WM391: Industrial Vision and Processing:
Evaluating Traditional and Convolutional Neural Network (CNN) Image Noise Reduction Algorithms.

Student number: u2191272

Word count: 3597 (excluding title page, contents, abstract, tables, reference list and appendix).

Contents

Abstract

Background

In recent decades, image denoising technology has been rapidly developed, producing a variety of traditional denoising algorithms such as the Gaussian method, mean method, median method, and non-local mean method. These classical algorithms are widely used, however, there are still challenges in maintaining image structural integrity. Therefore, further research on image denoising techniques is necessary. As mathematical knowledge progresses, the algorithms used for denoising will continue to advance. The introduction of neural network technology has been applied in the field of image noise reduction in recent years and has achieved good denoising effect. With the development of artificial intelligence, technologies such as neural network deep learning have been widely applied to various fields. At the same time, image noise reduction technology has been further developed. As a result, some image denoising methods, based on deep learning, have now emerged.

Purpose and Scope

This report investigates the effectiveness of traditional and deep learning approaches for image noise reduction, which focuses on the Bilateral Filter and U-Net Convolutional Neural Network model. Through the application of these methods on a standard dataset, a comparative analysis is conducted to evaluate their performance based on various metrics. The Bilateral Filter is one of the most efficient traditional denoising algorithms with its ability to reduce noise while maintaining image detail. The U-Net model is one of the most effective CNNs at denoising images with complex edges and textures. This comparative report examines and implements each denoising method. The evaluation reveals insights into the strengths and limitations of traditional and deep learning methods in the context of image noise reduction.

Introduction

This report examines traditional and Convolutional Neural Network (CNN) algorithms for reducing noise from images, comparing their effectiveness on the CBSD68 dataset. It focuses on the Bilateral Filter and the U-Net CNN model, evaluating their performance based on edge preservation, computational efficiency, and adaptability to different noise levels and types. To systematically compare these methods, the evaluation includes key performance metrics such as the Structural Similarity Index Measure (SSIM) and Mean Squared Error (MSE), alongside both quantitative and qualitative analyses. This comprehensive evaluation aims to define the effectiveness, strengths, and limitations of traditional versus CNN-based noise reduction techniques. Moreover, multiple approaches are recommended to improve the model's efficiency and robustness, providing insight into advancements in image noise reduction technology.

3624

A) Image Noise Reduction Algorithm (Traditional Methods)

Traditional image noise reduction algorithms are fundamental in the domain of digital image processing, designed to suppress noise from images to preserve the content and integrity of the image. These algorithms implement spatial filtering techniques that adjust image pixel values based on the characteristics of neighbouring pixels to reduce noise. Key traditional methods include mean filtering, Gaussian smoothing, and bilateral filtering, each tailored to reduce different types of noise and image degradation. Based on academic literature, the Bilateral Filter algorithm is the most effective image noise reduction algorithm due to its dual capacity to reduce noise while preserving edges, ensuring image quality remains uncompromised (Tomasi & Manduchi, 1998).

Bilateral Filter

The Bilateral Filter uniquely combines pixel spatial proximity—how close pixels are to each other—and pixel intensity similarity—how similar surrounding pixels are in terms of colour and brightness values. It then applies a Gaussian filter based on both the pixel's location and its intensity. This dual approach enables selective smoothing in areas of similar intensity while maintaining the sharpness of edges, where pixel intensity values change significantly. Unlike mean or Gaussian filters, which apply uniform smoothing across all pixels, the Bilateral Filter adjusts its effect based on the characteristics of each pixel's neighbourhood which maintains essential textures and edges crucial for the visual fidelity of the image (Banterle et al., 2011).

The equation for the bilateral filter is:

$$I'(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) e^{-\left(\frac{(I(x_i)-I(x))^2}{2\sigmaColor^2} + \frac{\|x_i-x\|^2}{2\sigmaSpace^2} \right)}$$

Equation 1: Bilateral Filter formula

where:

- ***I'(x)*:** Output pixel value.
- ***I(xi)*:** Input pixel value in the neighbourhood Ω around ***x***.
- ***Wp*:** Normalisation factor ensuring the weights sum to one.
- **\sigmaColour:** Controls how much the filter blurs similar colours.
- **\sigmaSpace**: Dictates how far to blend pixel values based on distance.

Implementation

The bilateral filter function is applied to a set of images to reduce noise and preserve edges. Firstly, the code iterates over a range of parameters, the diameter of the pixel area (d), the filter sigma in the colour space (sigmaColour), and the filter sigma in the coordinate space (sigmaSpace).

```python
# Function - process images and calculate similarities
def process_images(dataset_path, original_folder):
    # Generate parameter sets
    d_values = range(3, 10,2)  # d values from 3 (as mimimum kernal size) to 9 in steps of two for even kernel
    sigma_values = range(0, 300,20)  # sigmaColor and sigmaSpace values from 1 to 300

    total_iterations = len(d_values) * len(sigma_values) * len(sigma_values)
    current_iteration = 0

    # Initialize dictionary to store average similarities for each parameter
    all_average_similarities = {}
    # Iterate through each parameter set
    for d in d_values:
        for sigmaColor in sigma_values:
            for sigmaSpace in sigma_values:

                current_iteration += 1
                completion_percentage = (current_iteration / total_iterations) * 100

                # For each parameter set, process images and calculate similarities
                print(f"Processing with parameters: d={d}, sigmaColor={sigmaColor}, sigmaSpace={sigmaSpace} - {completion_percentage:.2f}% complete")
                similarities = []  # Initialize an empty list for each set of parameters
                noise_levels = ['noisy5', 'noisy10', 'noisy15', 'noisy25', 'noisy35', 'noisy50']
                # Iterate through each noise level
                for noise_level in noise_levels:
                    path = os.path.join(dataset_path, noise_level)
                    for img_filename in os.listdir(path):
                        noisy_image_path = os.path.join(path, img_filename)
                        original_image_path = os.path.join(dataset_path, original_folder, img_filename)

                        # Check if the original image exists
                        if os.path.exists(original_image_path):
                            filtered_image = apply_bilateral_filter(noisy_image_path, d, sigmaColor, sigmaSpace)
                            original_image = cv2.imread(original_image_path)
                            similarity = calculate_similarity(filtered_image, original_image)
                            similarities.append((similarity, img_filename, noise_level))
                        else:
                            print(f"Original image for {img_filename} not found.")
                # Ensure there are similarities calculated before trying to compute average
                if similarities:
                    avg_similarity = sum([item[0] for item in similarities]) / len(similarities)
                    print(f"Average similarity for parameters (d={d}, sigmaColor={sigmaColor}, sigmaSpace={sigmaSpace}): {avg_similarity:.2f}")
                    all_average_similarities[(d, sigmaColor, sigmaSpace)] = avg_similarity

    return all_average_similarities
```

