Computer Vision Project:

Reduce Noisy Images

by

Amirreza Mansour 40012358040

Arsham Masoudi 40012358039

Department of Computer Engineering, Buali Sina University

Dr. Hassan Khotanlo

February, March 2025

Table of contents

3.. Abstract .1
4...Introduction .2
5..Preprocessing .3
6...Split data to train and test .3.1
6... Transform for Detection Model .3.2
7...train step .4
7.. Noise Type Detection .4.1
9... Noise Reduction .4.2
9...Gaussian Noise Reduction .4.2.1
10..................................Salt and Pepper Noise Reduction .4.2.2
12..Periodic Noise Reduction .4.2.3
12.. Test Step .5
12...Label Detection test .5.1
12.. Confusion Matrix .5.1.1
14... Accuracy .5.1.2
14... F1 score .5.1.3
14... Precision .5.1.4
14... Specificity .5.1.5
14... Recall .5.1.6
14...Periodic autoencoder test .5.2
15.. Conclusion .6
16.. References .7

1. Abstract

In this project, the goal is to detect and remove noise from images using a combination of deep learning and image processing techniques. In the first stage, a Convolutional Neural Network (CNN) was trained to identify the type of noise in the images. The dataset used consists of 1,536 noisy images and their corresponding noise-free versions, along with noise type labels in a CSV file. The types of noise examined include salt-and-pepper, Gaussian, and periodic noise, each with distinct characteristics in the spatial and frequency domains.

After detecting the type of noise, image preprocessing methods were employed to remove or reduce it. For salt-and-pepper noise, which appears as random black and white dots in the image, we utilized a median filter that can effectively eliminate impulse noise without losing image details. For Gaussian noise, which is typically caused by imaging sensors and poor lighting conditions, the BM3D algorithm was used. This method is based on processing groups of similar image blocks and principal component analysis (PCA), demonstrating high performance in noise reduction.

To remove periodic noise, which usually appears as regular patterns in the frequency domain, we employed an autoencoder. This deep learning model learns the noise patterns by receiving noisy images and their noise-free counterparts, allowing it to reconstruct the original image. To evaluate the performance of the models, the Peak Signal-to-Noise Ratio (PSNR) metric was used. Comparing the PSNR

values before and after applying various filters indicated an improvement in image quality and effective noise reduction.

The results obtained demonstrate that the combination of deep learning and image processing techniques can provide a powerful solution for reconstructing images contaminated with various types of noise.

2. Introduction

Today, images are an integral part of our lives, and many new technologies owe their existence to images. However, images are not always captured and stored as we desire; they undergo changes that can alter their nature. This change in nature can be misleading for AI models that rely on images, as well as for humans themselves. Therefore, we need a way to improve and restore images to their original state.

In this project, we aim to enhance image quality using computer vision and machine learning techniques. The model developed in this project takes a noisy image as input and returns a less noisy version. It is important to note that AI models never achieve 100% efficiency and are always subject to errors.

This model is capable of detecting and correcting three types of noise: Gaussian, salt-and-pepper, and periodic noise. The dataset used for this project consists of 1,566 images, with corresponding noise-free images also available. Each image is labeled individually using a CSV file, specifying the type of noise present.

For coding this model, we used Google Colab and the T4 GPU. To implement the deep learning part, we utilized the PyTorch library in the Python programming language.

3. Preprocessing

Our dataset has 1566 images and we have noised images and without noise images. On the other hand we have labels for all of these images in the CSV file.

We uploaded the images to Google Drive and used them in Google Colab. To access them, you need to use the library `from google.colab import drive`. The main preprocessing tasks we performed include the following:

3.1. Split data to train and test

The total number of images available to us was 1,566. To split the data into train and test sets, we set aside 20 percent of images, ensuring they included all types of noise. These images were stored in a separate file to prevent the model from learning from them, so they could be used exclusively for testing at the end.

3.2. Transform for Detection Model

In the first step, we resize the images to 256 by 192. This value was determined through trial and error; if we reduce it, it may not work well for high-frequency noise, and if we increase it, we may encounter out-of-memory issues. Therefore, we need to maintain a trade-off between these factors.

At this stage, we only want to identify the type of noise in the images without making any changes to them. For this reason, we can use the `ColorJitter` function to enhance the images in terms of brightness and contrast.

Finally, we normalize the images with a mean of 0.5 and a standard deviation of 0.5. A better approach would have been to derive these parameters from the images themselves, but using fixed values also yields good results.

