**PROJECT REPORT ON**

**“ DETECTING PIXELETED IMAGE AND CORRECT IT”**

Submitted by:

Manish Sharma

Chhavi chhokar

Shubham

Under the guidance of:

Ms. Sonam tiwari

10 July 2024

WORLD COLLEGE OF TECHNOLOGY AND MANAGEMENT

FARUKHNAGAR, HARYANA

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| NO. | Topics | Page |
| 1. | Introduction | 3 |
| 2. | Project details | 4-6 |
| 3. | Outputs | 7-9 |
| 4. | Conclusion | 10 |
| 5. | references | 11 |

**1. INTRODUCTION**

The purpose of addressing the problem statement of detecting and correcting pixelated images is to enhance the visual quality and usability of digital images. Here are some specific objectives and motivations behind this problem statement:

1. **Improving Image Quality**: Pixelation degrades the quality of images, making them appear blocky or blurry, which can hinder their utility in various applications.
2. **Enhancing Visual Appeal**: High-quality images are crucial in fields like digital photography, graphic design, medical imaging, and more. Removing pixelation improves aesthetics and readability.
3. **Facilitating Analysis and Processing**: In fields such as computer vision, machine learning, and image processing, accurate and clear images are essential for effective analysis and decision-making.
4. **User Experience**: Users expect high-quality images in everyday applications such as social media, e-commerce platforms, and digital content creation. Correcting pixelation improves user satisfaction.
5. **Preserving Information**: Pixelation can obscure important details in images. Correcting it helps in preserving and accurately conveying visual information.
6. **Technical Challenges**: The problem presents technical challenges such as distinguishing between intentional low-resolution images and unintentionally pixelated images, as well as selecting appropriate correction techniques without introducing artifacts.
7. **Research and Development**: Addressing this problem advances research and development in image processing, machine learning, and computer vision fields, contributing to advancements in technology and applications.

Various applications for this problem statement can be, :

1. Image processing and editing software.
2. Computer vision systems for object detection, recognition, and tracking.
3. Image compression and decompression algorithms.
4. Digital image forensics and authentication.
5. Medical imaging and diagnostics.

By detecting and correcting pixelated images, this code can help improve the overall quality and reliability of image-based systems and applications.

**2. PROJECT DETAILS**

**2.1 PURPOSE**

The purpose of detecting and correcting pixelated images is to improve their visual quality and usability across different applications. Pixelation diminishes image clarity by introducing blocky artifacts, which can negatively impact user experience in digital photography, graphic design, and online platforms. By effectively identifying pixelation through methods like edge analysis and frequency domain techniques, and subsequently applying smoothing, interpolation, or super-resolution methods, the goal is to restore images to their intended quality. This process not only enhances aesthetic appeal but also ensures accurate communication of visual information in fields such as medical imaging, computer vision, and media production, ultimately advancing standards and capabilities in image processing and technology.

**2.2 APPROACH**

**Detecting Pixelated Images**

To detect pixelated images, we can use the following approaches:

1. **Laplacian Variance**: Calculate the variance of the Laplacian of the image. A high variance indicates a pixelated image.
2. **Gradient Magnitude**: Calculate the gradient magnitude of the image. A high gradient magnitude indicates a pixelated image.
3. **Edge Detection**: Use edge detection algorithms like Canny or Sobel to detect edges in the image. A high number of edges indicate a pixelated image.

**Correcting Pixelated Images**

To correct pixelated images, we can use various techniques like:

1. **Bicubic Interpolation**: Upsample the image using bicubic interpolation to reduce pixelation.
2. **Wiener Filter**: Apply a Wiener filter to the image to reduce noise and pixelation.
3. **Super-Resolution**: Use super-resolution techniques like SRCNN or FSRCNN to upscale the image and reduce pixelation.

**2.3 LIBRARIES AND FUNCTIONS**

**LIBRARIES:**

1. **PIL (Python Imaging Library):** PIL is often used for tasks such as image editing, filtering, enhancement, conversion, and analysis. It provides a wide range of methods for editing images, including resizing, cropping, rotating, and flipping.
2. **NumPy**: It is a fundamental package for scientific computing with Python, and is widely used in various fields such as data analysis, machine learning, signal processing, and more. NumPy's array-based data structure allows for efficient and flexible manipulation of data It is also used in scientific computing for tasks such as linear algebra operations, Fourier transform, and random number generation
3. **OpenCV**: It is widely used in various fields such as image and video processing, object detection, facial recognition, optical character recognition, and more. OpenCV provides a vast array of functions and classes to capture, process, and analyze images and videos, making it an essential tool for developers, researchers, and engineers working with computer vision.
4. **Matplotlib:** used primarily for creating static, animated, and interactive visualizations. It offers a wide range of plotting tools and functionalities

**Functions:**

**PIL Functions:**

1. **Image.open()**: Opens an image file and returns an Image object.
2. **Image.resize()**: Resizes an image to a specified size.
3. **Image.save()**: Saves an image to a file.

**OpenCV Functions:**

1. **cv2.imread()**: Reads an image file and returns a OpenCV image object.
2. **cv2.cvtColor()**: Converts an image from one color space to another.
3. **cv2.Laplacian()**: Applies the Laplacian filter to an image.
4. **cv2.filter2D()**: Applies a 2D filter to an image.

