

**EDUCATION AND SCIENCE MINISTRY OF UKRAINE**

**NATIONAL AVIATION UNIVERSITY**

Faculty of Aeronavigation, Electronics and Telecommunications

Department of computer-integrated complexes

ADMIT TO DEFENSE

Head of the graduate Department

\_\_\_\_\_ V. M. Sineglazov

“ \_\_\_\_\_ ” \_\_\_\_\_ 2023 y.

## **QUALIFICATION WORK**

(EXPLANATORY NOTE)

GRADUATE OF EDUCATION AND QUALIFICATION LEVEL

“BACHELOR”

Specialty 151 "Automation and computer-integrated technologies"

Educational and professional program "Computer-integrated technological processes and production"

**Theme: System of digital noise filtering of hyperspectral images**

Performer: student of group FAET-404 Priymachenko Roman Maximovich

Supervisor: Candidate of Technical Sciences, Oleksandr Hordiienko

Normocontroller:  Filyashkin M. K.

(signature)

**Kyiv 2023**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп'ютерно-інтегрованих комплексів

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

\_\_\_\_\_В.М. Синєглазов

“ \_\_\_\_\_ ” \_\_\_\_\_ 2023 р.

**КВАЛІФІКАЦІЙНА РОБОТА**

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ**

**“БАКАЛАВР”**

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютерно-інтегровані технологічні процеси і виробництва»

**Тема: Система цифрової фільтрації шумів гіперспектральних зображень**

Виконавець: студент групи ФАЕТ-404 Приймаченко Роман Максимович

Керівник: кандидат технічних наук, Гордієнко Олександр

Нормоконтролер:  Філяшкін М. К.

(підпис)

**Київ 2023**

NATIONAL AVIATION UNIVERSITY

Faculty of aeronavigation, electronics and telecommunications

Department of Aviation Computer Integrated Complexes

**Educational level:** bachelor

**Specialty:** 151 "Automation and computer-integrated technologies"

**APPROVED**

Head of Department

V.M. Sineglazov

" \_\_\_\_ " \_\_\_\_\_ 2023

**TASK**

**For the student's thesis**

Priymachenko Roman Maximovich

- 1. Theme of the project:** « System of digital noise filtering of hyperspectral images»
- 2. The term of the work (project):** 10.03.2023 – 10.06.2023
- 3. Output data to the work (project):** Datasets of satellite images with detectable noise, mathematical description of image filtering techniques, architecture of neural networks for digital filtering.
- 4. Contents of the explanatory note (list of questions to be developed):**  
Section 1. Introduction to hyperspectral image noise filtering; Section 2: Wavelets transforms in denoising; Section 3: Neural networks in digital filtering; Section 4: Digital noise filtering realization on hyperspectral images
- 5. List of mandatory graphic material:**  
Schematic representation of filtering methods and neural network architecture modeling.

## 6. Planned schedule:

№	Task	Execution term	Execution mark
1.	Getting the task	01.04.2023 – 02.04.2023	Done
2.	Formation of the purpose and main objectives of the study	02.04.2023 – 14.04.2023	Done
3.	Analysis of existing methods	15.04.2023 – 30.04.2023	Done
4.	Theoretical consideration of problem solving	01.05.2023 – 05.05.2023	Done
5.	Software implementation of the hyperspectral image classification program	06.05.2023 – 25.05.2023	Done
6.	Preparation of an explanatory note	26.05.2023 – 03.06.2023	Done
7.	Preparation of presentation and handouts	04.06.2023 – 06.06.2023	Done

## 7. Date of task receiving: “\_\_\_” \_\_\_\_\_ 2023 y.

Diploma thesis supervisor

\_\_\_\_\_

(signature)

Hordiienko O.M.

Issued task accepted

\_\_\_\_\_

(signature)

Priymachenko R.M

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп'ютерно-інтегрованих комплексів

**Освітній ступінь:** бакалавр

**Спеціальність:** 151 "Автоматизація та комп'ютерно-інтегровані технології"

**ЗАТВЕРДЖУЮ**

Завідувач кафедри

В.М. Синєглазов

“ \_\_\_\_\_ ” \_\_\_\_\_ 2023 р.