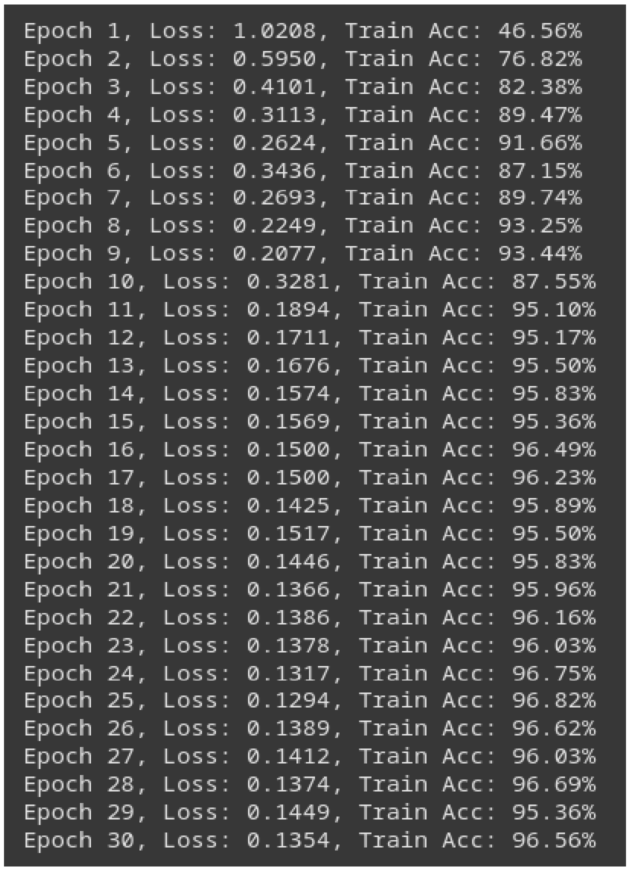
4. train step

4.1. Noise Type Detection

The first step in noise removal is understanding the type of noise. Our dataset contains three different types of noise, and the first stage involves determining the type of noise present in the input image. At this stage, we received a lot of help from ChatGPT.

For this task, we used a three-layer RNN model to extract image features and utilize them for noise classification. Instead of using this deep model, we could have relied on traditional, handcrafted methods, but they were more complex and likely to yield weaker results. Therefore, we leveraged neural network models for feature extraction and noise prediction.

One of the challenges we faced was selecting the number of parameters. We used the image size of (256*192) and chose 15 epochs for training. (All these values were obtained through experimentation) For example, we initially selected a high number of epochs and, based on the loss value at each stage, determined the point of failure (at epoch 15).



```
Epoch 1, Loss: 1.0208, Train Acc: 46.56%
Epoch 2, Loss: 0.5950, Train Acc: 76.82%
Epoch 3, Loss: 0.4101, Train Acc: 82.38%
Epoch 4, Loss: 0.3113, Train Acc: 89.47%
Epoch 5, Loss: 0.2624, Train Acc: 91.66%
Epoch 6, Loss: 0.3436, Train Acc: 87.15%
Epoch 7, Loss: 0.2693, Train Acc: 89.74%
Epoch 8, Loss: 0.2249, Train Acc: 93.25%
Epoch 9, Loss: 0.2077, Train Acc: 93.44%
Epoch 10, Loss: 0.3281, Train Acc: 87.55%
Epoch 11, Loss: 0.1894, Train Acc: 95.10%
Epoch 12, Loss: 0.1711, Train Acc: 95.17%
Epoch 13, Loss: 0.1676, Train Acc: 95.50%
Epoch 14, Loss: 0.1574, Train Acc: 95.83%
Epoch 15, Loss: 0.1569, Train Acc: 95.36%
Epoch 16, Loss: 0.1500, Train Acc: 96.49%
Epoch 17, Loss: 0.1500, Train Acc: 96.23%
Epoch 18, Loss: 0.1425, Train Acc: 95.89%
Epoch 19, Loss: 0.1517, Train Acc: 95.50%
Epoch 20, Loss: 0.1446, Train Acc: 95.83%
Epoch 21, Loss: 0.1366, Train Acc: 95.96%
Epoch 22, Loss: 0.1386, Train Acc: 96.16%
Epoch 23, Loss: 0.1378, Train Acc: 96.03%
Epoch 24, Loss: 0.1317, Train Acc: 96.75%
Epoch 25, Loss: 0.1294, Train Acc: 96.82%
Epoch 26, Loss: 0.1389, Train Acc: 96.62%
Epoch 27, Loss: 0.1412, Train Acc: 96.03%
Epoch 28, Loss: 0.1374, Train Acc: 96.69%
Epoch 29, Loss: 0.1449, Train Acc: 95.36%
Epoch 30, Loss: 0.1354, Train Acc: 96.56%
```

Figure 1: Testing to find the optimal epoch

To calculate and improve the model, we used the CrossEntropyLoss function from the PyTorch library (link).