Figure 1:Identify optimal parameters.

It then applies the bilateral filter to each image using these parameters.

The ‘calculate_similarity’ function assesses the denoised image against the original to generate a similarity score. The lower similarity scores indicate better denoising. The data is stored in a csv and plotted on a 4D graph.

	A	B	C	D
1	d	sigmaColour	sigmaSpace	average_similarity
2	7	120	20	-7.978594714
3	7	120	40	-7.979728555
4	7	120	60	-7.979936485
5	7	120	80	-7.980011633
6	7	120	100	-7.980045826
7	7	120	120	-7.980065462
8	7	120	140	-7.980076712
9	7	120	160	-7.980084204
10	7	120	180	-7.980088861

Table 1: First 10 results from Bilateral Filter.

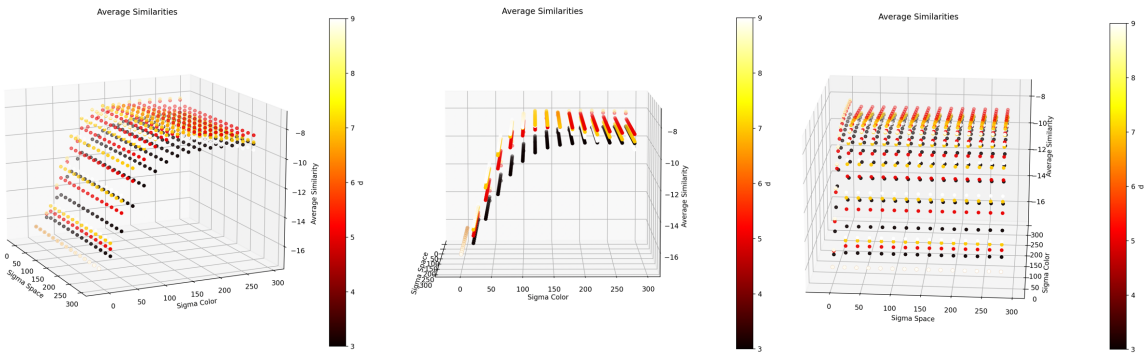

Figure 2: 4D graph of average similarities.

The results in Table 1demonstrate the optimum parameters as a higher similarity score indicates less deviation from the original image. For the average similarity values: d=7, sigmaColour=120, sigmaSpace=20. From sigmaSpace values 20 to 180, it is observed that the average similarity increases as the sigmaSpace value increases. This indicates that as the spatial component of the filter becomes more influential, the denoised image is more similar to the original image. This suggests a larger sigmaSpace is better for reducing noise and preserving edges.

Justification for Selection

The Bilateral Filter was implemented as the best suited traditional noise reduction algorithm due to its efficiency in reducing the Additive White Gaussian Noise (AWGN) (Smolka et al., 2023) applied in the CBSD68 dataset. There are four main justifications for employing this algorithm. Edge Preservation preserves edges from the image. This is crucial for maintaining the detail and sharpness of natural images in the CBSD68 dataset, which feature distinct boundaries and contrasts inherent in real-world scenes.

Selective Smoothing differentiates between noise and key image features through spatial and intensity-based filtering. This preserves high-frequency details, such as edges and textures, and ensures pixels with similar intensity levels to the target pixel are smoothed to keep image quality. The Bilateral Filter is designed for effective reduction of AWGN, which is the specific noise type present in the CBSD68 dataset images. Therefore, it has good compatibility with AWGN. The CBSD68 dataset's varied natural images benefit from the Bilateral Filter's versatility which ensures consistent noise reduction performance across different types of images from natural landscapes to architectural structures.

B) Current Status of Literature on Deep Neural Networks (DNNs) for Image Noise Reduction

The landscape of image noise reduction has dramatically evolved from traditional noise reduction algorithms to the introduction of deep learning techniques, through the development of specialised DNNs. These deep learning based neural networks have significantly improved on the precision and efficiency of noise reduction. The current literature highlights the effectiveness of DNNs in handling complex noise patterns, such as AWGN (Zhou et al., 2020), prevalent in the CBSD68 dataset. This is achieved through advanced feature-learning capabilities to enable the model to learn hierarchical features that robustly counteract AWGN distortions. CNNs have been particularly influential, demonstrating greater adaptability and performance in detecting and mitigating noise while preserving image details (Zhang et al., 2017).

