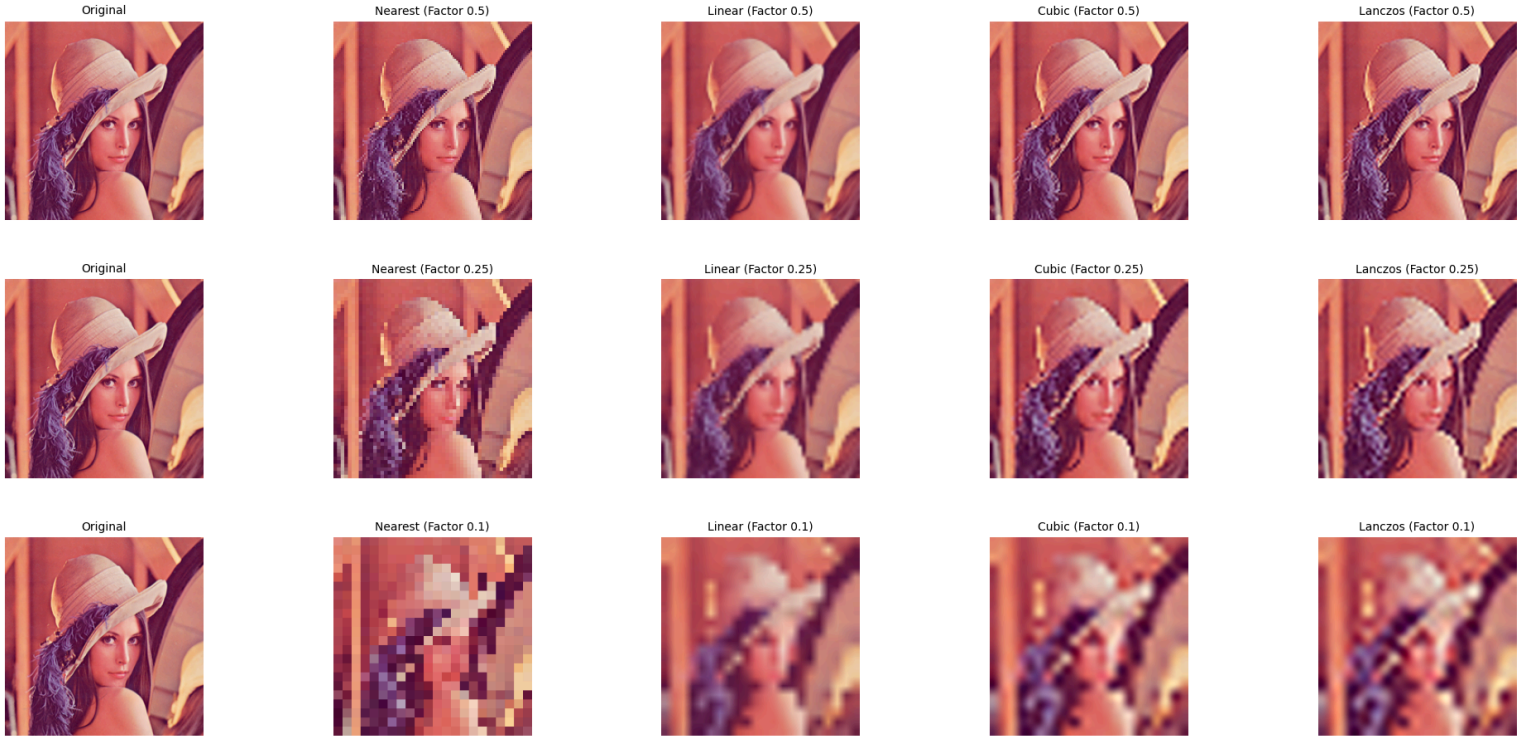
Introduction

This report focuses on two key techniques in digital image processing: image interpolation and histogram equalization, explored as part of the Artificial Intelligence and Computer Vision laboratory coursework. Using the 'Lena' image, I analyzed the effects of various interpolation methods and scaling factors on image quality during resizing. Additionally, I examined histogram equalization's impact on enhancing contrast in underexposed and overexposed images. These tasks highlight the practical applications and importance of these techniques in modern image processing.

