University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

An Application for Denoising Images with Spatially Correlated Noise Using a Wavelet-based Algorithm

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science
Submitted by:

Cornelius Athus R. Enriquez

July 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "An Application for Denoising Images with Spatially Correlated Noise Using a Wavelet-based Algorithm" prepared and submitted by Cornelius Athus R. Enriquez in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____
Perlita E. Gasmen, M.Sc. (*cand.*)
Adviser

____
Alex C. Gonzaga, Ph.D.
Co-Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	________	________
2. Richard Bryann L. Chua, M.Sc.	________	________
3. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	________	________
4. Vincent Peter C. Magboo, M.D.	________	________
5. Marbert John C. Marasigan, M.Sc. (*cand.*)	________	________
6. Geoffrey A. Solano, Ph.D.	________	________

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

____
Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____
Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____
Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

This project presents an image denoising application based on wavelet-based algorithms for the removal of spatially correlated noise. The application offers users a user-friendly interface to denoise images using various algorithms specifically designed for spatially correlated noise mitigation. Experimental evaluations using PSNR and SSIM as metrics demonstrate the effectiveness of the proposed application in reducing spatially correlated noise while maintaining image quality.

Keywords: Image Denoising, Wavelet-based denoising, Structured Noise2Void, Deep CNN, Denoising application

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

During this digital age, with the rise in the quantity of digital photos taken every day, there is a growing desire for more accurate and visually appealing images [1]. However, the quality of images acquired by modern cameras are still invariably reduced by noise, resulting in bad visual image quality. Because of this, the search for the best technique for reducing the noise of images without compromising quality has also been going on through these years.

Several techniques for removing image noise have been developed, including machine learning and deep learning models like the most frequently used Convolutional Neural Networks (CNN). Another technique that has been used for image denoising is wavelet transform. Images are converted into wavelets and on these wavelets, high frequency components, which are usually noise, are isolated or removed, resulting in a clean image.

However, most of these techniques assumes that the noise is independent Gaussian. Therefore, some of these developed algorithms might not be very effective if the noise does not assume independence like if they are spatially correlated [2].

Spatially correlated noise is a noise that has a correlation between pixels, violating the independence assumption for noise. This type of noise can appear on images that previously developed algorithms and image processing software are not specialized on denoising which could lead to lower quality resulting images.

Among the few developed image denoising techniques that specializes on spatially correlated noise are Structured Noise2Void by Broaddus et al. [3] and a wavelet-based algorithm proposed by Gonzaga [4]. Results showed that these algorithms performed better than thresholding under the presence of spatially correlated noise.

B. Statement of the Problem

Algorithms for denoising images, particularly images with spatially correlated noise have been developed. Nevertheless, there are only few implementations of these techniques that enables them to be actually used. Respectively, Gonzaga's [4] wavelet-based algorithm, which is proven to be effective in attenuating spatially correlated noise, has not yet been implemented, including any variation of it.

C. Objectives of the Study

Broadly, the objective of this study is to develop an image noise suppression application that will address spatially correlated noise through a wavelet-based algorithm where users can upload images, choose an algorithm, and save the resulting clean image. Simulated noise with horizontal correlation was generated and used to test the effectiveness of the algorithm with PSNR and SSIM as metrics. The application also features the use of other denoising algorithm, particularly DCNN and Structured Noise2Void.

Specifically, the research proposes an image denoising application that allows user to

1. Upload the image they wish to denoise in the accepted image formats.
2. Select the denoising algorithm from Deep CNN, Structured Noise2Void, and the proposed algorithm, Wavelet-based Algorithm.
3. Adjust denoising configurations for Wavelet-based Algoritm.
4. Start denoising and view the noisy and resulting image.
5. Save the resulting denoised image.

D. Significance of the Project

Many studies have been done on image denoising, resulting to algorithms that perform well on their purpose. However, most of these algorithms have the assumption that the image noise is a white noise, or spatially uncorrelated and uniformly distributed. Moreover, many image processing software also assumes Gaussian noise, which could lead to poor quality on the resulting image. This study focuses on denoising spatially correlated noise using wavelet transform, exploring a fast and new way of denoising images.

E. Scope and Limitations

The study focuses on developing an algorithm for removing spatially correlated noise on images. Consequently, the project is subject to the following scope and limitation:

1. In evaluating the wavelet-based algorithm, the noisy images used are clear images subjected to generated spatially correlated noise.

2. The simulated noise have horizontal correlation.

3. The software developed will accept limited number of image formats: jpg, jpeg, and png.

4. Input images for the application should contain spatially correlated noise.

F. Assumptions

The following are the assumptions regarding the image, and the application developed.

1. Images used in testing the algorithm are subject to spatially correlated noise, particularly with horizontal correlation.

2. Inputs for the application will be image files, specifically in the format of jpg, jpeg, and png.
3. Images contain spatially correlated noise.

II. Review of Related Literature

Image denoising is indeed a crucial process in various fields, including photography and medical imaging. In a study conducted by Cui et al. [5], image denoising was applied to Positron Emission Tomography (PET) scan images to enhance their quality and aid in accurate diagnosis. The removal of noise from these images is imperative for obtaining clear and reliable results.

However, it is important to acknowledge that image denoising techniques are not flawless and can have certain limitations. One of the compromises associated with denoising is the potential loss of details, such as edges. This is primarily because denoising methods often target high-frequency components that may contain edge information. The removal of noise in these frequency components can inadvertently result in the smoothing or blurring of edges, impacting the overall sharpness and fine details of the image [1].

Therefore, when applying image denoising techniques, it is essential to strike a balance between noise reduction and preserving important image features. Different denoising algorithms and parameters can be explored to find an optimal trade-off between noise removal and detail preservation, depending on the specific requirements of the application or analysis.

A wavelet is a waveform that represents a signal in both the time and frequency domains. In the past, Fourier Transform (FT) was widely used for signal analysis. However, FT does not retain the temporal information of a signal. Wavelet Transform (WT), on the other hand, overcomes this limitation by preserving the duration of a signal. It achieves this by decomposing the original signal into different frequency components, allowing for a more detailed examination of these components and facilitating transformations [6].

One of the key advantages of wavelet transform is its versatility in analyzing various types of data that can be converted into signals or wavelets. This includes audio, video, and image data. By applying wavelet transform to these different types of data, it becomes possible to gain insights into their frequency characteristics and explore their temporal and spectral properties more effectively. The ability to analyze such diverse forms of data makes wavelet transform a valuable tool in many fields, including signal processing, image processing, and data compression.

Wavelet transform indeed plays a crucial role in image processing and offers effective solutions to various image-related problems. Some of the key applications of wavelet transform in image processing include image compression, restoration and denoising, as well as edge and defect detection [7].

Images can be treated as signals, and wavelet transform provides a powerful approach to analyze different components of an image. The high-frequency components, representing the fine details in an image, and the low-frequency components, representing the overall approximation or global features, are transformed into wavelets. This decomposition allows for efficient processing of images, as wavelet transform operates on signals or wavelets.

By decomposing an image into its wavelet components, image compression techniques can be applied, selectively retaining or discarding certain wavelet coefficients based on their importance. This enables effective compression without significant loss of image quality.

Wavelet transform also aids in image restoration and denoising by manipulating the wavelet coefficients. By applying appropriate thresholding or filtering techniques to the wavelet coefficients, noise can be reduced or eliminated, resulting in a clearer and more visually pleasing image.

Additionally, wavelet transform is valuable for edge and defect detection in images. Edge information is often represented by high-frequency components, making it accessible through wavelet analysis. Defects or anomalies in images can be identified by analyzing the wavelet coefficients and identifying significant deviations.

The use of machine learning methods, particularly Convolutional Neural Networks (CNNs), has gained popularity in image denoising research. These techniques leverage the power of deep learning models to learn from existing images and effectively recognize and eliminate noise.

A study by Tian [8] introduced a Deep Convolutional Neural Network (CNN) called the batch-renormalization denoising network (BRDNet). This novel network achieved significant improvements in denoising performance, as measured by the Peak Signal-to-Noise Ratio (PSNR), surpassing other methods such as Block Matching and 3D Filtering.

Another approach proposed by Cheng [9] involved the development of a novel framework combining a CNN called NBNet with a subspace attention (SSA) module. The SSA module, based on subspace projection, contributed to the denoising performance of the network. The combined model demonstrated excellent results in terms of Structural Similarity Index (SSIM) and PSNR.

In the context of medical imaging, Usui et al. [10] utilized CNNs with transfer learning to denoise CT (Computed Tomography) images for the purpose of dose reduction. Transfer learning allowed the model to leverage knowledge gained from pre-trained networks, enabling effective denoising even with limited training data.

These studies highlight the effectiveness of CNN-based approaches in image denoising tasks. The utilization of deep learning models, coupled with innovative network architectures and additional modules, has shown promising results in enhancing denoising performance across various domains, including general image denoising and medical image processing.

Aside from CNNs, other neural networks were also used. In the study of Wang et al. [11], Back Propagation Neural Network was utilized optimized with whale optimization algorithm in comparison to other filtering algorithms Median filtering, Neighborhood average filtering and Wiener filtering, achieving better results. This new optimization algorithm was mainly used to help in neural network training.

Another technique was proposed by Bnou [12], where wavelet denoising based on an unsupervised learning model was used in which an unsupervised dictionary learning model K-SVD was trained on the wavelet decomposition of the noisy image. This presents a new possibility of using the wavelet transform of images for training machine learning algorithms, as these algorithms could react differently between learning an actual image or its wavelet transform.

Much research on denoising images assumes that the noise is independent or sparsely distributed across the image. For example, Lee & Jeong [13] required the assumption of the pixel-wise noise independence to implement their algorithm. Moreover, Cheng [9] mentioned that traditional image and signal denoising methods assumes independent noise. Few research explicitly point out the assumption of uncorrelated noise and even fewer actually focuses on denoising spatially correlated noise. This could pose a problem as denoising techniques that are effective on uncorrelated noise might not be for noise that are correlated. Aelterman conducted experiments on suppressing correlated and uncorrelated image noise and concluded that specialized algorithms should be used in cases where correlated noise is present on images [2] [3]. Among these few research is the study by Broaddus et al. [3] which used Structured Noise2Void as opposed to Noise2Void that assumes independent noise. Upon evaluation using two datasets, Strucn2v showed considerable improvement compared to standard and other blind spot based techniques.

For horizontally correlated images, practical approaches have been used. Jones and Nellist [14] used an algorithm was developed to remove noise and drift from scanning transmission electron microscope images which resulted in a 30% improvement in the signal-to-noise ratio.

The collection of studies discussed above has shed light on various ideas and techniques related to image denoising. It has been observed that Convolutional Neural

Networks (CNNs) are widely employed and considered the predominant technique in image denoising. The contributions and effectiveness of CNNs, both independently and in conjunction with other methods, have been thoroughly explored.

Despite the dominance of CNNs, wavelet transform remains a relevant technique in image denoising, as evidenced by studies incorporating it into their research. This review has revealed the continued importance of wavelet transform in addressing spatially correlated noise reduction.

Furthermore, this review has identified existing algorithms that can serve as benchmarks for comparison with the algorithm being developed. By leveraging these established methods, the researcher can gauge the performance and efficacy of their proposed approach.

It is worth noting that the review has highlighted a gap in the literature pertaining to wavelet denoising specifically targeting spatially correlated image noise. This signifies an unexplored area within the field of image denoising, which the researcher intends to address through their work.

By delving into wavelet transform and focusing on spatially correlated noise, this project aims to contribute to the existing knowledge and provide insights into this understudied aspect of image denoising.