Current literature emphasises the importance of tailored loss functions in training DNNs for noise reduction, including MSE, Perceptual Loss, and SSIM (Zhao et al., 2017). Performance metrics like Peak Signal-to-Noise Ratio (PSNR) and SSIM are fundamental to these analyses, as they provide quantitative insights into how DNNs improve noise reduction and retain details. This capability is pertinent in domains where visual accuracy is key, such as medical imaging/diagnostics (Gogineni & Chaturvedi, 2022).

Despite the advancements in image noise reduction, the high computational cost of training and deploying DNNs is a challenge, particularly for real-time applications. Additionally, the need for large volumes of training data is a limitation in situations where data is scarce or costly to source (Bansal et al., 2022). The future research is focused on developing more efficient DNN architectures which employ transfer learning and exploring few-shot and zero-shot learning to reduce dependency on large datasets, enabling DNNs to learn from minimal data (Safonova et al., 2023).

C) CNN Network Architecture Implemented

The U-Net architecture was selected as the CNN model for the image noise reduction in the dataset CBSD68. The U-Net was chosen over other denoising neural networks (for example, DnCNN), for its high capability to preserve details in images with complex features, as found in the CBSD68 dataset (Komatsu & Gonsalves, 2020). While DnCNN effectively denoises images and is simple to implement, U-Net's architecture utilises skip connections and receptive fields to retain high detail and texture information that are crucial for high-quality denoising.

U-Net Model Architecture

The U-Net architecture implemented for denoising the CBSD68 dataset is characterised by its 'U' symmetric shape, which follows an encoder-decoder structure.

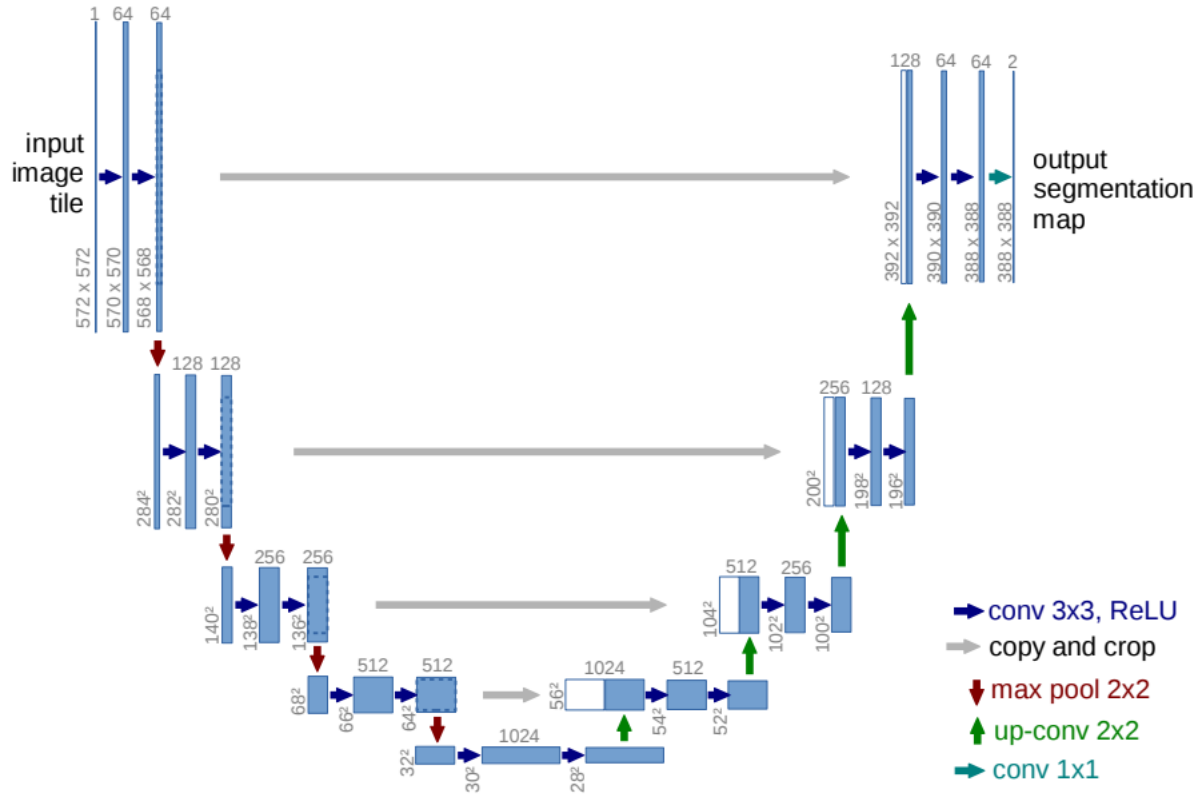

Figure 3: U-Net encoder, decoder model type architecture

(Ronneberger et al., 2015)

Encoder

The encoder, or a contraction path, is comprised of a sequence of convolutional and pooling layers. It systematically compresses and reduces the spatial dimensions of the image while increasing the feature channels (the depth) where the context of the input data is captured. This enables the CNN to encode higher-level features such as edges and noise patterns within the images (Couturier et al., 2018).