**ЗАВДАННЯ**

**на виконання дипломної роботи студента**

Приймаченка Романа Максимовича

- 1. Тема роботи (проекту):** «Система цифрової фільтрації шумів гіперспектральних зображень».
- 2. Термін виконання проекту (роботи):** 10.03.23 – 10.06.23
- 3. Вихідні дані до роботи (проекту):** Датасети супутникових знімків з найявним шумом, математичний опис технік фільтрації зображень, архітектура нейронних мереж для цифрової фільтрації.
- 4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):** Розділ 1. Введення в методи фільтрації гіперспектральних зображень; Розділ 2: Вейвлет перетворення в усуненні шумів; Розділ 3: Нейронні мережі в цифровій фільтрації; Розділ 4: Реалізація цифрової фільтрації шуму на гіперспектральних зображеннях

**5. Перелік обов'язкового графічного матеріалу:** Схематичне зображення методів фільтрації та моделювання архітектури нейронних мереж.

**6. Календарний план-графік**

№	Завдання	Термін виконання	Відмітка про виконання
1.	Отримання завдання	01.04.2023 – 02.04.2023	Виконано
2.	Формування мети та основних завдань дослідження	02.04.2023 – 14.04.2023	Виконано
3.	Аналіз існуючих методів	15.04.2023 – 30.04.2023	Виконано
4.	Теоретичний розгляд рішення задач	01.05.2023 – 05.05.2023	Виконано
5.	Програмна реалізація програми класифікації гіперспектральних зображень	06.05.2023 – 25.05.2023	Виконано
6.	Оформлення пояснювальної записки	26.05.2023 – 03.06.2023	Виконано
7.	Підготовка презентації та роздаткового матеріалу	04.06.2023 – 06.06.2023	Виконано

**7. Дата видачі завдання:** “ \_\_\_ ” \_\_\_\_\_ 2023 р.

Керівник дипломної роботи \_\_\_\_\_  
(підпис)

Гордієнко О.М.

Завдання прийняв до виконання \_\_\_\_\_  
(підпис)

Приймаченко Р.М.



## ABSTRACT

Text part of the work: \_\_\_\_, \_\_\_\_ fig., \_\_\_\_ table, \_\_\_\_ references.

**Object of research** – Hyperspectral image noises

**Subject of the research** - Noise filtering methods in neural networks

**Purpose of the work** – Define mathematical model with used of neural networks and create a software implementation with the help of Python

**Methods of research** - Theoretical analysis and experimental studies using a combinatorial neural network model.

The paper describes the basic principles of filtering approaches and their use. The effectiveness of the entire process of removing noise on satellite images depends on recognizing the type of noise and using the right type of filtering, or their combination/modifications. Experimental studies of the combination of neural network and traditional filtering methods were carried out, and a model based on inverse Wavelet transformations and a convolutional neural network was developed.



# CONTENT

INTRODUCTION.....	11
GLOSSARY .....	12
PROBLEM STATEMENT .....	13
1 INTRODUCTION TO HYPERSPECTRAL IMAGE NOISE FILTERING .....	14
1.1 Images in remote sensing.....	14
1.2 Characteristics of the remote sensing images .....	17
1.3 Satellites with hyperspectral sensors.....	18
1.4 Noise in remote sensing.....	19
1.4.1 Amplifier Noise .....	20
1.4.2 Salt-and-Pepper Noise .....	21
1.4.3 Speckle Noise .....	22
1.5 Consideration of modern systems and methods of noise filtering .....	24
1.5.1 Mean Filter .....	25
1.5.2 Standard Median Filter .....	25
1.5.3 Adaptive Wiener Filter .....	26
1.5.4 Gaussian Filter.....	27
1.6 Conclusion.....	29
2 WAVELETS TRANSFORMS IN DENOISING .....	30
2.1 Wavelets Transforms .....	30
2.2 Discrete Wavelet Transform .....	32
2.3 Single Level Decomposition.....	33
2.4 Multi-level Decomposition .....	34
2.5 Thresholding in Wavelets .....	34
2.6 Inverse Wavelet Transform .....	36

3 NEURAL NETWORKS IN DIGITAL FILTERING .....	39
3.1 Using Deep Learning neural networks .....	39
3.2 Convolutional Neural Networks .....	41
3.2.1 Convolution Layers .....	43
3.2.2 Pooling Layer .....	44
3.2.3 Fully Connected Layer.....	45
3.2.4 Activation Functions.....	46
3.3 U-Net architecture for CNN.....	47
3.3.1 Encoder Path.....	47
3.3.2 Decoder Path .....	49
3.3.3 Skip connections.....	50
3.4 Wavelets basis Neural network.....	51
3.5 Conclusion.....	52
4 DIGITAL NOISE FILTERING REALISATION ON HYPERSPECTRAL IMAGES .....	53
4.1 Cascade Wavelets CNN model introduction .....	53
4.2 Data preparation and model training details.....	59
4.3 Filtration results.....	64
4.4 Performance evaluation .....	67
4.5 Conclusion.....	68
CONCLUSION .....	69
REFERENCE.....	70
APPENDIX A .....	73
APPENDIX B .....	79
APPENDIX C .....	85