At the end, we achieved an accuracy of 91.67%, which is a good result for us.

4.2. Noise Reduction

After understanding the type of noise in the image, we need to correct it. This section consists of three functions, each responsible for correcting a specific type of noise. To implement these functions, we used computer vision algorithms, autoencoders, or neural networks.

4.2.1. Gaussian Noise Reduction

Gaussian noise is one of the most common types of noise found in images. This noise spreads throughout the image with a normal distribution and is visible as unusual spots in the image.

To remove Gaussian noise, we used bm3d (make sure to install the library using `pip install bm3d`). This algorithm, which is considered one of the autoencoders, can be controlled with the parameter sigma. In many cases, this parameter is set to 20/255, and this value yields good results on the dataset.

This function applies the sigma value to each of the image channels, normalizes the channels again, and returns the enhanced image. If you want to learn more, please read this Document.

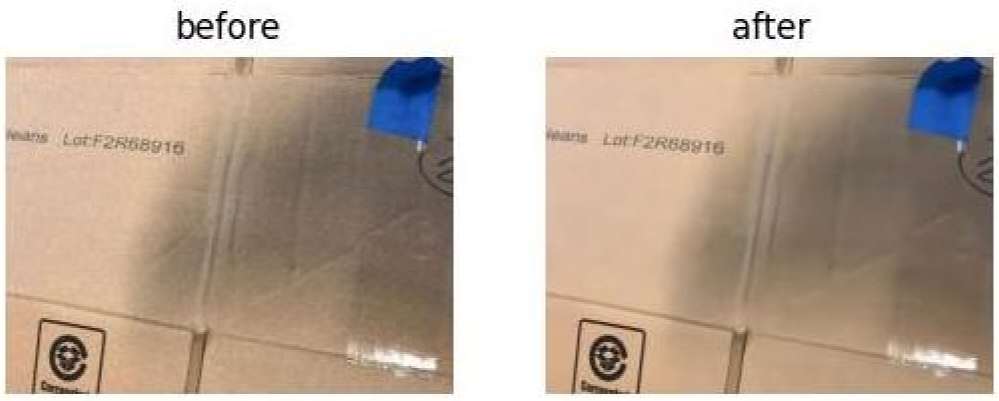


Figure 2: Before and after applying the filter

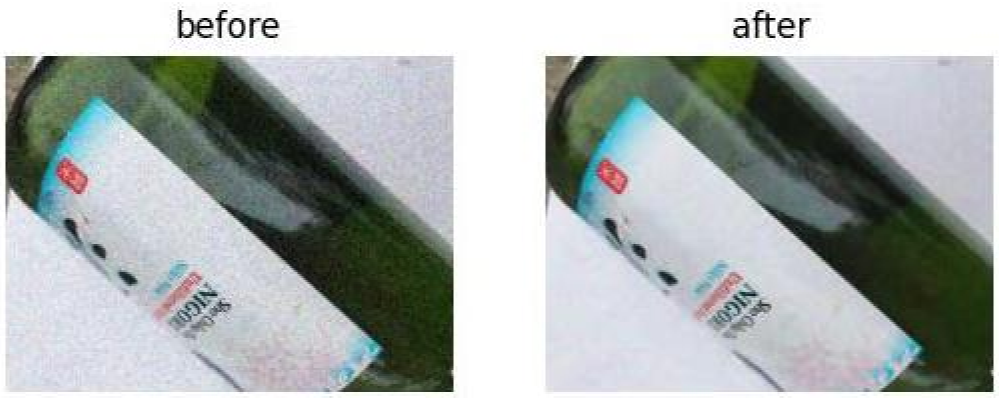


Figure 3: Before and after applying the filter

4.2.2. Salt and Pepper Noise Reduction

Salt-and-pepper noise is another common type of noise in images. This noise alters the pixel values to the minimum or maximum possible values, making it clearly visible.

To remove this noise, we use computer vision techniques. Specifically, we apply a median filter to address salt-and-pepper noise. In this filter, a kernel is used as a parameter to determine the extent of the changes. The larger the kernel size, the more blurred the image becomes, and the amount of noise is reduced. If you want to learn more, please read this Document.