In the code, the encoder pathway is established through a series of Down operations, each consisting of a pooling operation, followed by two convolutional operations (DoubleConv class).

```
self.down1 = Down(64, 128)
self.down2 = Down(128, 256)
self.down3 = Down(256, 512)
self.down4 = Down(512, 512)
```

Figure 4: Encoder 'Down' convolutions

Decoder

The decoder, or expansion path, reconstructs the spatial dimensions and reduces the feature channels (the depth). This is through a sequence of Up convolutions and concatenations, with the corresponding Up sampled encoder feature maps determined from the skip connections. The upscaling process uses the feature channels (depth) context data to restore the high detailed edges and textures to create a clean, denoised image.

```
self.up1 = Up(1024, 256)
self.up2 = Up(512, 128)
self.up3 = Up(256, 64)
self.up4 = Up(128, 64)
```

Figure 5: Encoder 'Up' convolutions

Skip Connections

The skip connections between the encoder and decoder provide a direct pathway, bypassing intermediate layers for information flow. This allows the network to propagate context and spatial information across the network layers, which assists in preserving critical features like edges during denoising (Couturier et al., 2018).

```
x = torch.cat([x2, x1], dim=1)
```

Figure 6: Skip Connections

The bottleneck

The bottleneck at the bottom of the 'U' structure links the encoder and decoder. It downsizes the feature dimensions and then parses the image through the 3x3 convolutional layers, and the non-linear activation function (ReLU) is applied. Then the upscaling process is initiated, re-expanding the feature dimensions to reconstruct the denoised image with a high quality.

```
x5 = self.down4(x4)
x = self.up1(x5, x4)
```

Figure 7: The bottleneck

The U-Net model has been implemented from the official Pytorch Github and referenced in the code (Milesial, 2022).

D) Loss Functions Used

In the context of CNNs used for image noise reduction, the loss function is a benchmark value to evaluate the performance of the model. Initially, MSE was employed as the loss function due to its simplicity and ease of implementation. MSE directly measures the average squared difference between the predicted values and the actual pixel value in the image.

Despite achieving a low loss value of 0.0014, which suggests minor differences between the denoised and original image, there were visual anomalies (Figure 8). These anomalies occurred as MSE is calculated by calculating a cumulative average of errors across individual pixels - hence it can miss localised discrepancies. This results in a misleading low MSE value, despite visual issues in the denoised image, as shown in Figure 8.
Therefore, the SSIM was implemented to replace MSE as a more complex, yet better-suited function for image denoising tasks. SSIM considers multiple changes in texture, contrast, and luminance which focuses on the preservation of perceptual and structural aspects of images in high detail. Although the loss value shown in Figure 11 with SSIM was higher at 0.114, which typically indicates a larger difference between the denoised and original image, SSIM considers perception and saliency (human visual perception sensitivity) rather than just absolute error (Wang et al., 2004). This resulted in a clearer denoised image. The same training model, level of noise, and hyperparameters was used.

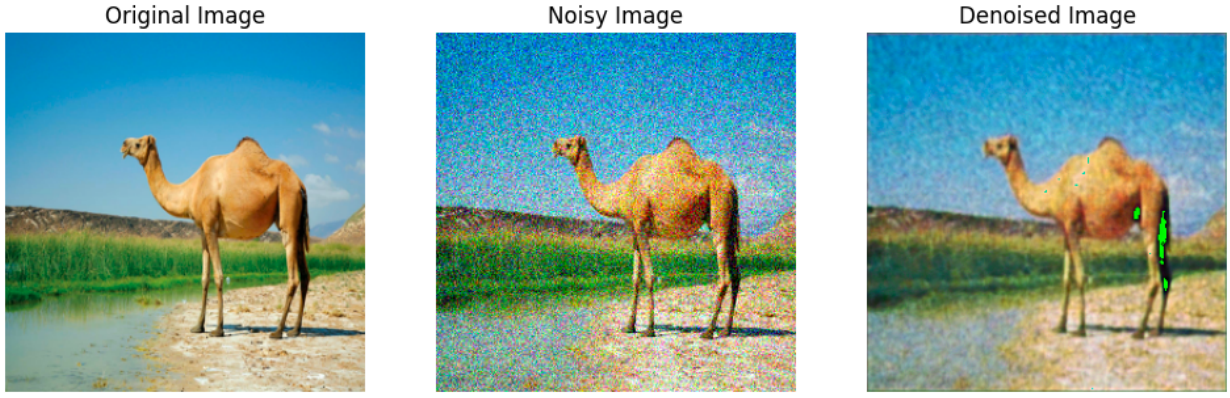

Figure 8: U-Net image with MSE.

Figure 9: U-Net image with SSIM.

```
Epoch 1/1003, Loss: 0.2812195143529347
Epoch 2/1003, Loss: 0.15501238405704498
Epoch 3/1003, Loss: 0.09148435720375606
Epoch 4/1003, Loss: 0.04889643724475588
Epoch 5/1003, Loss: 0.028730235461677824
Epoch 6/1003, Loss: 0.025549343521041528
Epoch 7/1003, Loss: 0.020151450032634393
Epoch 8/1003, Loss: 0.01391387078911066
Epoch 9/1003, Loss: 0.012519765379173415
Epoch 10/1003, Loss: 0.009989898252700056
Epoch 11/1003, Loss: 0.009993237044130052
Epoch 12/1003, Loss: 0.00817383938868131
Epoch 13/1003, Loss: 0.011268969691757644
Epoch 14/1003, Loss: 0.008099309767463378
Epoch 15/1003, Loss: 0.007207571994513273
Epoch 16/1003, Loss: 0.007341832521238497
Epoch 17/1003, Loss: 0.00820148576583181
Epoch 18/1003, Loss: 0.0073814370137240 2
Epoch 19/1003, Loss: 0.006061425698654992
Epoch 20/1003, Loss: 0.005370431779218572
Epoch 21/1003, Loss: 0.006625905599711197
Epoch 22/1003, Loss: 0.00717800938790398
Epoch 23/1003, Loss: 0.005905630399606058
Epoch 24/1003, Loss: 0.004549205469499741
Epoch 25/1003, Loss: 0.0046949272177049094
...
Epoch 1002/1003, Loss: 0.0014301743275219841
Epoch 1003/1003, Loss: 0.0014887601802391665
Finished Training
Losses pickled and saved to data/losses1_epochs_{1003}.pkl
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 10: U-Net Loss value with MSE.