## INTRODUCTION

In today's world, remote sensing has become an incredibly powerful and vital technology, changing the way we acquire information about our globe and beyond. It entails gathering data from afar through satellites, airplanes, drones, or ground-based sensors. The capacity of technology to collect data across wide regions for long time periods, allowing for thorough monitoring and analysis, is an important feature.

It is used in a wide variety of disciplines in thousands of different use cases, including most earth sciences, such as meteorology, geology, hydrology, ecology, oceanography, glaciology, geography, and in land surveying, as well as applications in military, intelligence, commercial, economic, planning, and humanitarian fields. Its importance cannot be emphasized enough, as it provides vital insights and data for informed decision-making and long-term growth.

Remote sensing gathers samples of emitted and reflected electromagnetic (EM) radiation from terrestrial, atmospheric, and aquatic ecosystems to detect and monitor the physical characteristics of the area without physical contact. However, one significant challenge that remote sensing encounters is the presence of image noise.

Noise refers to any unwanted information that contaminates an image, causing distortions. It arises from various sources when images are captured. The primary cause of noise in digital images is the process of converting an optical image into a continuous electrical signal during digital image acquisition. The introduction of noise into an image can occur in different ways, depending on the image creation method. For instance, satellite images often contain noise signals that result in distorted images, making it difficult to comprehend and study them effectively. To address this issue, appropriate filters are employed to limit or reduce the amount of noise present.

## **GLOSSARY**

CNN – Convolutional Neural Network

CWCNN – Cascade Wavelets Convolutional Neural Network

DWT – Discrete Wavelet Transform

IWT – Inverse Wavelet Transform

MF – Mean Filter

SMF – Standard Median Filter

AWF – Adaptive Wiener Filter

GF – Gaussian Filter

FCL – Fully Connected Layer

FCN – Fully Convolutional Network

GPU - Graphics Processing Units

## **PROBLEM STATEMENT**

The rapid development of remote sensing technology has made it possible to increase the number of obtaining satellite and aerial photographs for a wide range of tasks. However, these images often suffer from various types of noise, including Gaussian noise, impulse noise, and speckle noise, which can degrade the quality and accuracy of the extracted information. Traditional image filtering techniques such as mean filter, median filter, adaptive Wiener filter, and Gaussian filter are widely used to mitigate noise in remote sensing images. But, conventional methods have their limitations and may not provide satisfactory results in terms of noise suppression and preservation of image details.

To address the limitations of traditional image filtering techniques for remote sensing images, we propose the development of a digital noise filtering system that combines these traditional methods with neural networks to achieve better noise reduction while preserving important image details. By leveraging the capabilities of neural networks, which can learn complex mappings between noisy and clean images.

# **1 INTRODUCTION TO HYPERSPECTRAL IMAGE NOISE FILTERING**

Remote sensing is a technique used to acquire information about the Earth's surface without direct physical contact. This method involves utilizing specialized sensors on satellites or aircraft to record various types of data about the underlying surface. Remote sensing data holds immense value in offering insights into Earth's ecosystems, weather patterns, land use, and much more. There are two primary categories of remote sensing: passive and active.

Passive remote sensing entails capturing naturally available radiation, with reflected sunlight being the most commonly detected source. Passive sensors are limited to recording data when the observed material or object is naturally illuminated, either during daylight or through thermal emission.

In contrast, active remote sensing involves emitting a signal and then measuring the signal's return to the sensor. Examples of active remote sensing include radar and LiDAR systems. These systems are advantageous as they allow data collection irrespective of the time of day or prevailing weather conditions.

The importance of remote sensing has witnessed exponential growth in recent years, driven by our increasing reliance on technology and new data.

## **1.1 Images in remote sensing**

Remote sensing [1, 3] is the process of acquiring information about an object or a phenomenon without making physical contact with the object, typically through the use of satellite or aircraft-based sensor technologies. This field has a significant impact on areas like environmental monitoring, meteorology, geology, oceanography, agriculture, cartography, military intelligence, and more.