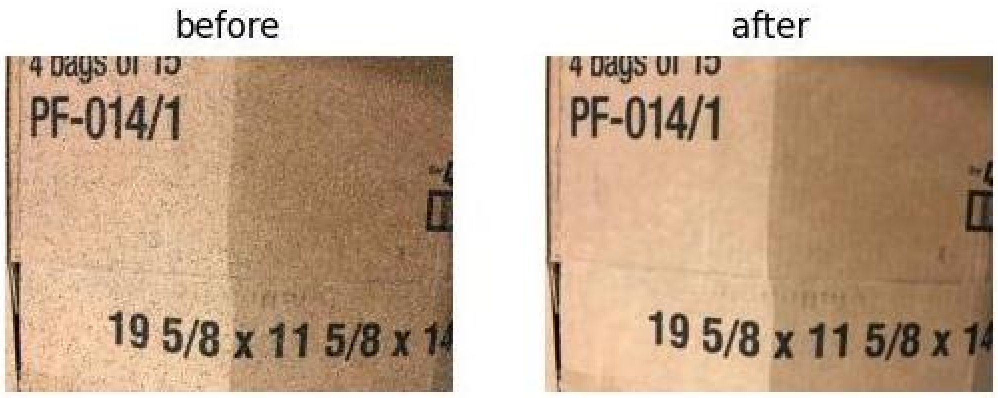


Figure 4: Before and after applying the filter

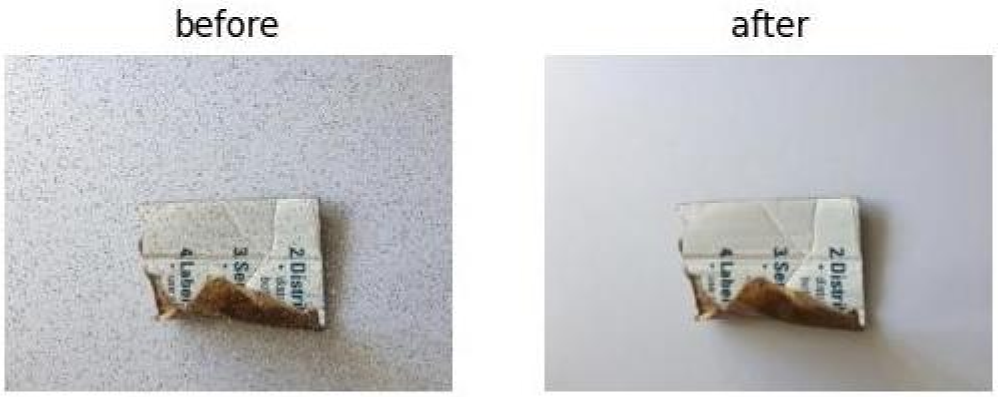


Figure 5: Before and after applying the filter

4.2.3. Periodic Noise Reduction

As the name suggests, periodic noise repeats in the image according to a specific pattern, altering the nature of the image. This type of noise typically has distinct frequencies, which is why its effects can be observed in the Fourier transform. Periodic noise has a directional component, and the images in our dataset exhibit seven different orientations (vertical, horizontal, main diagonal, minor diagonal, etc.). If you want to learn more, please read this Document.

The first approach that comes to mind is to transform the image into the frequency domain and filter out the abnormal points. However, implementing this method on a computer is challenging, and identifying the abnormal points poses a significant challenge. Therefore, we turn to autoencoder methods.

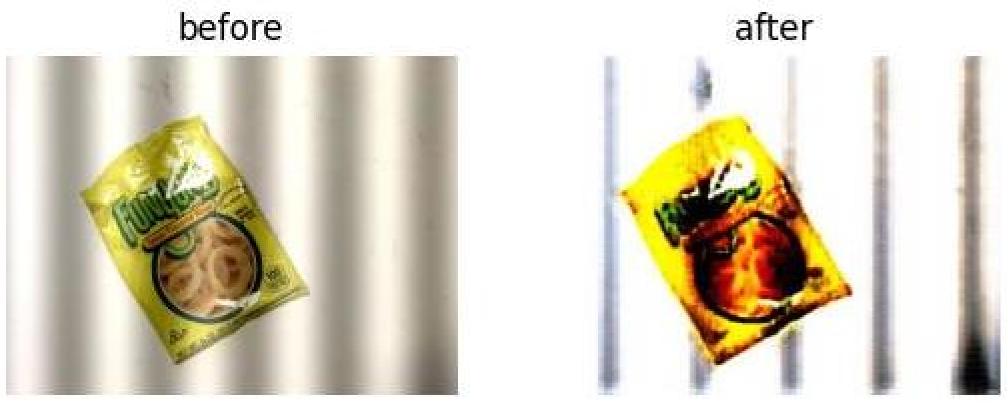


Figure 6: Before and after applying the filter

5. Test Step

We use 20% of the test data we have to evaluate the model, allowing us to analyze and compare both models.