```
[4]  ✓ 34m 17.9s
Epoch 1/1001, Loss: 0.8963826469012669
Epoch 2/1001, Loss: 0.6873525636536735
Epoch 3/1001, Loss: 0.5718549915722438
Epoch 4/1001, Loss: 0.49885293415614534
Epoch 5/1001, Loss: 0.4712976132120405
Epoch 6/1001, Loss: 0.41810994488852365
Epoch 7/1001, Loss: 0.4058194160461426
Epoch 8/1001, Loss: 0.37845143250056673
Epoch 9/1001, Loss: 0.33607953786849976
Epoch 10/1001, Loss: 0.320769088608878
Epoch 11/1001, Loss: 0.37714061566761564
Epoch 12/1001, Loss: 0.37192260367529734
Epoch 13/1001, Loss: 0.3411952682903835
Epoch 14/1001, Loss: 0.33139475754329134
Epoch 15/1001, Loss: 0.3091849684715271
Epoch 16/1001, Loss: 0.2921732323510306
Epoch 17/1001, Loss: 0.3083981786455427
Epoch 18/1001, Loss: 0.3015848057610648
Epoch 19/1001, Loss: 0.27330654859542847
Epoch 20/1001, Loss: 0.27352250473839895
Epoch 21/1001, Loss: 0.26521687848227365
Epoch 22/1001, Loss: 0.2356513057436262
Epoch 23/1001, Loss: 0.26107102632522583
Epoch 24/1001, Loss: 0.2848594103540693
Epoch 25/1001, Loss: 0.26992411272866385
...
Epoch 1000/1001, Loss: 0.11839973075049263
Epoch 1001/1001, Loss: 0.1148504274232047
Finished Training
Losses pickled and saved to data/losses6_epochs_{1001}.pkl
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 11: U-Net Loss value with SSIM.

In the code, implementation of SSIM defines a Gaussian window to simulate human visual perception sensitivity. This is then used to compare the similarity between original and denoised images. This formula captures the luminance, contrast, and structure comparison functions, which provides a more accurate representation of the image quality.

```python
class SSIMLoss(torch.nn.Module):
    # Class for SSIM loss to integrate SSIM as loss function
    def __init__(self, window_size=11, window_sigma=1.5, size_average=True, channel=3):
        super(SSIMLoss, self).__init__()
        self.window_size = window_size
        self.window_sigma = window_sigma
        self.size_average = size_average
        self.channel = channel

    def forward(self, img1, img2):
        # 1 - SSIM to minimize distance (loss)
        return 1 - ssim(img1, img2, self.window_size, self.window_sigma, self.size_average, self.channel)
```

Figure 12: Calculating SSIM value.

The SSIM calculations ensure perceptual quality, essential for the CBSD68 dataset, as it requires detail feature preservation. The higher SSIM loss value, versus the lower MSE, reflects the model's focus on perceptual rather than pixel-level accuracy. Consequently, the U-Net model equipped with SSIM as its loss function, produces denoised images that are more aligned with human perception of quality and detail, despite what the MSE loss value implies.

E) Training and Testing Process

The training process is the phase where the U-Net model learns to denoise images by adjusting its weights according to the loss function. The process iterates over the dataset in epochs, which are complete passes through the entire training dataset.

Dataset Utilisation and Pre-processing

The CBSD68 dataset consists of 68 images with complex features and various levels of noise. Initially, training the model directly with this limited set of images led to poor performance, as the dataset diversity was insufficient for the U-Net CNN to learn and generalise denoising patterns effectively. To enhance the variability and robustness of the CBSD68 dataset, pre-processing steps including image transformations and augmentations were implemented.

```
# transformations applied to original image to create a varied traiining dataset
transformations = transforms.Compose([
    transforms.RandomRotation((0, 360)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomResizedCrop((256, 256), scale=(0.8, 1.0), ratio=(0.9, 1.1)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.RandomGrayscale(p=0.1),
    transforms.RandomPerspective(distortion_scale=0.2, p=0.1),
    transforms.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=10),
    transforms.Resize((256, 256))
])
```

Figure 13: Image transformations and augmentations.

However, it was observed that applying data transformations and augmentations directly to the noisy images inadvertently led to the U-Net CNN learning the transformed noise patterns, rather than the underlying noise reduction features. This confusion occurred due to the noise undergoing transformations with the image, resulting in inconsistent noise patterns that the model struggled to generalise. To solve this, transformations were applied only to the ‘original_png’ images, and then AWGN noise was added. This ensured the model will learn to denoise from a consistent noise distribution, irrespective of the content transformations, which results in more effective training and better generalisation in the denoised images.