There are several types of images in remote sensing, which are primarily categorized based on the spectral coverage of the sensors. The main types include:

- **Panchromatic Images:** These images are captured in one broad spectral band, typically covering the visible and near-infrared parts of the spectrum. The term 'panchromatic' comes from 'pan' (all) and 'chroma' (colors), which can be misleading because panchromatic images are grayscale, but they cover a broad range of wavelengths.

- **Multispectral Images:** Multispectral imaging involves capturing data at specific frequencies across the electromagnetic spectrum, including visible light and infrared. The most familiar multispectral images are those produced in the three bands of the visible spectrum (red, green, blue - RGB).

- **Hyperspectral Images:** Hyperspectral imaging, like multispectral imaging, involves collecting and processing information from across the electromagnetic spectrum. While multispectral imaging divides the spectrum into a handful of bands, hyperspectral imaging divides it into hundreds or even thousands of narrow, adjacent spectral bands. This results in a continuous spectrum for each pixel in the image, providing a more detailed image of the object or scene.

- **Thermal Images:** Thermal images detect the heat emitted by objects, providing information about their temperature. They are useful in environmental studies, detecting heat loss in buildings, identifying people or animals in the dark, etc.

- **Radar and LiDAR Images:** These images are created by sending a signal (radio waves for Radar, light for LiDAR) and measuring the time it takes for the signal to bounce back to the sensor. This can provide detailed information about distances, shapes, and even the material properties of the objects.

As for the difference between multispectral (RGB) and hyperspectral images, it comes down to the number and width of spectral bands.

In multispectral imagery [2], the electromagnetic spectrum is divided into three broad bands that correspond to red, green, and blue light. This is much like how human eyes see color, and so these images can be readily interpreted.

Hyperspectral images, on the other hand, divide the spectrum into many narrow bands. Because different materials reflect light differently at different wavelengths, hyperspectral imagery can identify the materials that make up a scanned object or scene with high precision. For instance, hyperspectral imaging can differentiate between types of vegetation or detect pollutants in water or air. The disadvantage of hyperspectral imagery is that it generates a lot of data, which can be challenging to store, process, and interpret.

Therefore, while RGB multispectral images [5] provide a broad overview of a scene in a way that is easy for humans to interpret, hyperspectral images provide a much more detailed view that can reveal information not visible to the human eye, at the cost of increased complexity in data processing and interpretation.

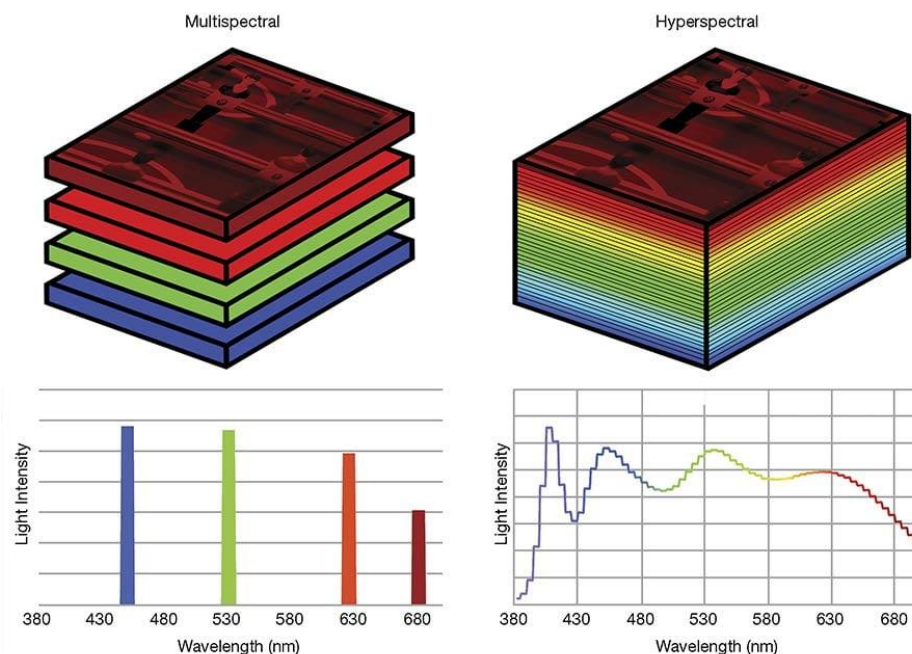


Figure 1.1 Visualization of multispectral and hyperspectral cubes which contain multiple images in different spectra



## 1.2 Characteristics of the remote sensing images

The effectiveness of analyzing and interpreting space images depends on the amount and nature of information available about remote sensing objects. These objects are determined based on the specific thematic task at hand. Space images are generated by capturing electromagnetic radiation emitted or reflected by natural formations and human-made objects on Earth.