Task 1

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



image_path = "C:/Users/caner/OneDrive/Desktop/Python/Lenna.png"

image = cv2.imread(image_path, cv2.IMREAD_COLOR)

if image is None:

    raise FileNotFoundError("Lenna image could not be loaded")

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

scaling_factors = [0.5, 0.25, 0.1]

interpolations = {

    "Nearest": cv2.INTER_NEAREST,

    "Linear": cv2.INTER_LINEAR,

    "Cubic": cv2.INTER_CUBIC,

    "Lanczos": cv2.INTER_LANCZOS4

}

fig, axes = plt.subplots(len(scaling_factors), len(interpolations) + 1, figsize=(20, 15))

for i, scale in enumerate(scaling_factors):

    for j, (method_name, method) in enumerate(interpolations.items()):

        small = cv2.resize(image, None, fx=scale, fy=scale, interpolation=method)

        restored = cv2.resize(small, (image.shape[1], image.shape[0]),
interpolation=method)

        restored_rgb = cv2.cvtColor(restored, cv2.COLOR_BGR2RGB)
```

```
        axes[i, j + 1].imshow(restored_rgb)
        axes[i, j + 1].set_title(f"{method_name} (Factor {scale})", fontsize=10)
        axes[i, j + 1].axis("off")
    axes[i, 0].imshow(image_rgb)
    axes[i, 0].set_title("Original", fontsize=10)
    axes[i, 0].axis("off")
plt.tight_layout()
plt.subplots_adjust(top=0.9, hspace=0.3, wspace=0.3)
plt.show()
```



1. Load the Image

```
image_path = "C:/Users/caner/OneDrive/Desktop/Python/Lenna.png"
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
if image is None:
    raise FileNotFoundError("Lenna image could not be loaded")
```

`image_path`: The location of the image file on my computer.

`cv2.imread()`: Reads the image in color mode (`cv2.IMREAD_COLOR`).

Error Check: Ensures the image is successfully loaded; raises an error if not.

2. Convert Image to RGB

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

OpenCV loads images in BGR format by default. This line converts the image to RGB format so it displays correctly using matplotlib.

4. Define Scaling Factors

```
scaling_factors = [0.5, 0.25, 0.1]
```

- Define scaling factors to reduce the image size: 0.5, 0.25, and 0.1.

5. Define Interpolation Methods

```
interpolations = {
    "Nearest": cv2.INTER_NEAREST,
    "Linear": cv2.INTER_LINEAR,
    "Cubic": cv2.INTER_CUBIC,
    "Lanczos": cv2.INTER_LANCZOS4
}
```

- Nearest: Nearest-neighbor interpolation (simple).

- Linear: Bilinear interpolation (faster).

- Cubic: Bicubic interpolation (higher quality but slower).

- Lanczos: Lanczos interpolation (very high quality but computationally expensive).

6. Process and Visualize the Image

```
fig, axes = plt.subplots(len(scaling_factors), len(interpolations) + 1,
figsize=(20, 15))

for i, scale in enumerate(scaling_factors):

    for j, (method_name, method) in enumerate(interpolations.items()):

        small = cv2.resize(image, None, fx=scale, fy=scale,
interpolation=method)

        restored = cv2.resize(small, (image.shape[1], image.shape[0]),
interpolation=method)

        restored_rgb = cv2.cvtColor(restored, cv2.COLOR_BGR2RGB)

        axes[i, j + 1].imshow(restored_rgb)

        axes[i, j + 1].set_title(f"{method_name} (Factor {scale})",
fontsize=10)

        axes[i, j + 1].axis("off")