III. Theoretical Framework

A. Image Noise

Noise is an unwanted part of an image [15]. We can often see them in images as scattered black and white dots, kind of like a sprinkle of salt and pepper. This type of noise is a Gaussian noise, which is the most common. There are also different types of noise, like quantization noise and speckle noise. The images below show examples of the mentioned types of noise. [16].

Figure 1: Gaussian noise

Figure 2: Quantization noise

Figure 3: Speckle noise

A noisy image is composed of the original, clear image, and the noise. It can be modeled as

$$g = f + e$$

where g is the observed noisy MxN image, f is the clear image, and e represents the mean-zero Gaussian noise.

B. Spatially Correlated Noise

Assumptions on noise states that it is normal, independent, and identically distributed across the image [17]. However, spatially correlated noise can still exist, violating the independence assumption. Speckle noise, which was mentioned above, is one example of a spatially correlated noise [16]. According to the paper "A survey of spatially correlated noise reduction in images" by D. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian [18], spatially correlated noise can be caused by a variety of factors, including sensor defects, optical distortions, and quantization errors. When dealing with correlated noise, denoising algorithm designed for Gaussian white noise might not be very effective. Jansen [19] found that Generalized Cross Validation (GCV) for white noise was not very effective in eliminating correlated noise. The figure below is an example of an image denoised using GCV. Dabov et al. [18] also note that spatially correlated noise can have a significant impact on the visual quality of an image, and can make it difficult to perform tasks such as image recognition and compression. They suggest that the use of spatially adaptive filtering techniques, such as non-local means, patch-based filtering, and Bayesian methods can be an effective way to remove this type of noise from images.

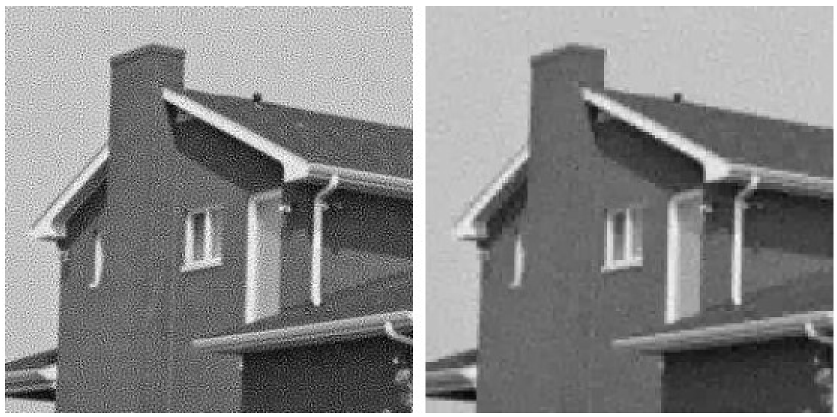

Figure 4: Left: image with artificial correlated noise. Right: denoised image using level-dependent wavelet thresholding

B.1 Horizontally Correlated Noise

Certain types of spatially correlated noise exhibit specific orientations, such as horizontally correlated and vertically correlated noise. These types of noise can occur due to various reasons. Horizontally correlated noise, in particular, is often observed in microscopy images, including those generated by Scanning Electron Microscopy (SEM) [20]. In SEM, beams of electrons are scanned across the specimen, gradually creating a magnified image [21]. This scanning process can introduce horizontally correlated noise into the resulting image due to variations in factors during each pass of the electron beam [20]. Figure 5 shows an example of image taken using a scanning transmission electron microscope where horizontal noise and drift can be seen [14].

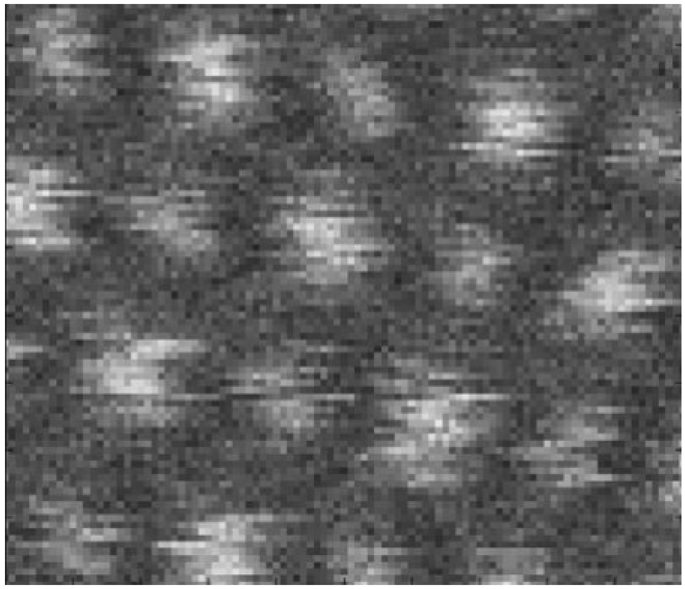

Figure 5: Image produced by Scanning Electron Microscopy (SEM)

C. Fourier Transform

The Fourier Transform provides frequency information of a signal, including their frequencies and magnitude. However, it does not include the time component. Because of this, they are only suitable for signals that do not change over time, as they cannot provide information on frequency and magnitude of signals during a certain period in time. On the other hand, there is Short-Time Fourier Transform (STFT) that gives frequency over time information by dividing the signal into smaller windows of stationary portions. Nevertheless, STFT still cannot tell us the frequencies on a specific time instance. This problem is solved by the Wavelet Transform

D. Wavelet Transform

The Wavelet Transform is a powerful tool for analyzing signals and images, allowing for a localized representation of both frequency and time information. Unlike the Fourier Transform, which uses sinusoidal functions as basis functions, the Wavelet Transform employs wavelets as the basis functions.

The formula for the Continuous Wavelet Transform (CWT) can be expressed as:

$$F(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{+\infty} f(t)\psi^* \left(\frac{t-\tau}{s}\right) dt$$

Here, $f(t)$ represents the input signal, and $\psi(t)$ is the mother wavelet. The CWT decomposes the signal into a set of coefficients $F(\tau, s)$, which provide information about the signal at different time (τ) and scale (s) levels.

The scale parameter s controls the width of the wavelet, allowing us to analyze different frequency components of the signal. By adjusting the scale, we can focus on high-frequency details or low-frequency trends in the signal. The inverse of the scale, $1/s$, represents the frequency of the wavelet.

The translation parameter τ determines the position of the wavelet in the time domain. It allows us to shift the wavelet along the signal to capture different temporal features.

The conjugate complex ψ^* of the mother wavelet is used in the transform to handle complex wavelet functions, such as the Morlet wavelet.

By applying the CWT, we obtain a representation of the signal in the time-frequency domain, where we can identify localized features and analyze the signal at different scales. The CWT is particularly useful for analyzing signals with non-stationary properties, as it adapts to changes in frequency content over time.

D.1 Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) is a variation of the Wavelet Transform that operates on discrete-time signals or discrete images. It decomposes the signal or image into different levels, revealing its frequency components at various scales. The DWT is widely used in image compression, denoising, and feature extraction.

The formula for the DWT can be expressed as:

$$D[a, b] = \frac{1}{\sqrt{b}} \sum_{m=0}^{p-1} f[t_m]\psi \left[\frac{t_m - a}{b}\right]$$

In this formula, $f[t_m]$ represents the input signal or pixel values of the image at the discrete time or position index t_m. The DWT decomposes the signal or image into a set of coefficients $D[a, b]$, where a and b represent the translation and scaling parameters, respectively.

The translation parameter a controls the position of the wavelet in the signal or image. It determines the starting point of the analysis window for each level of the transform. By shifting the wavelet across the signal or image, the DWT captures different localized features.

The scaling parameter b determines the scale or size of the analysis window. It represents the number of samples or pixels over which the wavelet is applied. By adjusting the scale, the DWT can analyze the signal or image at different frequency resolutions. A smaller scale corresponds to a higher-frequency analysis, while a larger scale captures lower-frequency components.

The DWT is performed by applying a set of low-pass and high-pass filters to the signal or image. The low-pass filter extracts the approximation or low-frequency components, while the high-pass filter reveals the detail or high-frequency components. This decomposition process is repeated iteratively to obtain multiple levels of approximation and detail coefficients.

The choice of wavelet function used in the DWT determines the properties of the decomposition. Different wavelets, such as the Haar wavelet, Daubechies wavelets, and Coiflets, offer different trade-offs between time and frequency localization. The selection of an appropriate wavelet depends on the specific application requirements.

In image processing, the DWT can be applied as a two-dimensional transform, analyzing both the rows and columns of the image. This allows for efficient compression and denoising techniques, as the DWT separates the image into its approximate and detail components, where the high-frequency components often represent edges or noise.

E. PSNR and SSIM

Metrics are necessary to assess the effectiveness of different image processing algorithms. Some of these metrics include Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Method (SSIM).

E.1 PSNR

The Peak Signal-to-Noise Ratio (PSNR) is indeed a commonly used measure in image quality assessment. It provides an indication of the image quality by comparing the maximum possible power of an image to the power of the noise that distorts it. PSNR is typically expressed in decibels (dB) and is often used in image and video denoising applications.

The formula for calculating PSNR is as follows:

$$PSNR = 10 \cdot \log_{10}\left(\frac{\text{peakval}^2}{\text{MSE}}\right) \text{ dB}$$

In this formula, "peakval" represents the maximum possible value of the image data. For unsigned 8-bit integer data, the maximum value is 255. The MSE (Mean Squared Error) is a measure of the average squared difference between the original image and the denoised image.

By applying this formula, the PSNR value can be calculated to assess the quality of the denoised image. Higher PSNR values indicate better image quality, as they reflect a higher ratio between the peak power and the noise power.

It is important to note that the typical range of PSNR values can vary depending on the image format and the specific application. For 8-bit images, PSNR values typically range from 30 to 50 dB, while for 16-bit images, the range is usually between 60 and 80 dB in image and video denoising applications [22].

E.2 SSIM

The Structural Similarity Index (SSIM) is a widely used measure for evaluating the similarity between two images, considering the factors of luminance, contrast, and structure. It assesses the correlation of these characteristics on a local level and aggregates the outcomes across the entire image to obtain an overall similarity score [23]. The SSIM index offers valuable insights into whether an image denoising algorithm has not only effectively removed noise but also preserved essential image details.

Mathematically, the SSIM can be expressed as follows:

$$SSIM(x, y) = [l(x, y)]^{\alpha}[c(x, y)]^{\beta}[s(x, y)]^{\gamma}$$

Here, $l(x, y)$, $c(x, y)$, and $s(x, y)$ represent the comparisons of luminance, contrast, and structure, respectively, between the two images denoted as x and y. The parameters α, β, and γ control the relative importance of each component in the overall similarity score. The utilization of SSIM as a quantitative metric offers a comprehensive evaluation of the performance of the image denoising algorithm.

F. Wavelet-based algorithm for diminishing spatially correlated noise

The wavelet-based algorithm used as the implementation for denoising spatially correlated noise is be based on the algorithm proposed by Gonzaga [4] which is as follows:

This algorithm combines the advantages of wavelet thresholding, which removes noise by shrinking the wavelet coefficients, and Wiener filtering, which further enhances the denoising performance by considering the statistical properties of the noise and the image.

By iteratively updating the estimates of the noise-free coefficients and applying Wiener filtering, the algorithm progressively refines the denoised image until con-

Algorithm 1 Wavelet-based algorithm for attenuating spatially correlated noise

1: Obtain the two-dimensional wavelet transform of the image.
2: Obtain an estimate of $Var(W_{\mathbf{g}}(j, m, n))$.
3: Obtain initial estimates of Wf by thresholding.
4: Obtain values of error terms by taking the difference $Wg - Wf$.
5: Estimate $Var(We(j, m, n))$.
6: Estimate Wf by Wiener filtering.
7: Go to Step 4 and iterate until the difference of successive estimates of Wf are arbitrarily small.
8: Obtain f by taking the inverse wavelet transform of the final estimate of Wf in Step 7.

vergence is reached. The quality of the denoised image improves as the difference between successive estimates becomes smaller.