Each remote sensing object possesses distinct spectral and energy characteristics, as well as varying geometric sizes, shapes, and temporal and spatial behaviors. When selecting a space system for image generation, all these features of remote sensing objects need to be taken into consideration.

Initially, the following factors are considered:

- The spectral range relevant to the objects and processes being observed and studied.
- The level of detail required to accurately capture the geometric shape of objects and their spatial relationships.
- Radiometric resolution, which refers to the maximum number of bits used to represent the range of pixel brightness values in images of Earth's surface objects.
- The area, or the geometric dimensions of the survey frame, representing a specific region on Earth's surface to be observed.
- Ensuring the capability for either one-time monitoring or periodic observation with a specific time interval for a given geographical area.

### 1.3 Satellites with hyperspectral sensors

Satellites with hyperspectral sensors play a crucial role in gathering detailed, comprehensive, and multi-dimensional data about the Earth's surface, enabling better understanding, monitoring, and management of our environment and resources. Here some of them:

- MODIS (Moderate Resolution Imaging Spectro radiometer), which is equipped on both the Terra and Aqua satellites. NASA (USA) launched Terra in 1999, followed by Aqua in 2002. MODIS is capable of capturing high-frequency imagery across a wide spectrum with 36 spectral bands ranging from 0.4 to 14.4  $\mu\text{m}$ . It is primarily utilized for monitoring global processes such as vegetation dynamics, carbon cycling, and water cycles.

- Landsat series, which consists of American Earth-observing satellites. The most recent addition to this series is Landsat 9, which was launched in 2021. These satellites employ various instruments including the Multispectral Scanner (MSS), Thematic Mapper (TM), Enhanced Thematic Mapper (ETM), and Operational Land Imager (OLI). They provide imagery of the Earth's surface across visible, near-infrared, and thermal infrared spectrums.

- The ASTER (Advanced Spaceborne Thermal Emission and Reflection Radiometer) instrument is installed on the Terra satellite. NASA launched this satellite in 1999. ASTER captures imagery in 14 channels spanning the visible, near-infrared, and thermal infrared spectrums.

- The Sentinel series comprises satellites launched by the European Space Agency as part of the Copernicus Earth observation program. For instance, Sentinel-2 is equipped with a Multispectral Instrument (MSI) enabling observations in the visible, near-infrared, and shortwave infrared spectrums.

- WorldView-3, launched by DigitalGlobe (now Maxar Technologies). WorldView-3 is capable of providing high-resolution imagery as well as multispectral images.

## 1.4 Noise in remote sensing

Noise in remote sensing refers to unwanted or random variations that are introduced during the acquisition, transmission, or processing of remote sensing data. It can have a detrimental effect on the quality, accuracy, and interpretability of the acquired imagery. Understanding and mitigating noise is crucial for obtaining reliable information and making informed decisions based on remote sensing data.

Remote sensing data can be affected by various sources of noise [3, 4], which can arise at different stages of the data acquisition and processing pipeline, such as:

- **Sensor Noise:** Sensor noise originates from inherent imperfections in the remote sensing hardware, including electronic components, detectors, and optics. It can manifest as random variations in signal intensity, non-uniformity, or systematic biases across the image. Sensor noise can degrade the image quality and introduce uncertainties in the derived information.
- **Transmission Noise:** Noise can be introduced during the transmission of remote sensing data from the sensor to the ground station or satellite. Interference, signal loss, or distortion in the transmission path can corrupt the data, leading to errors or noise-induced artifacts in the acquired imagery.
- **Environmental Noise:** Environmental factors such as weather conditions, surface roughness, or topographic variations can contribute to noise in remote sensing data. For example, variations in illumination due to cloud cover, shadows, or sun angle changes can introduce noise or inconsistencies in image datasets.
- **Processing Noise:** Noise can also be introduced during the processing and manipulation of remote sensing data. Errors in resampling, image registration, geometric correction, or radiometric calibration procedures can introduce noise or artifacts that affect the final image quality.