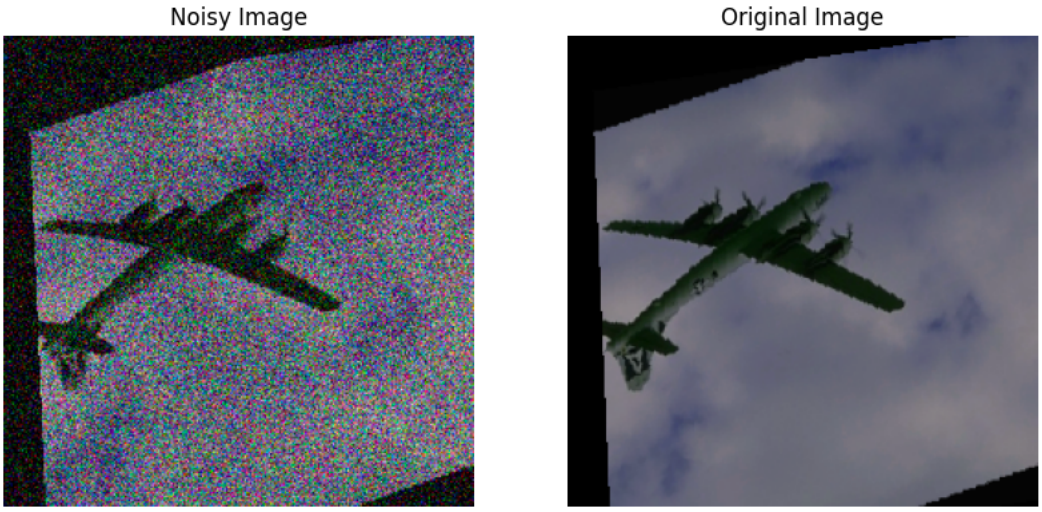

Figure 14: Image 0 in CBSD68 dataset with random AWGN noise and transformations.

Hyperparameters

Hyperparameters are the variables set before training the U-Net CNN to dictate how the network behaves and learns. The selection of hyperparameters such as batch size, number of epochs, and learning rate was tailored to optimise the U-Net CNN's performance.

Batch size is the number of training samples the network processes before updating its internal weight values. A small batch size (a small subset of the data), results in the model updating the weight values more frequently which introduces more variability in the data, leading to better generalisation. A large batch size (a larger subset of the data) processes more data at once which results in the model updating the weight values less frequently. This results in stable and reliable gradient descent steps, as there is more information included in the gradient calculation. For the CBSD68 dataset, the data is complex, and so the optimal batch size was assigned eight (‘batch_size=8’), for a balance between computational efficiency and model generalisation.

An epoch is one full cycle through the entire training dataset. Increasing the number of epochs gives the model more iterations to predict, compare, and adjust its weight values to reduce the loss value. However, too many epochs will result in overfitting, where the model is overspecialised on the data, reducing the ability to perform well on new data. For the CBSD68 dataset, ‘n_epochs=1000’ providing the model time to train from the dataset. The model is monitored to avoid overfitting.

The learning rate (lr) is a set value used to adjust the weights of the network to determine the step size during gradient descent. A large learning rate can result in the model to optimise (converge to its minima) quicker. However, if the lr value is too large the model will overshoot the minima, preventing convergence. A lower lr rate ensures smaller adjustments to the weights, resulting in a smooth convergence at the minima. But if the lr rate is too small, the model may

take an excessively long time to train. For the CBSD68 dataset, the learning rate is set at 0.0001, to guarantee smooth and precise weight adjustments for steady convergence.

Optimiser Function

The optimisation function implemented in the U-Net CNN was Adaptive Moment Estimation (Adam). It combines AdaGrad and RMSProp algorithms to manage sparse gradients on noisy problems (Chaudhury & Yamasaki, 2021). Adam was selected for the CBSD68 dataset due to its adaptive capability dynamically adjusting the learning rate essential for the diverse and complex image features. By modularly adapting its learning rate, Adam ensures a more effective and stable convergence, avoiding problems like overshooting the minimum or getting stuck in local optima (Chaudhury & Yamasaki, 2021).

Key Features of Adam Optimiser:

Adam dynamically adapts the learning rate for each parameter based on the estimates of first (mean) and second (uncentered variance) moment estimates of the gradients. This results in more effective parameter tuning and efficiency of the model.

Moment Estimation: Adam calculates and maintains two moving averages for each parameter.

1. The average of the gradients, which determines the direction and momentum of weight updates.
2. The squared average of the gradients, which determines the scale (size) of the learning rate is adjusted.

This facilitates dynamic learning rate adjustments enabling quicker convergence to the minima.

Benefits for applying to CBSD68 dataset:

Adam's adaptability is effective for the CBSD68 dataset as complex image features require precise learning rate adjustments (Alzubaidi et al., 2021). This enables stable, efficient learning, and sidestepping issues such as overshooting or local optima entrapment.

Implementation:

Adam is initialised with a lr of 0.0001, for precise weight value adjustments, crucial for denoising in detail. Adam systematically updates the model's weights in each epoch, driven by SSIM-based loss gradients.

Testing Methodology

After training the U-Net model, a testing methodology is employed to evaluate the model's performance on unseen test data. This process is crucial for assessing the denoising capabilities and the model's generalisation capabilities.

Dataset Split

The dataset was partitioned in an 80/20 split to ensure a substantial amount of data is used for model training, while reserving 20% of the dataset for testing the model.

Model Testing Process

First, the test dataset completed the same pre-processing and normalization steps as the training dataset to maintain consistency.

Next, the model was switched to evaluation mode (model.eval()) to disable dropout layers and batch normalisation to ensure consistency in predictions.

```
# Set model to evaluation mode
model.eval()
```

Figure 15: Set model to evaluation mode.

The SSIM was used as the metric to evaluate the denoised images against the original images to preserve perceptual image quality. The resulting value is the loss function value.

```
criterion = SSIMLoss()
```

Figure 16: U-Net test loss value with SSIM.

Lastly, in addition to SSIM, the image denoising model was tested using other relevant metrics (including MSE) to provide a comprehensive understanding of the U-Net model's effectiveness.