G. Django Framework

Django framework is a powerful web development framework that follows the Model-View-Controller (MVC) architectural pattern. Django offers a comprehensive set of tools and features to facilitate the development of web applications.

The Django framework comprises several key components that contribute to the efficient and structured development of web applications:

- Model: The Model component represents the data structure and defines the database schema. It includes the models that map to database tables and encapsulate the business logic for data manipulation.

- View: The View component handles the logic and processes user requests. It retrieves data from the model, applies necessary transformations or operations, and prepares the data to be rendered in the template.

- Template: The Template component defines the presentation layer and is responsible for generating the user interface. It contains HTML files with embedded template tags and variables that dynamically render the data received from the view.

- URL Dispatcher: The URL Dispatcher maps incoming requests to appropriate views based on predefined URL patterns. It enables proper routing and navigation within the application, ensuring that each request is directed to the correct view

- Forms: Django provides a built-in form handling mechanism that simplifies the validation and processing of user input. Forms help ensure data integrity and provide a user-friendly interface for input validation.

By utilizing the Django framework, developers can take advantage of its robust features, such as built-in security measures, user authentication, session management, and database abstraction. These features contribute to the overall reliability and scalability of the web application.

IV. Design and Implementation

A. Wavelet-based Algorithm

Gonzaga's wavelet-based algorithm [4] was modified to specialize for spatially correlated noise. Thus, the resulting algorithm designed is as follows:

Algorithm 2 Wavelet-based algorithm for attenuating spatially correlated noise

1: Obtain the two-dimensional wavelet transform of the image.
2: Obtain initial estimates of Wf by thresholding on the horizontal coefficients.
3: Estimate Wf by Wiener filtering.
4: Obtain f by taking the inverse wavelet transform of the final estimate of Wf in Step 3.

The first step in the denoising process involved obtaining the wavelet coefficients of a single-channel noisy image using a two-dimensional wavelet transform. This transformation effectively separated the image into two components: the approximation coefficients and the detail coefficients. The approximation coefficients represent the low-frequency components of the image, while the detail coefficients encompass the high-frequency components, which are further categorized into horizontal, vertical, and diagonal details.

To reduce the impact of horizontal noise, a soft thresholding function was applied specifically to the horizontal detail coefficients. This process effectively removed unwanted noise in the horizontal direction, enhancing the clarity of the image. Subsequently, Wiener filtering, a statistical estimation technique, was employed on the entire image. This filtering method further suppressed residual noise and other types of noise present in the image, resulting in a cleaner representation.

Finally, to reconstruct the denoised image, an inverse wavelet transform was performed on the modified coefficients. This transformation restored the image by combining the denoised approximation and detail coefficients, resulting in a high-quality representation with reduced noise.

By employing these denoising techniques in sequence—wavelet transform, soft thresholding, Wiener filtering, and inverse wavelet transform—it is possible to effectively enhance an image by mitigating various types of noise and improving its overall quality.

B. Use Cases

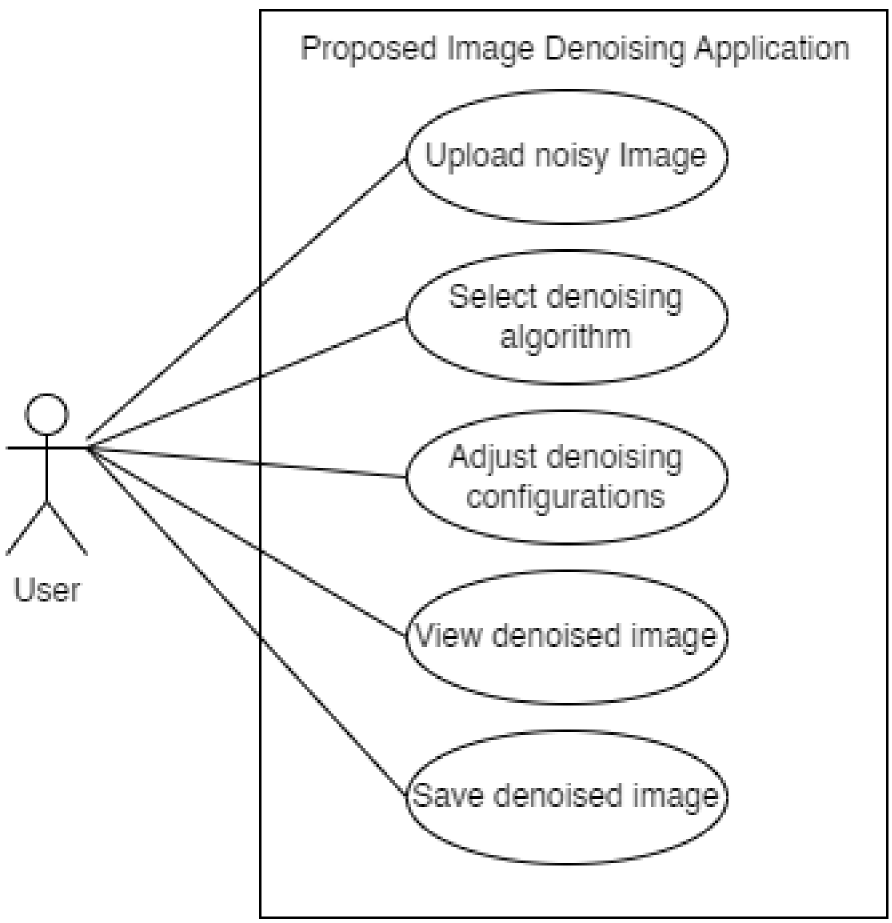

Figure 6: Use Case Diagram of System

The above use case diagram describes the functionalities that users can access in the system. First, noisy image can be uploaded. The application should only accept image files in the upload section. Next, the user can select the denoising algorithm that they can use to denoise the image and edit configurations based on the selected algorithm. Lastly, after finalizing the configurations, the denoised image should appear beside the original noisy image that users can download or save.

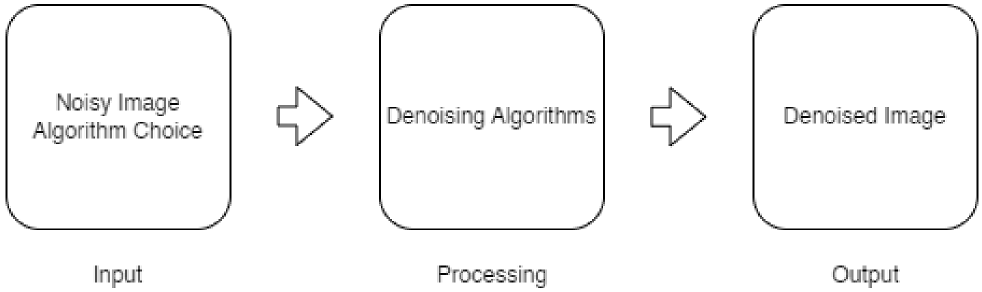

Figure 7: IPO diagram of the system

To simplify how the system works, above is an IPO diagram where the inputs, process, and output of the system is illustrated. Inputs include the noisy image in the acceptable format, uploaded by the user, together with the choice of denoising algorithm to denoise the image with. For the process, depending on the algorithm chosen, a specific implementation of an algorithm will be used with the image as an input. Lastly, after the application implemented the chosen algorithm, the resulting denoised image will be the output, displayed on the page and can also be downloaded by the user.

C. System Architecture

To develop the web application, the Django Framework was employed, enabling the division of the system into the application and presentation layers.

C.1 Presentation Layer

The presentation layer comprises the templates, which are HTML files specifically designed to facilitate seamless user interaction with the application. These templates define the structure and layout of the web pages, allowing the presentation of dynamic content and the incorporation of user interface elements. Through the use of HTML, CSS, and JavaScript, the presentation layer ensures a visually appealing and user-friendly experience.

C.2 Application Layer

The application layer encompasses the views, which play a crucial role in handling request processing, input retrieval, and manipulation. Views act as the intermediary between the user's actions and the underlying business logic of the application. They receive and process requests from the presentation layer, perform necessary operations, and generate appropriate responses. This layer also includes the URLs that map incoming requests to specific views, enabling proper routing and navigation within the application.

By separating the application and presentation layers, Django provides a clear and organized structure for web development. This separation allows for modular development, easier maintenance, and better code reuse. The Django Framework's built-in functionalities and conventions streamline the development process, enabling a better focus on implementing the algorithms and creating an engaging user interface.

D. Technical Architecture

To run Python scripts for applying image noise to the dataset, training, evaluating, and saving models, Google Colab Virtual Environments were used with the following specifications:

1. CPU: 2-core Xeon 2.2GHz
2. Memory: 13 GB
3. Disk: 130 GB

The technical architecture of the application is built using the Python programming language. It is an image processing application that incorporates various image denoising techniques, such as wavelet denoising and machine learning. The following Python libraries were utilized:

1. PIL (Python Imaging Library): This library is used to handle image files, allowing conversion of images into a format suitable for image processing. It provides functions for reading, manipulating, and saving images.

2. Keras: Keras is a popular deep learning library that provides high-level APIs for building and training neural networks. In the context of image denoising, Keras is used to load pre-trained models and make predictions on the input images.

3. PyWavelets (pywt): PyWavelets is a library that provides various wavelet transforms and utilities for signal and image processing. In image denoising, pywt is utilized to decompose the input images into different wavelet coefficients, allowing for the removal of noise and reconstruction of the denoised image.

4. NumPy: NumPy is a fundamental library for numerical computing in Python. It provides powerful array and matrix operations, making it suitable for representing and manipulating images as arrays. NumPy is widely used in image processing tasks, including image denoising, due to its efficiency and convenience.

5. Skimage: Skimage, short for scikit-image, is an image processing library that provides a collection of algorithms for image manipulation and analysis. It offers various functions and utilities for image denoising, including the calculation of evaluation metrics such as PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index). These metrics are used to assess the quality of the denoising algorithm's output.

6. n2v: The n2v library is specifically designed for training and applying the Structured Noise2Void (N2V) model. This model is used for denoising images corrupted by structured noise. The n2v library provides functionalities for data preparation, model training, and prediction using the N2V model.

This technical architecture enables efficient image processing and denoising capabilities within the application.

V. Results

A. Denoising images

A.1 Dataset

In order to evaluate the image denoising algorithm, the CIFAR-10 dataset from Keras was utilized. The CIFAR-10 dataset consists of 60,000 RGB images with a resolution of 32x32 pixels. Among these images, 50,000 belong to the training set, while the remaining 10,000 are assigned to the test set. Originally designed for image classification tasks, this dataset was repurposed for this paper to serve as a collection of clean images that would be subjected to simulated correlated noise.