    axes[i, 0].imshow(image_rgb)

    axes[i, 0].set_title("Original", fontsize=10)

    axes[i, 0].axis("off")

plt.tight_layout()

plt.subplots_adjust(top=0.9, hspace=0.3, wspace=0.3)

plt.show()
```

In this part, I used a nested loop to apply each interpolation method to images resized with the defined scaling factors. For every scaling factor, the original image is first reduced in size using `cv2.resize` with the chosen interpolation method. Then, the reduced image is upscaled back to its original dimensions using the same method to analyze the impact of resizing on image quality. I displayed the results using `imshow`, where the first column shows the original image, and the subsequent columns display the resized and restored images for each interpolation method. Each subplot includes a title indicating the scaling factor and the interpolation method. To ensure the layout is clean and readable, I used `tight_layout` and `subplots_adjust`. This way, I achieved a clear comparison for all scaling factors and methods.

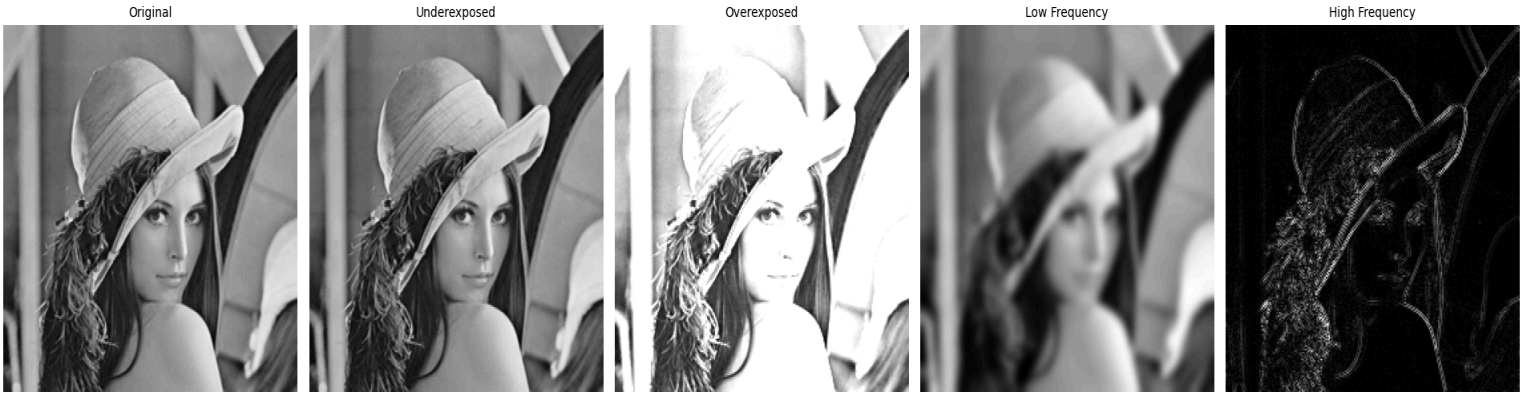
Is effect somehow related with frequency type of image? What is the interpolation method that gives a best results (subjective evaluation) for given type of image?

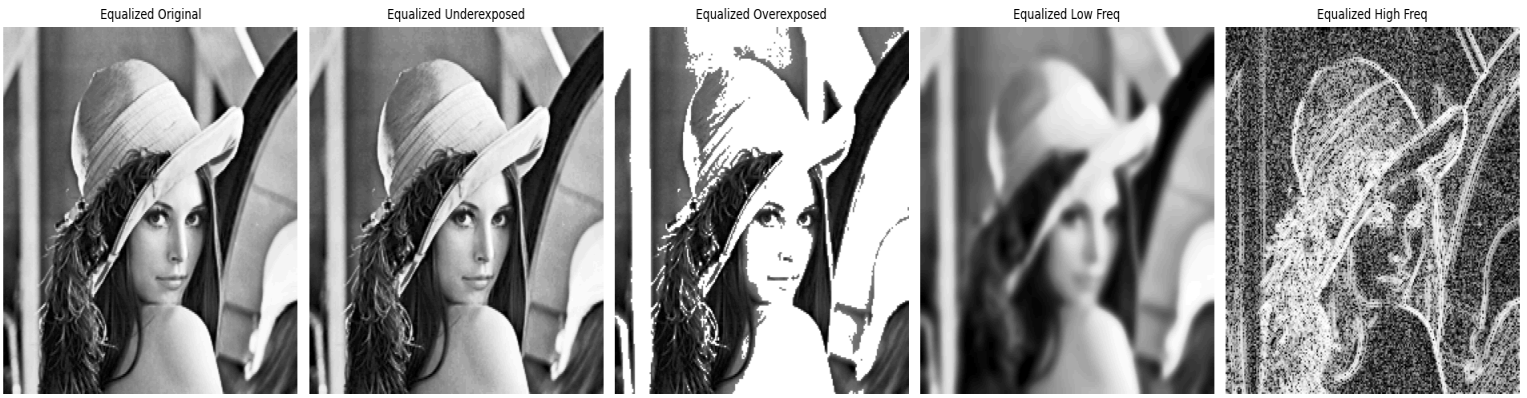
Yes, the effect is related to the frequency type of the image. For low-frequency images with smoother transitions, Linear and Cubic interpolation methods work well, preserving smoothness without major artifacts. In high-frequency images with sharper details, Lanczos gives the best results by maintaining finer details, although it's more computationally demanding. Based on my evaluation, Cubic offers a good quality-performance balance for most cases, while Lanczos is ideal for high detail preservation. Nearest Neighbor results in noticeable pixelation, making it less suitable for quality-focused tasks.

Task 2

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