F) Model Evaluation Analysis

Following the testing phase, detailed evaluation analysis was conducted to interpret the results from the testing. The quantitative and qualitative analysis provides insight into the model's strengths and limitations.

Quantitative Analysis

The model achieved a training SSIM loss function value of 0.0023 and a test SSIM loss function value of 0.0011 on the test dataset, indicating a high degree of similarity between the denoised images and their originals. This score surpasses the performance of traditional denoising methods and competes well with contemporary deep learning approaches.

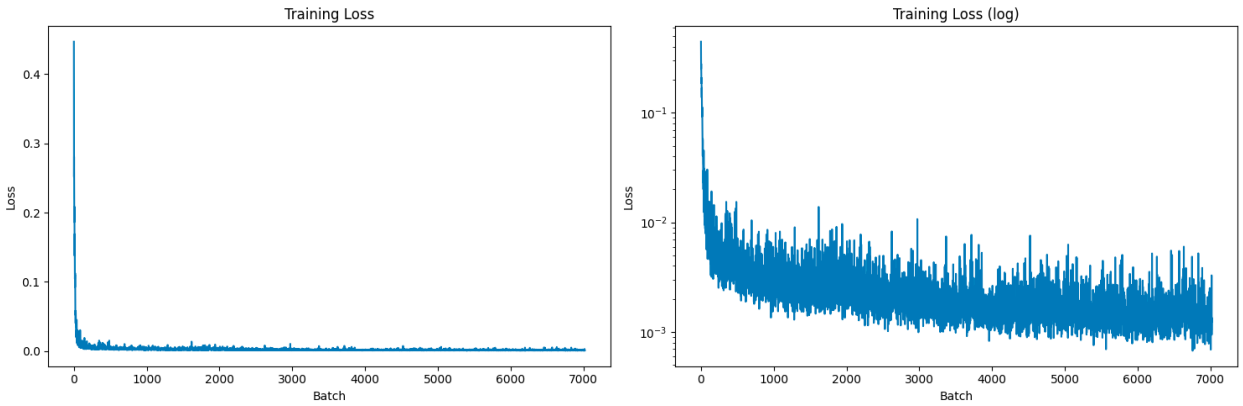

Figure 17: Graphical view of training loss function values.

Test Trained Model

```
# load test dataset
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

#old loss function => criterion = nn.MSELoss()


# Set model to evaluation mode
model.eval()


criterion = SSIMLoss()
# criterion = nn.MSELoss()

# run test dataset through evalutation mode and calculate loss
test_losses = []
with torch.no_grad():
    for batch in test_loader:
        noisy_img, original_img = batch
        noisy_img, original_img = noisy_img.to(device), original_img.to(device)

        output = model(noisy_img)
        loss = criterion(output, original_img)
        test_losses.append(loss.item())

average_loss = sum(test_losses) / len(test_losses)
print(f"Average loss on test dataset: {average_loss}")
```

[28] ✓ 3.4s Python

```
Average loss on test dataset: 0.0011513171484693885
```

Figure 18: Test loss function value.

Qualitative Analysis

Visually, the denoised images, in comparison to the original images, highlight the model's capability to retain textures and edges, crucial for perceptual quality. In the code, users can upload new unseen .png images to the model. AWGN is applied to these images and then the trained model applies the denoising algorithm to remove the noise. This allows users to observe the model's denoising effectiveness on any given .png image. Example random images with varying amounts of noise are presented below.

Figure 19: Random images from the web used to test the U-Net model.

Strengths and Limitations

The model demonstrates strengths in convergence and stability, shown by consistent decreasing loss values, suggesting that the training process is converging. Plus, a gradual reduction in loss per epoch suggests the training process is stable. However, without validating the loss values, overfitting is still possible.

G) Possible Alternative Approaches

Based on the current findings and implementations of this report, these approaches could be implemented to improve the model's efficiency and robustness.

Split: Optimal method for data splitting

In the implementation of the U-Net model, an 80/20 split was used. However, the SPlit method uses support points to represent the CBSD68 dataset distribution for training and testing. This will capture the dataset's characteristics more efficiently, enhance model generalisation, and reduce overfitting (Joseph & Vakayil, 2021).

Generative Adversarial Networks (GANs)

Incorporating GANs into the U-Nets model training will create varied, synthetic noisy images. This would provide a diverse training set and refine the model's capability of denoising complex noise patterns (Alqahtani et al., 2019).

Advanced Loss Functions

Implementing perceptual-based loss functions like Perceptual Loss or Feature Loss can make the denoised output more visually accurate by emphasising high-level features specific in human perception (Johnson et al., 2016).

Expanding Dataset Size and Diversity

Enriching the training set with a broader array of images and noise types, possibly through augmentation and synthetic data, would provide a wider learning scope for the model.

E) Traditional vs CNN Image Noise Reduction Algorithms

Each approach is evaluated based on the key criteria: edge preservation, computational efficiency, and generalisation.

Edge Preservation

Traditional methods, like the implemented bilateral filter, excel in preserving edges and therefore image details by weighting pixel values based on spatial proximity. However, the U-Net architecture improves on this capability through the ability to learn and enhance edges and textures contextually. The U-Net's skip connections reintroduce lost details from earlier layers which enables reconstruction of high-quality images with edge definition.

Computational Efficiency

The bilateral filter requires much less computational power than the U-Net architecture which makes it suitable for real-time applications. U-Net's high quality, high complexity architecture requires increased computational demands. Its deep learning capability necessitates substantial processing power and time, specifically when training the model not making it possible for all practical applications.