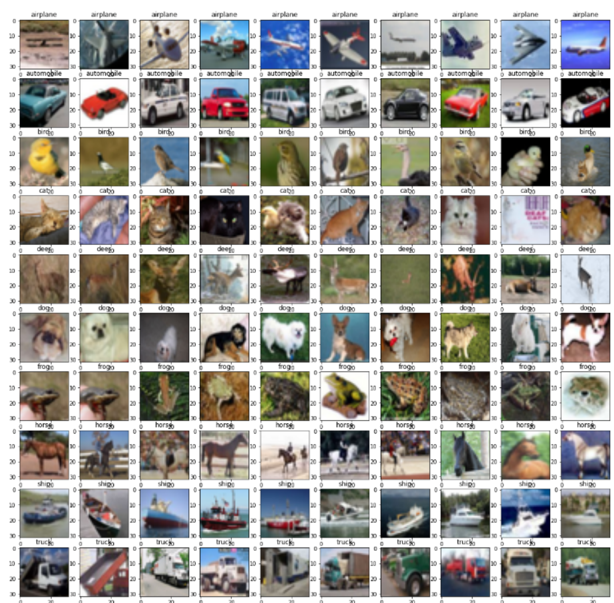

Figure 8: CIFAR-10 Dataset

A.2 Spatially Correlated Noise

To simulate image noise with horizontal correlation, a specific approach was adopted. This involved applying a convolution operation to introduce noise onto the image. The process entailed convolving random Gaussian noise with a 1 by 3 noise kernel across different color channels.

The random Gaussian noise component served as a source of random variation, mimicking the inherent noise present in real-world images. The 1 by 3 noise kernel was specifically designed to introduce horizontal correlation to the noise pattern.

By convolving the random Gaussian noise with the 1 by 3 noise kernel, the resulting noise pattern exhibited horizontal correlation. This means that the noise variations in adjacent pixels along the horizontal direction were statistically correlated, creating a specific noise pattern characteristic.

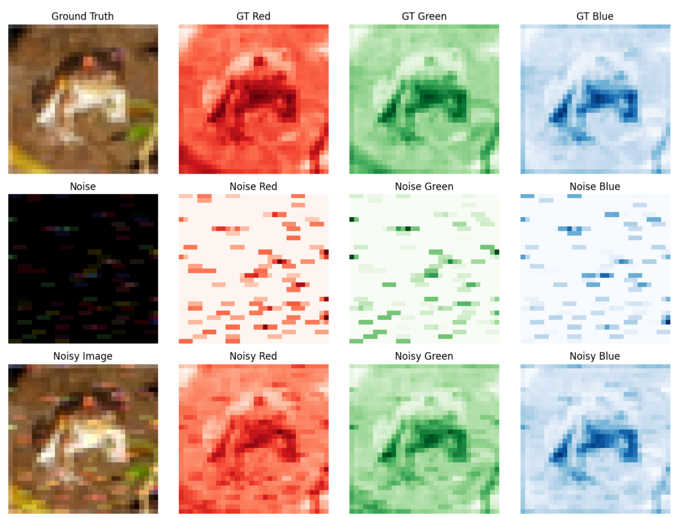

Figure 9: Simulated Horizontally Correlated Noise

A.3 Evaluation of Wavelet-based Algorithm

Figure 10 present an example of denoised images obtained using the wavelet-based algorithm. The algorithm operates only on single-channel images, thus, RGB images are divided in to three channels where the algorithm was applied simultaneously.

To further evaluate the algorithm statistically, it was applied to the test dataset of CIFAR-10 where the average PSNR and SSIM was measured.

As can be seen on Table 1, the wavelet-based denoising algorithm a good average PSNR of 27.1327 and a standard deviation of 0.9485. For mean SSIM, the algorithm

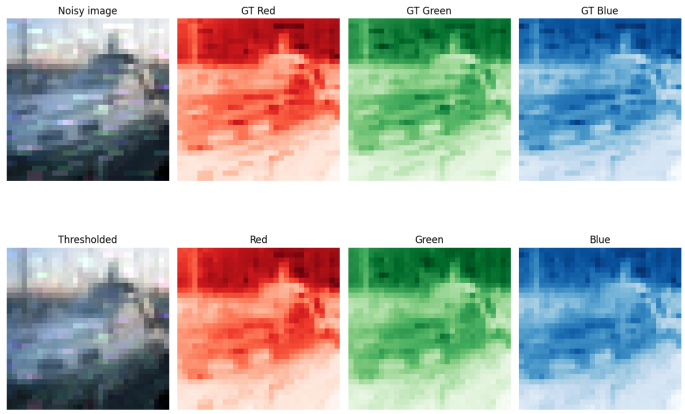

Figure 10: Image denoised using Wavelet-based Algorithm

Table 1: Evaluation of the Algorithm

Metrics	Value
PSNR (Mean)	27.1327
PSNR (Std)	0.9485
SSIM (Mean)	0.9201
SSIM (Std)	0.0290

also got a good value of 0.9201 where 1 means identical to the ground truth image. These metrics signifies that the proposed algorithm has a good and consistent performance across the dataset.

B. Screenshots of the System

The following are the screenshots of the system showing the different functionalities available and sequence for using the application.

The home page (Figure 11) describes the application and the integrated algorithms that can be used for denoising images. It also contains a button that redirects to the denoising tool.

Figure 12 shows a page that includes a form where the user can upload the image to be denoised, choose the technique to be used, and alter configurations for wavelet-based denoising algorithm. Tooltips can also be activated upon hover to guide users on what to input for the configurations.

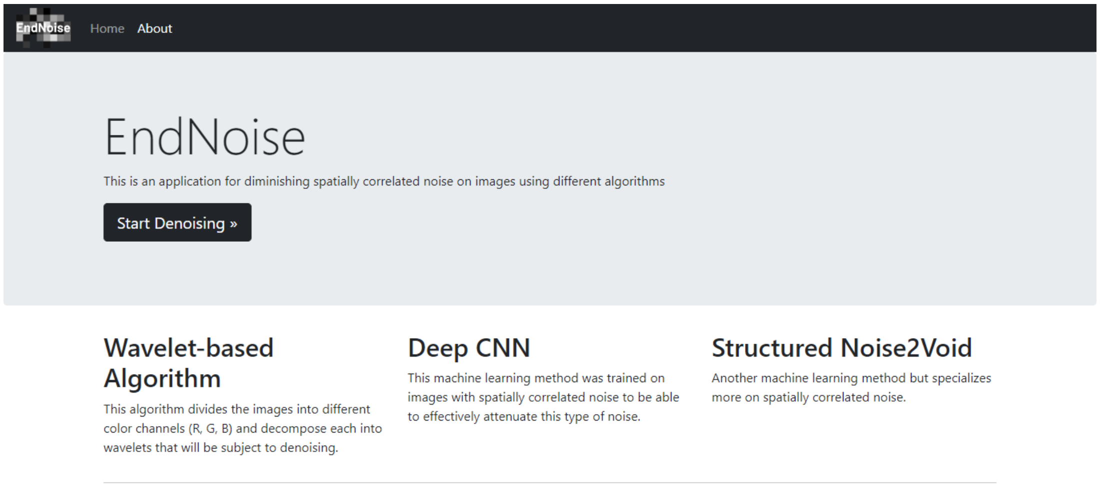

Figure 11: Home Page

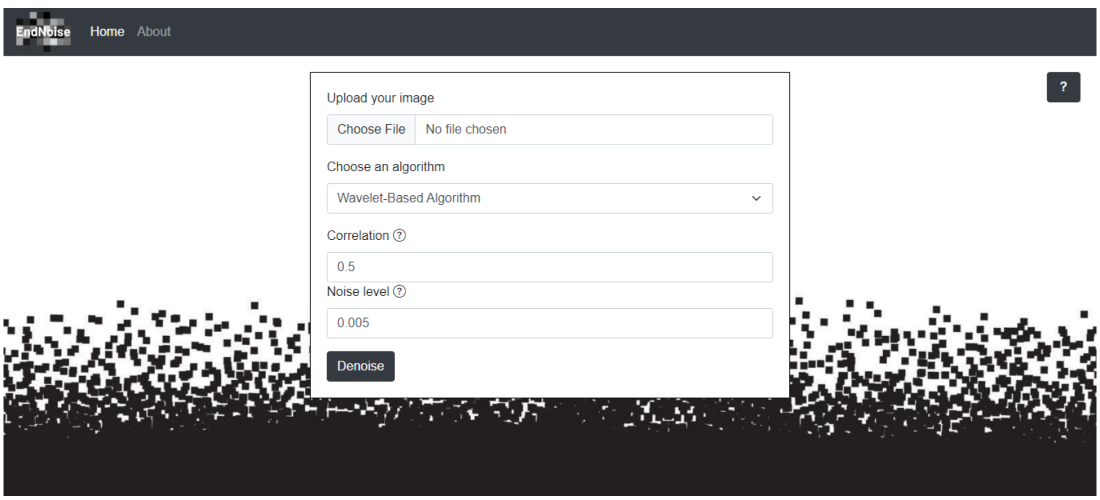

Figure 12: Input page

The input page (Figure 13) also includes a modal that can be activated to show the instructions for using the system.

When the inputs are valid and the denoise button is clicked, it will start a loader that signifies the start of the denoising process. Once done, the user will be redirected to the results page.

The results page (Figure 15) will display the algorithm that was used. Then, it will show the input image together with the denoised image on the right. A download button below the processed image will be available to download the denoised image.

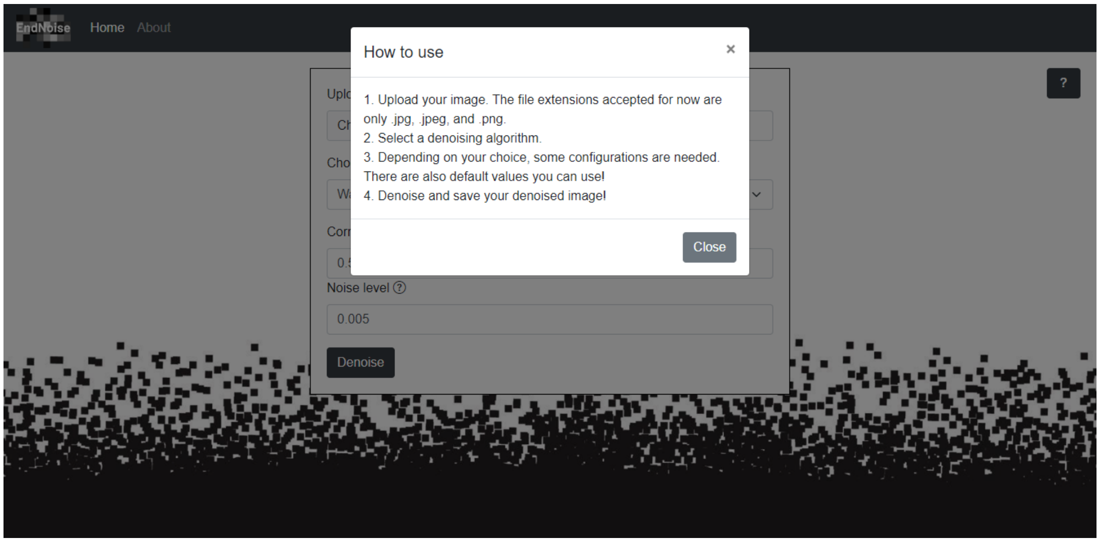

Figure 13: Input page instructions

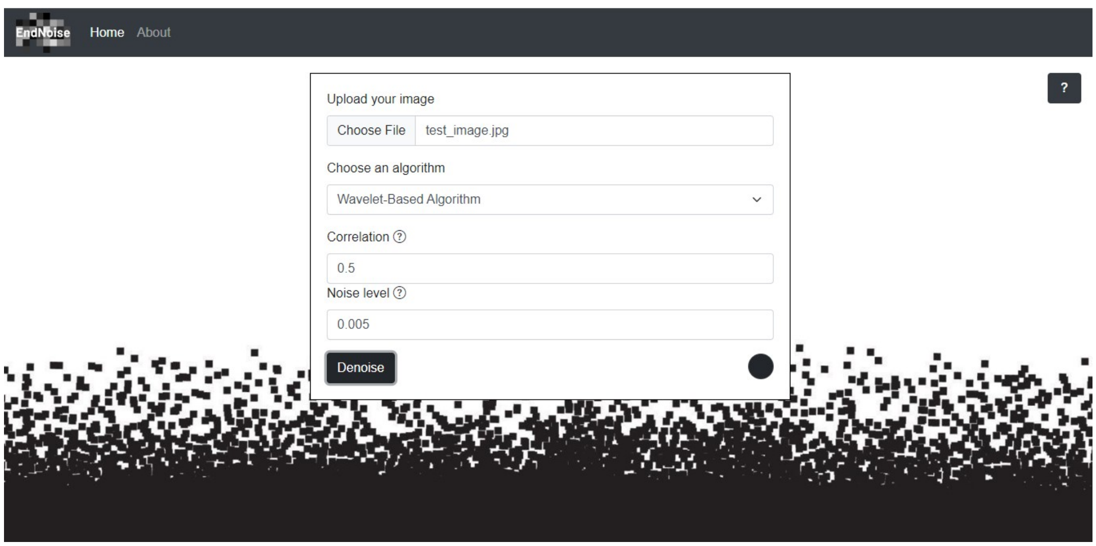

Figure 14: Loading

Also, a button to denoise another image will be available to redirect the user again to the denoise page.