def adjust_exposure(image, alpha, beta):

    """Adjust exposure of an image."""

    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

image_path = "C:/Users/caner/OneDrive/Desktop/Python/Lenna.png"

image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:

    raise FileNotFoundError("Image not found. Check the file path.")

low_frequency_image = cv2.GaussianBlur(image, (15, 15), 0)  # Smoothing
(Low-Frequency)

high_frequency_image = cv2.Laplacian(image, cv2.CV_64F)     # Edge Detection
(High-Frequency)

high_frequency_image = cv2.convertScaleAbs(high_frequency_image)

underexposed = adjust_exposure(image, alpha=0.5, beta=0)  # Darkened

overexposed = adjust_exposure(image, alpha=1.5, beta=50)  # Brightened

equalized_original = cv2.equalizeHist(image)

equalized_under = cv2.equalizeHist(underexposed)

equalized_over = cv2.equalizeHist(overexposed)

equalized_low = cv2.equalizeHist(low_frequency_image)
```

```
equalized_high = cv2.equalizeHist(high_frequency_image)

titles = [

    "Original", "Underexposed", "Overexposed", "Low Frequency", "High
Frequency",

    "Equalized Original", "Equalized Underexposed", "Equalized Overexposed",

    "Equalized Low Freq", "Equalized High Freq"



]

images = [

    image, underexposed, overexposed, low_frequency_image,
high_frequency_image,

    equalized_original, equalized_under, equalized_over, equalized_low,
equalized_high

]



# I Arranged plots in a 2x5 grid for better alignment

fig, axes = plt.subplots(2, 5, figsize=(20, 10))

for i, ax in enumerate(axes.ravel()):

    ax.imshow(images[i], cmap='gray')

    ax.set_title(titles[i], fontsize=10)

    ax.axis("off")



plt.tight_layout()

plt.show()
```





1. Function to Adjust Exposure

```
def adjust_exposure(image, alpha, beta):
    """Adjust exposure of an image."""
    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
```

alpha: Controls the contrast of the image.

beta: Adjusts the brightness.