5.1. Label Detection test

5.1.1. Confusion Matrix

In this Figure 6, 0 represents salt-and-pepper noise, 1 represents Gaussian noise, and 2 represents periodic noise.

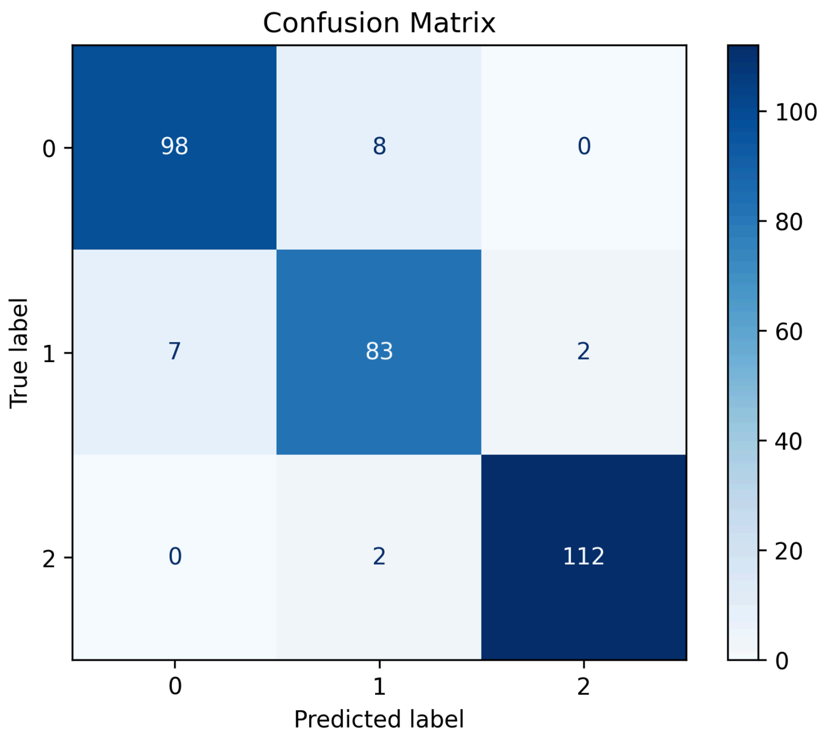


Figure 6: Confusion Matrix of Label Detection Model

Based on the confusion matrix, we can see that the values along the main diagonal are the correctly predicted values, and given the high rate of these values, we can conclude that the model performs well (out of 312 test images, only 19 were misclassified).

Another observation from the confusion matrix is the high prediction rate for periodic noise (out of 114 images, only 2 were misclassified, resulting in an accuracy of 98.2%!). Among the three labels, it can be said that the lowest accuracy is associated with Gaussian noise.

5.1.2. Accuracy

The accuracy value has been determined to be 0.91.

5.1.3. F1 score

The F1 score value has been determined to be 0.91.

5.1.4. Precision

The Precision value has been determined to be 0.91.

5.1.5. Specificity

The Specificity value has been determined to be 0.86.

5.1.6. Recall

The Recall value has been determined to be 0.91.

5.2. Periodic autoencoder test

The test on the trained model at this stage does not yield good results, as the loss value is quite high (around 0.49), and the output images do not meet the desired quality.

6. Conclusion

In this project, we aimed to reduce image noise using machine learning and computer vision models. The project consists of two main components. The first part is noise detection, which identifies the type of noise present in the input image, and the second part is noise reduction, which attempts to correct the noise based on the type identified in the first part.

In the noise reduction section, we have three main components:

- For Gaussian noise, we used the BM3D library and autoencoders.
- For salt-and-pepper noise, we employed computer vision techniques, transforming the image into the frequency domain and applying a median filter.
- For periodic noise, we utilized autoencoders to attempt to reduce the noise, but the results were not satisfactory, as they altered the nature and colors of the image.

The biggest challenge of this project was dealing with periodic noise.

7. References

- https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

- https://matematicas.uam.es/~fernando.chamizo/dark/d_periodic.html

- https://chatgpt.com/

- https://pytorch.org/docs/stable/index.html

- https://en.wikipedia.org/wiki/Image_noise

- https://www.udemy.com/course/pytorch-deep-learning/?srsltid=AfmBOooC1iYoHm0FEkwVAbKgTbyOdx9-GowXbKP2iwpv-FVagdI5jDq6