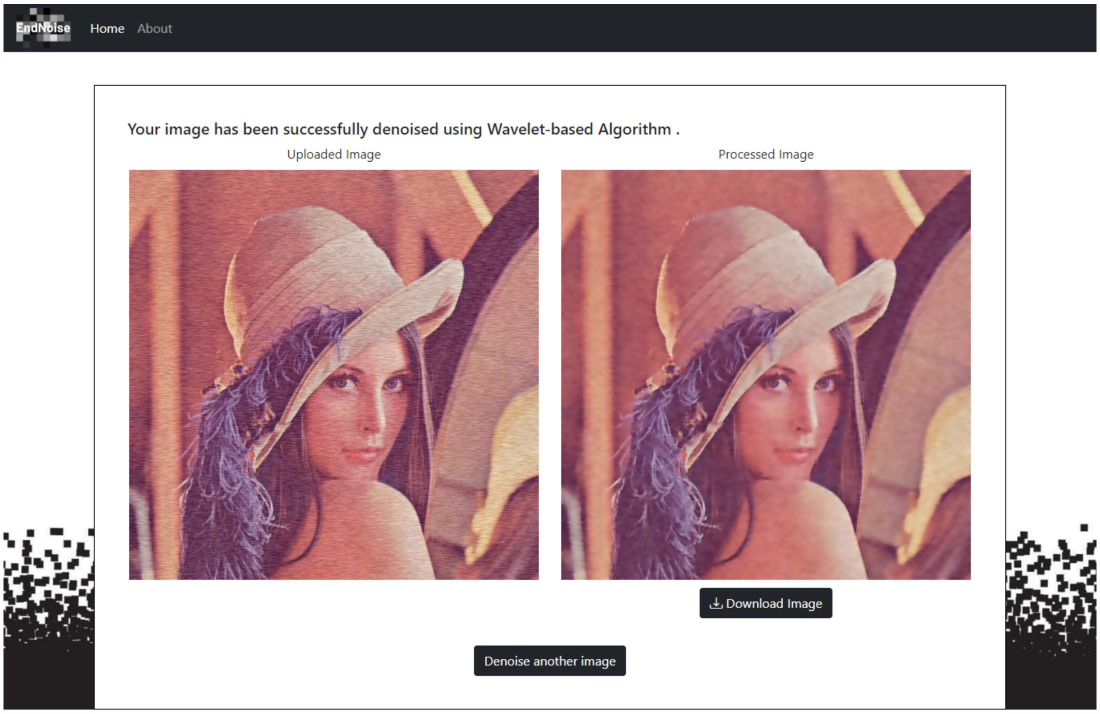

Figure 15: Results page

VI. Discussions

A. Objectives

The results of this Special Problem demonstrate the successful achievement of all the objectives outlined in the paper. A fully functional web application was successfully developed, incorporating three algorithms. The application encompasses all the desired functionalities, allowing users to effectively utilize the noise attenuating algorithms and achieve desired results.

The successful fulfillment of these objectives underscores the significance and effectiveness of the implemented algorithms, as well as the development of the web application as a practical tool for image noise suppression.

B. Challenges

During the development of this special problem, several challenges were encountered. Firstly, training the machine learning models required significant computational resources and a large image dataset. The training process was computationally intensive, necessitating substantial time and computing power.

Secondly, integrating the trained models into the system posed difficulties. Saving and loading the models encountered compatibility issues with different environments, requiring multiple iterations to ensure seamless integration.

Lastly, the image processing pipeline presented its own set of challenges. From the moment images were uploaded to the system to the point of display and user-driven saving, numerous steps were involved. Each step in the process had the potential to introduce some degradation to the processed images, thereby impacting the overall quality.

Addressing these challenges required careful consideration and troubleshooting to optimize the training process, ensure compatibility across different environments, and minimize degradation in image processing steps. Overcoming these hurdles was essential to deliver a robust and reliable system capable of effectively reducing image noise.

C. Significance of the Application

When training machine learning models to denoise images, they are typically unaware of the specific characteristics of the noise that needs to be removed. As a result, this lack of knowledge can lead to suboptimal accuracy and errors. However, utilizing algorithms that specialize in certain noise characteristics can greatly improve performance. This highlights the importance of exploring alternative algorithms that not only offer faster processing times but also deliver better performance under specific conditions.

The aforementioned demonstrates the necessity of employing different algorithms for different types of noise. Consequently, the system recognizes the significance of providing users with a range of algorithms to choose from when reducing image noise. Different algorithms may exhibit varying performance depending on the specific conditions, making it crucial to grant users the freedom to test and determine which algorithm best suits their particular situation.

VII. Conclusions

In conclusion, this study aimed to compare three denoising algorithms by applying horizontally-correlated noise to an image dataset. The models were trained and tested, and the results indicated that the Wavelet-based algorithm outperformed the other algorithms in terms of both Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). This finding suggests that the Wavelet-based algorithm is particularly effective in reducing spatially-correlated noise in images.

Furthermore, the denoising application successfully integrated all three algorithms, allowing users to utilize them for removing spatially correlated noise from their own images. This integration provides users with a practical tool to improve the quality of their images by eliminating unwanted noise.

Overall, this study highlights the effectiveness of the Wavelet-based algorithm in denoising images and underscores the importance of providing users with multiple algorithm options to address different noise characteristics. The successful development and integration of these algorithms into the denoising application demonstrate the practical applicability and usefulness of this research.

VIII. Recommendations

This Special Problem acknowledges the significance of having prior knowledge about the characteristics of the noise to be removed from an image. Building upon this understanding, an improvement that can be proposed is the development of a classification model capable of extracting the specific characteristics of noise present in an image.

By training a classification model on a dataset that includes different types and characteristics of noise, it can learn to classify and identify the specific noise components present in an image. This model can then be used to provide insights into the type and characteristics of noise affecting an image, enabling a more tailored and targeted denoising approach.

Implementing such a classification model would enhance the denoising process by providing valuable information about the nature of the noise. This knowledge can then be leveraged to select or customize denoising algorithms or techniques that are most suitable for the identified noise characteristics. Ultimately, this approach would contribute to improved denoising accuracy and the ability to effectively remove the specific noise components present in an image.

Integrating a noise classification model into the existing denoising system would further enhance its capabilities, providing users with a more automated and intelligent solution for noise removal.

IX. Bibliography

[1] L. Fan, F. Zhang, H. Fan, and C. Zhang, "Brief review of image denoising techniques. visual computing for industry," *Biomedicine, and Art*, pp. 1–12, 2019.

[2] J. Aelterman, B. Goossens, A. Pizurica, and W. Philips, "Suppression of correlated noise," *Recent Advances in Signal Processing*, pp. 211–237, 2009.

[3] C. Broaddus, A. Krull, M. Weigert, U. Schmidt, and G. Myers, "Removing structured noise with self-supervised blind-spot networks," in *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, pp. 159–163, IEEE, 2020.

[4] A. Gonzaga, "Wavelet-based algorithm for attenuation of spatially correlated noise," 04 2010.

[5] J. Cui, K. Gong, N. Guo, C. Wu, X. Meng, K. Kim, K. Zheng, Z. Wu, L. Fu, B. Xu, *et al.*, "Pet image denoising using unsupervised deep learning," *European journal of nuclear medicine and molecular imaging*, vol. 46, no. 13, pp. 2780–2789, 2019.

[6] C. Polat and M. S. Özerdem, "Introduction to wavelets and their applications in signal denoising," *Bitlis Eren Univ. J. Sci. Technol.*, vol. 8, pp. 1–10, June 2018.

[7] S. A. Broughton, *Discrete Fourier analysis and wavelets: applications to signal and image processing.* John Wiley & Sons, 2018.

[8] C. Tian, Y. Xu, and W. Zuo, "Image denoising using deep CNN with batch renormalization," *Neural Netw.*, vol. 121, pp. 461–473, Jan. 2020.

[9] S. Cheng, Y. Wang, H. Huang, D. Liu, H. Fan, and S. Liu, "NBNet: Noise basis learning for image denoising with subspace projection," in *2021 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, June 2021.

[10] K. O. Usui, *Quantitative evaluation of deep convolutional neural network-based image denoising for low-dose computed tomography. Visual Computing for Industry.* 2021.

[11] C. Wang, M. Li, R. Wang, H. Yu, and S. Wang, "An image denoising method based on BP neural network optimized by improved whale optimization algorithm," *EURASIP J. Wirel. Commun. Netw.*, vol. 2021, Dec. 2021.

[12] K. Bnou, S. Raghay, and A. Hakim, "A wavelet denoising approach based on unsupervised learning model," *EURASIP J. Adv. Signal Process.*, vol. 2020, Dec. 2020.

[13] K. Lee and W.-K. Jeong, "ISCL: Interdependent Self-Cooperative learning for unpaired image denoising," Feb. 2021.

[14] L. Jones and P. D. Nellist, "Identifying and correcting scan noise and drift in the scanning transmission electron microscope," *Microscopy and Microanalysis*, vol. 19, no. 4, pp. 1050–1060, 2013.

[15] C. Boncelet, "Chapter 7 - image noise models," in *The Essential Guide to Image Processing* (A. Bovik, ed.), pp. 143–167, Boston: Academic Press, 2009.

[16] S. Gabarda, G. Cristobal, and N. Goel, "Anisotropic blind image quality assessment: Survey and analysis with current methods," *Journal of Visual Communication and Image Representation*, vol. 52, pp. 101–105, 2018.

[17] J. P. C. Kleijnen, "White noise assumptions revisited: Regression metamodels and experimental designs in practice," in *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, p. 107–117, Winter Simulation Conference, 2006.

[18] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “A survey of spatially correlated noise reduction in images,” *IEEE Signal Processing Magazine*, vol. 35, no. 6, pp. 152–168, 2018.

[19] M. Jansen and A. Bultheel, “Multiple wavelet threshold estimation by generalized cross validation for images with correlated noise,” *IEEE transactions on image processing*, vol. 8, no. 7, pp. 947–953, 1999.

[20] H. Ji and S.-Y. Lee, “Designing an anisotropic noise filter for measuring critical dimension and line edge roughness from scanning electron microscope images,” *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 36, no. 6, p. 06JA06, 2018.

[21] A. Mohammed and A. Abdullah, “Scanning electron microscopy (sem): A review,” in *Proceedings of the 2018 International Conference on Hydraulics and Pneumatics—HERVEX, Băile Govora, Romania*, vol. 2018, pp. 7–9, 2018.