Generalisation

The generalisation of an algorithm is determined by its ability to perform consistently across different data and noise. The bilateral filter has a low generalisation the algorithm relies on predefined rules for noise reduction, resulting in low adaptation to other noise variations without manual adjustments. In contrast, the U-Net model is trained on diverse datasets and can learn a wide range of noise patterns. Its performance improves with the variety and volume of training data, demonstrating better model generalisation.

Conclusion

The analysis within this report provides an understanding of noise reduction algorithms, evaluating the capabilities and limitations of both traditional and modern CNN-based techniques.

The bilateral filter's strength lies in its simplicity and effectiveness in edge preservation. It specialises at denoising certain noise like AWGN, but it struggles to adapt to varied noise without adjustments. In contrast, the U-Net, by learning from data, excels in removing diverse noise types and preserving complex image details. However, its computational intensity and complexity may limit its use in real-time applications.

The decision to utilise the bilateral filter and U-Net is dependent on the available computational resources and complexity of the noise. For areas where accuracy is paramount, such as in medical imaging, U-Net offers significant advantages.

Future research is aimed towards enhancing the efficiency of CNNs, making them more accessible in wider industries with limited computational capacity.

References

Alqahtani, H., Kavakli-Thorne, M. and Kumar, G. (2019) 'Applications of generative Adversarial Networks (Gans): An updated review', *Archives of Computational Methods in Engineering*, 28(2), pp. 525–552. doi:10.1007/s11831-019-09388-y.

Alzubaidi, L. *et al.* (2021) 'Review of Deep Learning: Concepts, CNN Architectures, challenges, applications, Future Directions', *Journal of Big Data*, 8(1). doi:10.1186/s40537-021-00444-8.

Bansal, Ms.A., Sharma, Dr.R. and Kathuria, Dr.M. (2022) 'A systematic review on data scarcity problem in deep learning: Solution and applications', *ACM Computing Surveys*, 54(10s), pp. 1–29. doi:10.1145/3502287.

Banterle, F. *et al.* (2011) 'A low-memory, straightforward and fast bilateral filter through subsampling in Spatial Domain', *Computer Graphics Forum*, 31(1), pp. 19–32. doi:10.1111/j.1467-8659.2011.02078.x.

Chaudhury, S. and Yamasaki, T. (2021) 'Robustness of adaptive neural network optimization under training noise', *IEEE Access*, 9, pp. 37039–37053. doi:10.1109/access.2021.3062990.

Couturier, R., Perrot, G. and Salomon, M. (2018) 'Image denoising using a deep encoder-decoder network with skip connections', *Neural Information Processing*, pp. 554–565. doi:10.1007/978-3-030-04224-0_48.

Gogineni, R. and Chaturvedi, A. (2022) 'Convolutional Neural Networks for medical image analysis', *Convolutional Neural Networks for Medical Image Processing Applications*, pp. 75–90. doi:10.1201/9781003215141-4.

Gonzalez, R.C., Woods, R.E. and Masters, B.R. (2009) 'Digital Image Processing, third edition', *Journal of Biomedical Optics*, 14(2), p. 029901. doi:10.1117/1.3115362.

Johnson, J., Alahi, A. and Fei-Fei, L. (2016) 'Perceptual losses for real-time style transfer and Super-Resolution', *Computer Vision – ECCV 2016*, pp. 694–711. doi:10.1007/978-3-319-46475-6_43.

Joseph, V.R. and Vakayil, A. (2021) 'Split: An optimal method for data splitting', *Technometrics*, 64(2), pp. 166–176. doi:10.1080/00401706.2021.1921037.

Komatsu, R. and Gonsalves, T. (2020) 'Comparing U-Net based models for denoising color images', *AI*, 1(4), pp. 465–487. doi:10.3390/ai1040029.

Milesial (2022) *Milesial/Pytorch-UNet: Pytorch implementation of the U-Net for image semantic segmentation with high quality images*, *GitHub*. Available at: https://github.com/milesial/Pytorch-UNet (Accessed: 18 March 2024).

Ronneberger, O., Fischer, P. and Brox, T. (2015) 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *Lecture Notes in Computer Science*, pp. 234–241. doi:10.1007/978-3-319-24574-4_28.

Safonova, A. *et al.* (2023) 'Ten deep learning techniques to address small data problems with remote sensing', *International Journal of Applied Earth Observation and Geoinformation*, 125, p. 103569. doi:10.1016/j.jag.2023.103569.

Smolka, B., Kusnik, D. and Radlak, K. (2023) 'On the reduction of mixed Gaussian and impulsive noise in heavily corrupted color images', *Scientific Reports*, 13(1). doi:10.1038/s41598-023-48036-1.

Tomasi, C. and Manduchi, R. (1998) 'Bilateral filtering for gray and color images', *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)* [Preprint]. doi:10.1109/iccv.1998.710815.

Wang, Z. *et al.* (2004) 'Image quality assessment: From error visibility to structural similarity', *IEEE Transactions on Image Processing*, 13(4), pp. 600–612. doi:10.1109/tip.2003.819861.

Zhang, K. *et al.* (2017) 'Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising', *IEEE Transactions on Image Processing*, 26(7), pp. 3142–3155. doi:10.1109/tip.2017.2662206.

Zhao, H. *et al.* (2017) 'Loss functions for image restoration with Neural Networks', *IEEE Transactions on Computational Imaging*, 3(1). doi:10.1109/tci.2016.2644865.

Zhou, Y. *et al.* (2020) 'When AWGN-based Denoiser meets real noises', *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07), pp. 13074–13081. doi:10.1609/aaai.v34i07.7009.

Appendices (if any)