This function is used to create underexposed (darker) and overexposed (brighter) images from the original.

2. Load the Original Image

```
image_path = "C:/Users/caner/OneDrive/Desktop/Python/Lenna.png"

image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:

    raise FileNotFoundError("Image not found. Check the file path.")
```

Loads the **original grayscale image** from the given file path.

`cv2.IMREAD_GRAYSCALE`: Ensures the image is loaded in grayscale.

3. Prepare Low-Frequency and High-Frequency Images

```
low_frequency_image = cv2.GaussianBlur(image, (15, 15), 0)  # Smoothing
(Low-Frequency)

high_frequency_image = cv2.Laplacian(image, cv2.CV_64F)     # Edge
Detection (High-Frequency)

high_frequency_image = cv2.convertScaleAbs(high_frequency_image)
```

Low-Frequency Image:

- A **Gaussian Blur** is applied to smooth the image and reduce sharp details (low-pass filter).

High-Frequency Image:

- The **Laplacian Filter** detects edges and sharp details (high-pass filter).
- Converted to an absolute scale using `cv2.convertScaleAbs()`.
-

4. Create Underexposed and Overexposed Images

```
underexposed = adjust_exposure(image, alpha=0.5, beta=0)  # Darkened

overexposed = adjust_exposure(image, alpha=1.5, beta=50)  # Brightened
```

Underexposed:

- Reduces contrast and brightness by setting `alpha=0.5` and `beta=0`.

Overexposed:

- Increases brightness and contrast with `alpha=1.5` and `beta=50`.

5. Perform Histogram Equalization

```
equalized_original = cv2.equalizeHist(image)
equalized_under = cv2.equalizeHist(underexposed)
equalized_over = cv2.equalizeHist(overexposed)
equalized_low = cv2.equalizeHist(low_frequency_image)
equalized_high = cv2.equalizeHist(high_frequency_image)
```

`cv2.equalizeHist()`:

- Enhances image contrast by redistributing intensity values.
- Applied to:
    - Original image
    - Underexposed and overexposed versions
    - Low- and high-frequency images

6. Plot Original and Processed Images

```
fig, axes = plt.subplots(2, 5, figsize=(20, 10))
for i, ax in enumerate(axes.ravel()):
    ax.imshow(images[i], cmap='gray')
    ax.set_title(titles[i], fontsize=10)
    ax.axis("off")
plt.tight_layout()
plt.show()
```

`plt.subplots(2, 5)`:

- Creates a 2x5 grid to display images.
- **Row 1**: Original, underexposed, overexposed, low-frequency, and high-frequency images.
- **Row 2**: Histogram-equalized versions of the images in Row 1.

`imshow()`:

- Displays each image in grayscale.

Titles:

- Clearly identify the type of processing applied (e.g., "Underexposed", "Equalized High Freq").

`tight_layout()`:

- Ensures proper spacing between subplots.

Conclusion

In this laboratory work, I examined the effects of image interpolation and histogram equalization, two essential techniques in digital image processing. Through experimentation, it became evident that interpolation methods play a significant role in the quality of resized images, with high-frequency details being the most sensitive to changes. Among the methods tested, **Lanczos** and **Cubic** interpolation provided the best results, balancing detail preservation and smoothness, particularly for images with mixed frequency content.

In the second task, histogram equalization was applied to images with varying exposure and frequency characteristics. The results showed that histogram equalization enhances contrast effectively, especially in underexposed images, where it brought out details in darker areas. However, for overexposed images, some information was permanently lost despite the improvement in contrast. The technique had a noticeable impact on high-frequency images by enhancing edges, while its effect on low-frequency images was minimal due to the lack of detail.

Overall, these experiments emphasized the importance of selecting the right image processing techniques based on the specific properties of the image and the desired outcome. They highlighted the strengths and limitations of interpolation and equalization methods, offering valuable insights into their practical applications in digital image processing.