[22] U. Sara, M. Akter, and M. S. Uddin, “Image quality assessment through fsim, ssim, mse and psnr—a comparative study,” *Journal of Computer and Communications*, vol. 7, no. 3, pp. 8–18, 2019.

[23] S. S. Channappayya, A. C. Bovik, and R. W. Heath, “A linear estimator optimized for the structural similarity index and its application to image denoising,” in *2006 International Conference on Image Processing*, pp. 2637–2640, IEEE, 2006.

X. Appendix

A. Source Code

train.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import pickle
import time
import ssl
import urllib
import os
import zipfile
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as psnr
from scipy.ndimage import gaussian_filter
from scipy import signal
from keras.datasets import cifar10

from scipy.ndimage import convolve
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import save_model

from n2v.models import N2V, N2VConfig
from n2v.internals.N2V_DataGenerator import N2V_DataGenerator
from n2v.utils.n2v_utils import manipulate_val_data, autocorrelation
from csbdeep.utils import plot_history

# Load the CIFAR-10 dataset
(x_train, _), (x_test, _) = cifar10.load_data()

# Print the shapes of the data arrays
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

# Normalize the image data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.

# Adding horizontally correlated noise to the training images
purenoise_train = []
noise_kernel = np.array([[1, 1, 1]]) / 3  # horizontal correlations
a_train, b_train, c_train, _ = x_train.shape

for i in range(a_train):
    noise = np.random.rand(b_train, c_train, 3) * 1.5
    noise = np.multiply(noise, noise_kernel)
    purenoise_train.append(noise)

purenoise_train = np.array(purenoise_train)
purenoise = purenoise_train - purenoise_train.mean()
```

```python

x_train_noisy = x_train + purenoise_train

# Adding horizontally correlated noise to the test images
purenoise_test = []
noise_kernel = np.array([[1, 1, 1]]) / 3  # horizontal correlations

a_test, b_test, c_test, _ = x_test.shape

for i in range(a_test):
    noise = np.random.rand(b_test, c_test, 3) * 1.5
    noise = np.multiply(noise, noise_kernel)
    purenoise_test.append(noise)

purenoise_test = np.array(purenoise_test)
purenoise = purenoise_test - purenoise_test.mean()

x_test_noisy = x_test + purenoise_test

# Configurations for the Deep CNN model to be trained
dcnn_model = Sequential([
    # encoder network
    Conv2D(
        32,
        3,
        activation='relu',
        padding='same',
        input_shape=(None, None, 3)),
    MaxPooling2D(2,
        padding='same'),
    Conv2D(
        16,
        3,
        activation='relu',
        padding='same'),
    MaxPooling2D(
        2,
        padding='same'),
    # decoder network
    Conv2D(16,
        3,
        activation='relu',
        padding='same'),
    UpSampling2D(2),
    Conv2D(
        32,
        3,
        activation='relu',
        padding='same'),
    UpSampling2D(2),
    # output layer
    Conv2D(
        3,
        (3, 3),
        activation='sigmoid',
        padding='same')
])

dcnn_model.compile(optimizer='adam', loss='binary_crossentropy')
dcnn_model.summary()

```

```
# Train the model
start_time = time.time()

history_dcnn=dcnn_model.fit(x_train_noisy, x_train, epochs=20, batch_size=256, validation_data=(x_test_noisy, x_test))

end_time = time.time()
training_time_dcnn = end_time - start_time

# Save the underlying Keras model and weights
save_model(dcnn_model.keras_model, 'dcnn.h5')
dcnn_model.keras_model.save_weights('dcnn_weights.h5', overwrite=True, save_format=None, options=None)

# Configurations for the
# train_steps_per_epoch is set to (number of training patches)/(batch size), like this each training patch
# is shown once per epoch.
config = N2VConfig(x_train_noisy,
                    unet_kern_size=3,
                    unet_n_first=64,
                    unet_n_depth=3,
                    train_steps_per_epoch=128,
                    train_epochs=20,
                    batch_norm=True,
                    train_batch_size=128,
                    n2v_perc_pix=0.198,
                    n2v_patch_shape=(32,32),
                    n2v_manipulator='normal_withoutCP',
                    n2v_neighborhood_radius=5,
                    single_net_per_channel=False,
                    structN2Vmask=[[0, 1, 1, 1, 0]])

# Let's look at the parameters stored in the config-object.
vars(config)

# a name used to identify the model --> change this to something sensible!
model_name = 'n2v_2D'
# the base directory in which our model will live
basedir = 'models'
# We are now creating our network model.
model = N2V(config, model_name, basedir=basedir)


start_time = time.time()

history = model.train(x_train_noisy, x_test_noisy)

end_time = time.time()
training_time_n2v = end_time - start_time

# Save the underlying Keras model
save_model(model.keras_model, 'n2v_model.h5')
model.keras_model.save_weights('n2v_weights.h5', overwrite=True, save_format=None, options=None)
```

wavelet-based_algorithm.py

```
import numpy as np
import pywt
from google.colab import files
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.datasets import mnist
from scipy.signal import wiener
```

```python
from keras.datasets import cifar10
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as psnr
from scipy.ndimage import convolve

(x_train, _), (x_test, _) = cifar10.load_data()

# normalize the image data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Get the number of samples in the dataset
train_num_samples = x_train.shape[0]
test_num_samples = x_test.shape[0]

x_train = x_train[:train_num_samples]
x_test = x_test[:test_num_samples]
np.save('x_train.npy', x_train)
np.save('x_test.npy', x_test)

# Separate the RGB channels for all images
x_train_red = x_train[:, :, :, 0]
x_train_green = x_train[:, :, :, 1]
x_train_blue = x_train[:, :, :, 2]

np.save('x_train_red.npy', x_train_red)
np.save('x_train_green.npy', x_train_green)
np.save('x_train_blue.npy', x_train_blue)

# Separate the RGB channels for all images
x_test_red = x_test[:, :, :, 0]
x_test_green = x_test[:, :, :, 1]
x_test_blue = x_test[:, :, :, 2]

np.save('x_test_red.npy', x_test_red)
np.save('x_test_green.npy', x_test_green)
np.save('x_test_blue.npy', x_test_blue)

# Function for adding horizontally correlated noise to each of the channels of
# the images
def createNoise(shape):
  # Assuming you have the shape of the image: (b_train, c_train)
  # Assuming you want to generate sparse noise with a scale factor of 0.3

  # Generate separate sparse noise for each channel
  noise_r = np.zeros(shape)
  noise_g = np.zeros(shape)
  noise_b = np.zeros(shape)

  # Define the number of non-overlapping regions
  num_regions = 100
  noise_kernel = np.array([[1, 1, 1]]) / 7  # horizontal correlations

  # Generate random non-overlapping regions for each channel
  for _ in range(num_regions):
      region = np.random.rand(512, 512) < 0.0005  # Define the sparsity level (adjust as needed)
      noise_r[region] = np.random.rand() * 500
      noise_g[region] = np.random.rand() * 500
      noise_b[region] = np.random.rand() * 500
  noise_r = convolve(noise_r, noise_kernel)
  noise_g = convolve(noise_g, noise_kernel)
```

```
  noise_b = convolve(noise_b, noise_kernel)
  # Combine the separate noise channels into a single noise image
  return np.stack((noise_r, noise_g, noise_b), axis=-1), noise_r, noise_g, noise_b

from scipy.ndimage import convolve
noise_kernel = np.array([[1, 1, 1]]) / 3  # horizontal correlations

purenoise_train_combined = []
purenoise_train_correlated_r = []
purenoise_train_correlated_g = []
purenoise_train_correlated_b = []
a_train, b_train, c_train, _ = x_train.shape

for i in range(a_train):
    combined, noise_r, noise_g, noise_b = createNoise(noise_kernel, (b_train, c_train))
    purenoise_train_combined.append(combined)
    purenoise_train_correlated_r.append(noise_r)
    purenoise_train_correlated_g.append(noise_g)
    purenoise_train_correlated_b.append(noise_b)

purenoise_train = np.array(purenoise_train_combined)
purenoise_train = purenoise_train - purenoise_train.mean()
x_train_noisy = x_train + purenoise_train

purenoise_train_r = np.array(purenoise_train_correlated_r)
purenoise_train_r = purenoise_train_r - purenoise_train_r.mean()

purenoise_train_g = np.array(purenoise_train_correlated_g)
purenoise_train_g = purenoise_train_g - purenoise_train_g.mean()

purenoise_train_b = np.array(purenoise_train_correlated_b)
purenoise_train_b = purenoise_train_b - purenoise_train_b.mean()

x_train_noisy_r = x_train_red + purenoise_train_r
x_train_noisy_g = x_train_green + purenoise_train_g
x_train_noisy_b = x_train_blue + purenoise_train_b



purenoise_test_combined = []
purenoise_test_correlated_r = []
purenoise_test_correlated_g = []
purenoise_test_correlated_b = []
a_test, b_test, c_test, _ = x_test.shape

for i in range(a_test):
    combined, noise_r, noise_g, noise_b = createNoise(noise_kernel, (b_test, c_test))
    purenoise_test_combined.append(combined)
    purenoise_test_correlated_r.append(noise_r)
    purenoise_test_correlated_g.append(noise_g)
    purenoise_test_correlated_b.append(noise_b)

purenoise_test = np.array(purenoise_test_combined)
purenoise_test = purenoise_test - purenoise_test.mean()
x_test_noisy = x_test + purenoise_test

purenoise_test_r = np.array(purenoise_test_correlated_r)
purenoise_test_r = purenoise_test_r - purenoise_test_r.mean()

purenoise_test_g = np.array(purenoise_test_correlated_g)
purenoise_test_g = purenoise_test_g - purenoise_test_g.mean()

```

```
purenoise_test_b = np.array(purenoise_test_correlated_b)
purenoise_test_b = purenoise_test_b - purenoise_test_b.mean()

x_test_noisy_r = x_test_red + purenoise_test_r
x_test_noisy_g = x_test_green + purenoise_test_g
x_test_noisy_b = x_test_blue + purenoise_test_b


# Wavelet-based Algorithm
# 1. Obtain the two-dimensional wavelet transform of the image
coeffs_r = []
coeffs_g = []
coeffs_b = []

for r,g,b in zip(x_test_noisy_r,x_test_noisy_g,x_test_noisy_b):
  coeff_r = pywt.wavedec2(r, 'db8', mode='constant', level=1)
  coeff_g = pywt.wavedec2(g, 'db8', mode='constant', level=1)
  coeff_b = pywt.wavedec2(b, 'db8', mode='constant', level=1)
  coeffs_r.append(coeff_r)
  coeffs_g.append(coeff_g)
  coeffs_b.append(coeff_b)

wf_final_r = []
wf_final_g = []
wf_final_b = []

for r, g, b in zip(coeffs_r, coeffs_g, coeffs_b):
    # 2. Obtain initial estimates of Wf by thresholding
    thresh = 0.1
    approx_r, (h1_r, v1_r, d1_r) = r
    approx_g, (h1_g, v1_g, d1_g) = g
    approx_b, (h1_b, v1_b, d1_b) = b

    h1_thresh_r = pywt.threshold(h1_r, thresh, mode='soft')

    h1_thresh_g = pywt.threshold(h1_g, thresh, mode='soft')

    h1_thresh_b = pywt.threshold(h1_b, thresh, mode='soft')