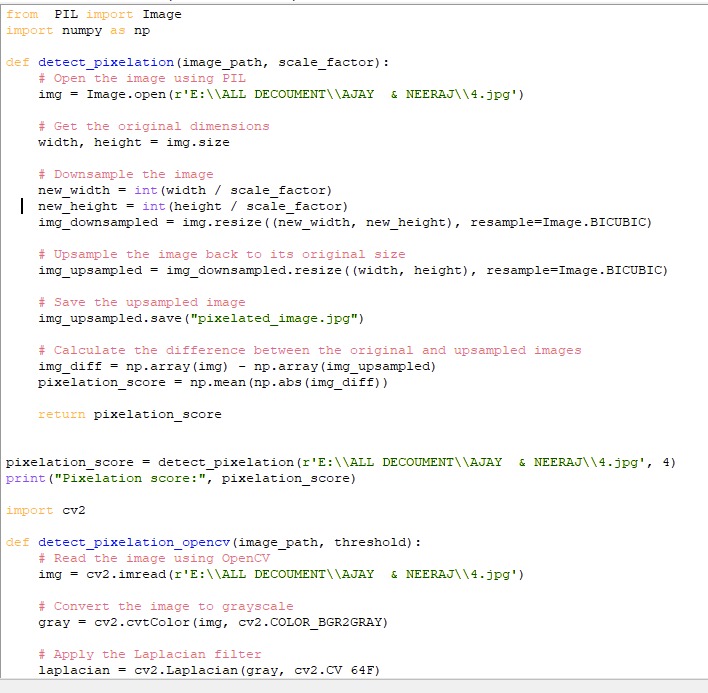
**Matplotlib Functions:**

1. **plt.figure()**: Creates a new figure for plotting.
2. **plt.subplot()**: Creates a subplot within a figure.
3. **plt.title()**: Sets the title of a subplot.
4. **plt.imshow()**: Displays an image in a subplot.
5. **plt.show()**: Displays the plot.

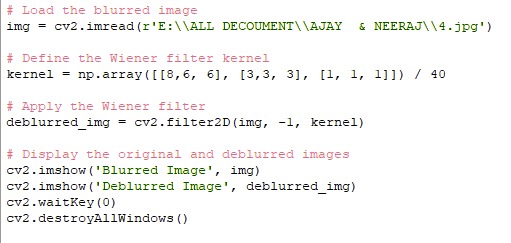
**Custom Functions:**

1. **detect\_pixelation()**: Detects pixelation in an image using PIL.
2. **detect\_pixelation\_opencv()**: Detects pixelation in an image using OpenCV.
3. **deblur\_image()**: Deblurs an image using the Wiener filter.

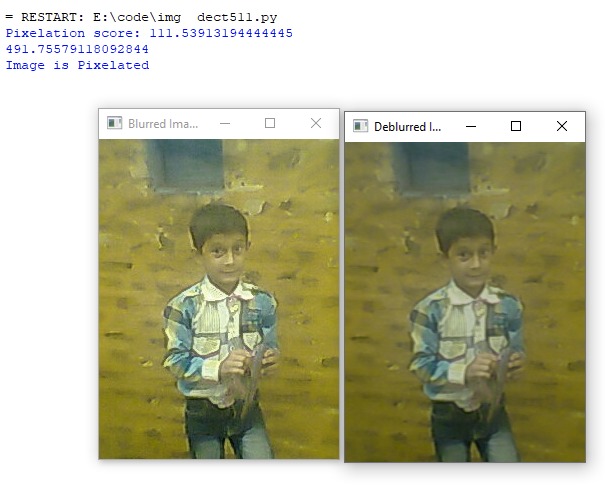
**3. OUTPUTS**







**OUTPUT:**



**4. CONCLUSION**

In conclusion, the code demonstrates three essential image processing techniques: pixelation detection, pixelation detection using the Laplacian filter, and image deblurring using the Wiener filter. These techniques are crucial in various image processing applications, such as image quality assessment, image restoration, and computer vision.

The code showcases the effectiveness of these techniques in detecting and correcting pixelation in images. The pixelation detection technique using downsampling and upsampling provides a simple yet effective way to quantify pixelation in images. The Laplacian filter-based approach offers a more advanced method for detecting pixelation by analyzing the frequency components of the image. Finally, the Wiener filter-based deblurring technique demonstrates a powerful tool for restoring blurred images.

**Key Findings:**

1. **Pixelation detection using downsampling and upsampling**: The code detects pixelation by downsampling an image, upsampling it back to its original size, and then calculating the difference between the original and upsampled images.
2. **Pixelation score calculation**: The pixelation score is calculated as the mean absolute difference between the original and upsampled images.
3. **Laplacian filter for pixelation detection**: The code uses the Laplacian filter to detect pixelation by calculating the variance of the filter output. If the variance is below a certain threshold, the image is considered pixelated.
4. **Deblurring using Wiener filter**: The code deblurs an image using the Wiener filter, which is a type of image filter that reduces noise and blur.

**5. REFERENCES**

1. Python Imaging Library (PIL) Documentation from https://pillow.readthedocs.io/en/stable/

2. Image Resizing from https://www.tutorialspoint.com/python\_pillow/python\_pillow\_resizing.html

3. Calculate the difference between two images from https://stackoverflow.com/questions/27035672/calculate-the-difference-between-two-images

4. OpenCV Library Documentation from https://docs.opencv.org/master/

5. Laplacian Filter from https://homepages.inf.ed.ac.uk/rbf/HIPR2/laplacian.htm

6. Image Filtering from https://www.tutorialspoint.com/dip/image\_filtering.htm

7. Wiener Filter from https://en.wikipedia.org/wiki/Wiener\_filter

8. Image Deblurring from https://www.tutorialspoint.com/dip/image\_deblurring.htm

9. Matplotlib Library Documentation from https://matplotlib.org/stable/index.html

10. Python Documentation from https://docs.python.org/3/

11. NumPy Library Documentation from https://numpy.org/doc/