    # 3. Estimate Wf by Wiener filtering.
    wiener_power= 0.005
    approx_r = wiener(approx_r,3,wiener_power)
    approx_g = wiener(approx_g,3,wiener_power)
    approx_b = wiener(approx_b,3,wiener_power)

    denoised_r = (approx_r, (h1_thresh_r, v1_r, d1_r))
    denoised_g = (approx_g, (h1_thresh_g, v1_g, d1_g))
    denoised_b = (approx_b, (h1_thresh_b, v1_b, d1_b))

    wf_final_r.append(pywt.waverec2(denoised_r, 'db8', mode='constant'))
    wf_final_g.append(pywt.waverec2(denoised_g, 'db8', mode='constant'))
    wf_final_b.append(pywt.waverec2(denoised_b, 'db8', mode='constant'))


wf_final = []

for r, g, b in zip(wf_final_r, wf_final_g, wf_final_b):
    wf_final.append(np.stack((r,g,b), axis=-1))

psnrs = []
ssims = []
```

```python

for gt, wf in zip(x_test, wf_final):
  psnrs.append(psnr(gt,wf))
  ssims.append(ssim(gt,wf,multichannel=True))

print("PSNR mean: ", np.mean(psnrs))
print("PSNR std:  ", np.std(psnrs))
print("SSIM mean: ", np.mean(ssims))
print("SSIM std:  ", np.std(ssims))
```

views.py

```python
import io
import os
import pickle

import keras
import numpy as np
import pywt
from django.conf import settings
from django.core.files.base import ContentFile
from django.core.files.storage import FileSystemStorage
from django.db.models.signals import post_delete
from django.dispatch.dispatcher import receiver
from django.shortcuts import redirect, render
from keras.models import load_model
from PIL import Image as Img
from scipy.signal import fftconvolve, wiener
from tensorflow.keras.models import load_model

from .forms import ImageUploadForm
from .models import UploadedImage


@receiver(post_delete, sender=UploadedImage)
def post_save_image(sender, instance, *args, **kwargs):
    """ Clean Old Image file """
    try:
        instance.image.delete(save=False)
    except:
        pass

def process_image(request):
    if request.method == 'POST':
        form = ImageUploadForm(request.POST, request.FILES)
        if form.is_valid():
            images = UploadedImage.objects.all()
            images.delete()
            uploaded_image = form.cleaned_data['image']
            filename, file_extension = os.path.splitext(uploaded_image.name)
            image = Img.open(uploaded_image)
            choice = form.cleaned_data['choice']
            if choice == 'wba':
                thresh = form.cleaned_data['thresh']
                wiener_power = form.cleaned_data['wiener_power']
                processed_image = wavelet(image, thresh, wiener_power, filename, file_extension)
            elif choice == 'dcnn':
                processed_image = dcnn(image, filename, file_extension)
            else:
                processed_image = noise2void(image, filename, file_extension)
            UploadedImage.objects.create(image=uploaded_image, processed_image=processed_image, algo_choice=choice)
```

```
            return redirect('image_result')
    else:
        form = ImageUploadForm()

    return render(request, 'denoise/upload.html', {'form': form})

def show_result(request):
    image = UploadedImage.objects.first()
    return render(request, 'denoise/denoise.html', {'image': image})

def wavelet(image, thresh, wiener_power, filename, file_extension):
    rgb_image = np.array(image.convert("RGB"))
    input_image = rgb_image.astype('float32')/ 255.0
    input_image = np.array(input_image)
    image_r = input_image[:, :, 0]
    image_g = input_image[:, :, 1]
    image_b = input_image[:, :, 2]
    coeff_r = pywt.wavedec2(image_r, 'db8', mode='constant', level=1)
    coeff_g = pywt.wavedec2(image_g, 'db8', mode='constant', level=1)
    coeff_b = pywt.wavedec2(image_b, 'db8', mode='constant', level=1)

    output_image = denoise_wba(coeff_r,coeff_g,coeff_b, thresh, wiener_power)
    output_image = (output_image*255.0).astype('uint8')

    image = Img.fromarray(output_image)
    image_file = io.BytesIO()
    image.save(image_file, format='PNG')
    image_file.seek(0)

    content_file = ContentFile(image_file.read(), name = filename + "_denoised" + file_extension)
    return content_file

def denoise_wba(coeffs_r, coeffs_g, coeffs_b, thresh, wiener_power):
    approx_r, (h1_r, v1_r, d1_r) = coeffs_r
    approx_g, (h1_g, v1_g, d1_g) = coeffs_g
    approx_b, (h1_b, v1_b, d1_b) = coeffs_b

    h1_thresh_r = pywt.threshold(h1_r, thresh, mode='soft')

    h1_thresh_g = pywt.threshold(h1_g, thresh, mode='soft')

    h1_thresh_b = pywt.threshold(h1_b, thresh, mode='soft')

    approx_r = wiener(approx_r,3,wiener_power)
    approx_g = wiener(approx_g,3,wiener_power)
    approx_b = wiener(approx_b,3,wiener_power)

    threshed_r = (approx_r, (h1_thresh_r, v1_r, d1_r))
    threshed_g = (approx_g, (h1_thresh_g, v1_g, d1_g))
    threshed_b = (approx_b, (h1_thresh_b, v1_b, d1_b))

    wf_final_r = (pywt.waverec2(threshed_r, 'db8', mode='constant'))
    wf_final_g = (pywt.waverec2(threshed_g, 'db8', mode='constant'))
    wf_final_b = (pywt.waverec2(threshed_b, 'db8', mode='constant'))

    wf_final = (np.stack((wf_final_r,wf_final_g,wf_final_b), axis=-1))
    return wf_final.astype('float32')/np.max(wf_final)


def dcnn(image, filename, file_extension):
```

```
    rgb_image = np.array(image.convert("RGB"))
    input_image = rgb_image.astype('float32')/ 255.0

    input = []
    input.append(input_image)
    input = np.array(input)
    loaded_model = load_model('denoise/dcnn_model.h5')
    loaded_model.load_weights('denoise/dcnn_weights.h5')

    prediction = loaded_model.predict(input)

    output_image = prediction[0]
    output_image = (output_image*255.0).astype('uint8')
    image = Img.fromarray(output_image)
    image_file = io.BytesIO()
    image.save(image_file, format='PNG')
    image_file.seek(0)
    content_file = ContentFile(image_file.read(), name = filename + "_denoised" + file_extension)
    return content_file

def noise2void(image, filename, file_extension):
    rgb_image = np.array(image.convert("RGB"))
    input_image = rgb_image.astype('float32') / 255.0

    input = []
    input.append(input_image)
    input = np.array(input)

    keras.utils.get_custom_objects()['n2v_abs'] = n2v_abs
    keras.utils.get_custom_objects()['n2v_mse'] = n2v_mse

    loaded_model = load_model('denoise/n2v_model.h5')
    loaded_model.load_weights('denoise/n2v_weights.h5')  # Load the N2V weights

    prediction = loaded_model.predict(input)

    output_image = prediction[0]
    output_image = (output_image*255.0).astype('uint8')
    image = Img.fromarray(output_image)
    image_file = io.BytesIO()
    image.save(image_file, format='PNG')
    image_file.seek(0)
    content_file = ContentFile(image_file.read(), name = filename + "_denoised" + file_extension)
    return content_file

def n2v_abs(y_true, y_pred):
    return keras.backend.mean(keras.backend.abs(y_true - y_pred))

def n2v_mse(y_true, y_pred):
    return keras.backend.mean(keras.backend.square(y_true - y_pred))

def about(request):
    return render(request, 'denoise/about.html')
```

forms.py

```
from django import forms
from .models import UploadedImage
from django.core.validators import MinValueValidator, MaxValueValidator, FileExtensionValidator

class ImageUploadForm(forms.Form):
```

```
    image = forms.ImageField(validators=[FileExtensionValidator(['jpg', 'jpeg', 'png'])])
    CHOICES = (
        ('wba', 'Wavelet-Based Algorithm'),
        ('dcnn', 'Deep Convolutional Neural Network'),
        ('n2v', 'Structured Noise2Void'),
    )
    choice = forms.ChoiceField(choices=CHOICES,widget=forms.Select(attrs={'class': 'form-select', 'id': 'algo'}))
    thresh = forms.FloatField(
        required=False,
        widget=forms.NumberInput(attrs={'class': 'hidden-field'}),
        validators=[MinValueValidator(0), MaxValueValidator(255)],
        initial=0.5)
    wiener_power = forms.FloatField(
        required=False,
        widget=forms.NumberInput(attrs={'class': 'hidden-field'}),
        validators=[MinValueValidator(0), MaxValueValidator(1)],
        initial=0.005
    )
```

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.process_image, name='image_upload'),
    path('result/', views.show_result, name='image_result'),
    path('about/', views.about, name='about'),
]
```

about.html

```
{% load static %}

{% load widget_tweaks %}

<!doctype html>
<html lang="en">

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
    integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">

  <title>EndNoise</title>
  <link rel="icon" href="{% static 'denoise.ico' %}">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x" crossorigin="anonymous">
  <link rel="stylesheet" type="text/css"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css">

  <style>
    .footer-with-bg {
      background-image: url('{% static "bg1.jpg" %}');
      background-repeat: no-repeat;
      background-size: cover;
```

```
      background-position: center bottom;
      position: fixed;
      bottom: 0;
      left: 0;
      width: 100%;
      height: 600px;
      /* Adjust the height as per your image dimensions */
      z-index: -1;
      /* To ensure the footer content is displayed on top */
    }
  </style>
</head>

<body>

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark" style="padding-top:0;padding-bottom: 0;">
    <a class="navbar-brand" href="{% url 'image_upload' %}"><img src="{% static 'endnoise.png' %}" style="height: 50px;"
        alt=""></a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
      aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
      <div class="navbar-nav">
        <a class="nav-link " href="{% url 'image_upload' %}">Home</a>
        <a class="nav-link active" href="{% url 'about' %}">About<span class="sr-only">(current)</span></a>
      </div>
    </div>
  </nav>
  <main role="main">

    <div class="jumbotron">
      <div class="container">
        <h1 class="display-3">EndNoise</h1>
        <p>This is an application for diminishing spatially correlated noise on images using different algorithms</p>
        <p><a class="btn btn-dark btn-lg" href="{% url 'image_upload' %}" role="button">Start Denoising &raquo;</a></p>
      </div>
    </div>

    <div class="container">
      <div class="row">
        <div class="col-md-4">
          <h2>Wavelet-based Algorithm</h2>
          <p>This algorithm divides the images into different color channels (R, G, B) and decompose each into wavelets
            that will be subject to denoising.</p>
        </div>
        <div class="col-md-4">
          <h2>Deep CNN</h2>
          <p>This machine learning method was trained on images with spatially correlated noise to be able to
            effectively attenuate this type of noise.</p>
        </div>
        <div class="col-md-4">
          <h2>Structured Noise2Void</h2>
          <p>Another machine learning method but specializes more on spatially correlated noise.</p>
        </div>
      </div>

      <hr>

    </div> <!-- /container -->

```

```
  </main>
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
    integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-Fy6S3B9q64WdZWQUiU+q4/2Lc9npb8tCaSX9FK7E8HnRr0Jz8D6OP9dO5Vg3Q9ct"
    crossorigin="anonymous"></script>

</body>

</html>
```

denoise.html

```
{% load static %}

{% load widget_tweaks %}

<!doctype html>
<html lang="en">

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
    integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">

  <title>EndNoise</title>
  <link rel="icon" href="{% static 'denoise.ico' %}">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x" crossorigin="anonymous">
  <link rel="stylesheet" type="text/css"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css">

  <style>
    .footer-with-bg {
      background-image: url('{% static "bg1.jpg" %}');
      background-repeat: no-repeat;
      background-size: cover;
      background-position: center bottom;
      position: fixed;
      bottom: 0;
      left: 0;
      width: 100%;
      height: 600px;
      /* Adjust the height as per your image dimensions */
      z-index: -1;
      /* To ensure the footer c
            ontent is displayed on top */
    }

    .center-align {
      display: flex;
      justify-content: center;
      align-items: center;
    }
  </style>
```

```
</head>

<body>

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark" style="padding-top:0;padding-bottom: 0;">
    <a class="navbar-brand" href="{% url 'image_upload' %}"><img src="{% static 'endnoise.png' %}" style="height: 50px;"
        alt=""></a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
      aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
      <div class="navbar-nav">
        <a class="nav-link active" href="{% url 'image_upload' %}">Home<span class="sr-only">(current)</span></a>
        <a class="nav-link" href="{% url 'about' %}">About</a>
      </div>
    </div>
  </nav>
  <div class="container"
    style="margin-top: 3%;margin-bottom: 5%; padding: 3%; border: 1px solid black; background-color: white;">
    <h5>Your image has been successfully denoised using
      {% if image.algo_choice == 'wba' %}
      Wavelet-based Algorithm
      {% elif image.algo_choice == 'dcnn' %}
      Deep CNN
      {% else %}
      Structured Noise2Void
      {% endif %}.
    </h5>
    <div class="row">
      <div class="col-md-6">
        <div class="text-center">
          <label>Uploaded Image</label>
        </div>
        <div class="d-flex justify-content-center">
          <div class="image-box">
            <img src="{{ image.image.url }}" alt="Uploaded Image" class="img-fluid">


          </div>
        </div>
      </div>
      <div class="col-md-6">
        <div class="text-center">
          <label>Processed Image</label>
        </div>
        <div class="d-flex justify-content-center">
          <div class="image-box">
            <img src="{{ image.processed_image.url }}" alt="Processed Image" class="img-fluid">
            <div class="center-align">
              <a id="download-link" href="{{ image.processed_image.url }}" download="image.jpg"
                style="display: none;"></a>
              <button id="download-button" class="btn btn-dark" style="margin: 10px;"><i class="bi bi-download"></i>
                Download Image</button>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="row">
      <div class="col-md-12 text-center mt-4">
```

```
          <a class="btn btn-dark" href="{% url 'image_upload' %}">Denoise another image</a>
        </div>
      </div>
    </div>
    <footer class="footer-with-bg">
    </footer>
    <script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
      integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
      crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"
      integrity="sha384-Fy6S3B9q64WdZWQUiU+q4/2Lc9npb8tCaSX9FK7E8HnRr0Jz8D6OP9dO5Vg3Q9ct"
      crossorigin="anonymous"></script>

    <script>
      document.getElementById("download-button").addEventListener("click", function () {
        var downloadLink = document.getElementById("download-link");
        downloadLink.click();
      });
    </script>
  </body>

  </html>
```

upload.html

```
{% load widget_tweaks %}
{% load static %}

<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM" crossorigin="anonymous">

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
        integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css">
    <title>Endnoise</title>
    <link rel="icon" href="{% static 'denoise.ico' %}">
    <style>
        body {
            font-family: sans-serif;
            background-color: #ffffff;
        }

        .file-upload {
            background-color: #ffffff;
            width: 600px;
            margin: 0 auto;
            padding: 20px;

        }

        .file-upload-btn {
            width: 100%;
```

```
        margin: 0;
        color: #fff;
        background: #1FB264;
        border: none;
        padding: 10px;
        border-radius: 4px;
        border-bottom: 4px solid #15824B;
        transition: all .2s ease;
        outline: none;
        text-transform: uppercase;
        font-weight: 700;
    }

    .file-upload-btn:hover {
        background: #1AA059;
        color: #ffffff;
        transition: all .2s ease;
        cursor: pointer;
    }

    .file-upload-btn:active {
        border: 0;
        transition: all .2s ease;
    }

    .file-upload-content {
        display: none;
        text-align: center;
    }

    .file-upload-input {
        position: absolute;
        margin: 0;
        padding: 0;
        width: 100%;
        height: 100%;
        outline: none;
        opacity: 0;
        cursor: pointer;
    }

    .image-upload-wrap {
        margin-top: 20px;
        border: 4px dashed #1FB264;
        position: relative;
    }

    .image-dropping,
    .image-upload-wrap:hover {
        background-color: #1FB264;
        border: 4px dashed #ffffff;
    }

    .image-title-wrap {
        padding: 0 15px 15px 15px;
        color: #222;
    }

    .drag-text {
        text-align: center;
    }
```

```
        .drag-text h3 {
            font-weight: 100;
            text-transform: uppercase;
            color: #15824B;
            padding: 60px 0;
        }

        .file-upload-image {
            max-height: 200px;
            max-width: 200px;
            margin: auto;
            padding: 20px;
        }

        .remove-image {
            width: 200px;
            margin: 0;
            color: #fff;
            background: #cd4535;
            border: none;
            padding: 10px;
            border-radius: 4px;
            border-bottom: 4px solid #b02818;
            transition: all .2s ease;
            outline: none;
            text-transform: uppercase;
            font-weight: 700;
        }

        .remove-image:hover {
            background: #c13b2a;
            color: #ffffff;
            transition: all .2s ease;
            cursor: pointer;
        }

        .remove-image:active {
            border: 0;
            transition: all .2s ease;
        }

        .footer-with-bg {
            background-image: url('{% static "bg1.jpg" %}');
            background-repeat: no-repeat;
            background-size: cover;
            background-position: center bottom;
            position: fixed;
            bottom: 0;
            left: 0;
            width: 100%;
            height: 600px;
            /* Adjust the height as per your image dimensions */
            z-index: -1;
            /* To ensure the footer content is displayed on top */
        }

        .fixed-top-right {
            position: fixed;
            top: 80px;
            right: 20px;
```

```
        }
    </style>
</head>

<body>

    <nav class="navbar navbar-expand-lg navbar-dark bg-dark" style="padding-top:0;padding-bottom: 0;">
        <a class="navbar-brand" href="{% url 'image_upload' %}"><img src="{% static 'endnoise.png' %}"
                style="height: 50px;" alt=""></a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
            <div class="navbar-nav">
                <a class="nav-link active" href="{% url 'image_upload' %}">Home<span
                        class="sr-only">(current)</span></a>
                <a class="nav-link" href="{% url 'about' %}">About</a>
            </div>
        </div>
    </nav>
    <button type="button" class="btn btn-dark fixed-top-right" data-toggle="modal" data-target="#exampleModal">
        <i class="bi bi-question-lg"></i>
    </button>
    <!-- Modal -->
    <div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title" id="exampleModalLabel">How to use</h5>
                    <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                        <span aria-hidden="true">&times;</span>
                    </button>
                </div>
                <div class="modal-body">
                    1. Upload your image. The file extensions accepted for now are only .jpg, .jpeg, and .png. <br>
                    2. Select a denoising algorithm. <br>
                    3. Depending on your choice, some configurations are needed. There are also default values you can
                    use! <br>
                    4. Denoise and save your denoised image!
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
                </div>
            </div>
        </div>
    </div>


    <script class="jsbin" src="https://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
    <div class="file-upload" style="margin-top: 20px; border: 1px solid black;">
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <div class="mb-3">
                <label class="form-label" for="{{ form.file_field.id_for_label }}">Upload your image</label>
                <input type="file" class="form-control" id="{{ form.file_field.id_for_label }}"
                    name="{{ form.image.name }}">
            </div>

            <div class="mb-3">
                <label class="form-label" for="{{ form.choice.id_for_label }}">Choose an algorithm</label>
```

```
                    {% render_field form.choice class+="form-control" id=form.choice.id_for_label %}
                </div>
                <div id="integer_input">
                    <div class="mb-3">
                        <!-- {{ form.thresh.label_tag }} -->
                        <label class="form-label" for="{{ form.thresh.id_for_label }}">Correlation</label>
                        <!-- <button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="right" title="Tooltip on right">
                        Tooltip on right
                      </button> -->
                        <i class="bi bi-question-circle" data-toggle="tooltip" data-placement="right" title="Adjust this to depending on how
correlated the noise is.
Higher value for higher correlation.
Range: 0 to 255"></i>
                        {% render_field form.thresh class+="form-control" id=form.thresh.id_for_label %}

                        <div class="mb-3">
                            <!-- {{ form.thresh.label_tag }} -->
                            <label class="form-label" for="{{ form.wiener_power.id_for_label }}">Noise level</label>
                            <i class="bi bi-question-circle" data-toggle="tooltip" data-placement="right" title="Adjust this to depending on how
much noise is present on the image.
Higher value for higher amount of noise.
Range: 0 to 1"></i>
                            {% render_field form.wiener_power class+="form-control" id=form.wiener_power.id_for_label %}
                        </div>
                    </div>
                </div>
                <div class="d-flex align-items-center">
                    <button id="upload-button" type="submit" class="btn btn-dark" onclick="handleUpload()">Denoise</button>
                    <div class="ml-auto" id="spinner" role="status" style="display: none;">
                        <div class="spinner-grow" role="status" aria-hidden="true"></div>
                    </div>
                </div>
            </form>
        </div>
        <footer class="footer-with-bg">

        </footer>



        <script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
            integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
            crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-Fy6S3B9q64WdZWQUiU+q4/2Lc9npb8tCaSX9FK7E8HnRr0Jz8D6OP9dO5Vg3Q9ct"
            crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz"
            crossorigin="anonymous"></script>


        <script>
            function handleUpload() {
                // Show the spinner
                $("#spinner").show();

                // Delay the form submission to display the spinner
                setTimeout(function () {
                    $("#upload-form").submit(); // Replace "upload-form" with the ID of your form
```

```
                }, 500); // Adjust the delay duration as needed
            }
            $(document).ready(function () {
                // Function to handle the show/hide logic
                function handleFieldVisibility() {
                    var selectedChoice = $("#algo").val();
                    if (selectedChoice === "wba") {
                        $("#integer_input").show();  // Show the additional fields
                    } else {
                        $("#integer_input").hide();  // Hide the additional fields
                    }
                }

                // Call the function on page load
                handleFieldVisibility();

                // Call the function when the choice field value changes
                $("#algo").on("change", function () {
                    handleFieldVisibility();
                });
            });


            function removeUpload() {
                $('.file-upload-input').replaceWith($('.file-upload-input').clone());
                $('.file-upload-content').hide();
                $('.image-upload-wrap').show();
            }
            $('.image-upload-wrap').bind('dragover', function () {
                $('.image-upload-wrap').addClass('image-dropping');
            });
            $('.image-upload-wrap').bind('dragleave', function () {
                $('.image-upload-wrap').removeClass('image-dropping');
            });

        </script>


    </body>

    </html>
```

XI. Acknowledgment

I would like to express my deepest gratitude and appreciation to all those who have contributed to the successful completion of this project. Without their support, dedication, and expertise, this achievement would not have been possible.

First and foremost, I would like to acknowledge my advisers, Perlita E. Gasmen and Alex C. Gonzaga, for their invaluable guidance, mentorship, and unwavering support throughout the entire duration of this project. Their insightful feedback, constructive criticism, and encouragement have been instrumental in shaping this work.

In addition, I am indebted to my friends and family for their unwavering encouragement, understanding, and belief in me. Their love and support have been a constant source of motivation, pushing me to overcome challenges and strive for excellence.

Lastly, I want to express my appreciation to my blockmates, organizations, and the entire UP Manila community for making my time in this university one of the most meaningful years of my life. Your support, camaraderie, and the opportunities I have received have had a profound impact on my personal and academic growth.

Once again, I extend my heartfelt thanks to everyone involved, directly or indirectly, in this project. Your contributions have made a lasting impact, and I am truly grateful for your unwavering support and dedication.