Noise in remote sensing data poses several challenges and impacts the interpretation and analysis of the acquired imagery. It can obscure or distort

important features, reduce the accuracy of derived measurements, and affect subsequent data processing tasks such as classification or change detection.

Effective noise mitigation techniques, such as filtering, de-noising algorithms, or sensor calibration, are crucial for improving the quality and reliability of remote sensing data products.

Addressing noise in remote sensing involves a combination of hardware improvements, calibration procedures, and advanced image processing techniques. Developing robust noise modeling and estimation methods, as well as employing sophisticated de-noising algorithms, can significantly enhance the quality and utility of remote sensing imagery, enabling accurate and reliable analysis for various applications in environmental monitoring, land cover mapping, disaster assessment, and resource management [4].

There are three common types of image noise:

#### **1.4.1 Amplifier Noise**

The conventional model for amplifier noise assumes that it is additive, follows a Gaussian distribution, and is dependent on each pixel as well as the signal intensity. This type of noise is primarily caused by thermal effects, including the reset noise of capacitors. Essentially, it is an idealized version of white noise, which arises due to random fluctuations in the signal. In Gaussian noise, each pixel's value in the image will be slightly altered from its original value.

Gaussian noise [6] can be described as a form of statistical noise that exhibits a probability density function (PDF) identical to that of the normal distribution, also known as the Gaussian distribution.

The probability density function  $\mathcal{P}$  of a Gaussian random variable  $Z$  is given by:

$$\mathcal{P}G(Z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(Z-\mu)^2}{2\sigma^2}} \quad (1.1)$$

$Z$  denotes the grey level,  $\mu$  represents the mean grey value, and  $\sigma$  indicates the standard deviation associated with the grey level.

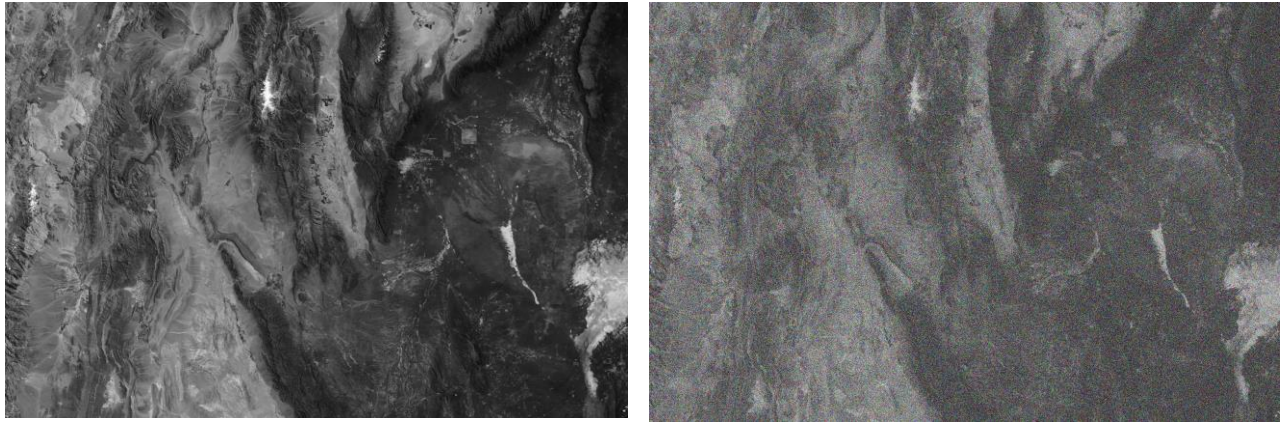


Figure 1.2. Before and after Gaussian noise

### 1.4.2 Salt-and-Pepper Noise

Salt and pepper noise [7], also known as impulse noise, spike noise, random noise, or independent noise, refers to a type of image distortion characterized by sparse occurrences of light and dark disturbances. In this form of noise, individual pixels within the image exhibit significantly different colors or intensities compared to their surrounding pixels. It primarily affects a small number of pixels within the image. Common sources of this type of noise include dust particles present inside the camera and overheated or faulty elements of the Charge-coupled device.

Salt and pepper distribution noise can be expressed by:

$$p(x) = \begin{cases} p_1, & x = A \\ p_2, & x = B \\ 0, & \textit{otherwise} \end{cases} \quad (1.2)$$

$p_1$  and  $p_2$  represent the Probability Density Functions (PDFs), while  $p(x)$  represents the distribution of salt and pepper noise within an image.  $A$  and  $B$  correspond to the size of arrays representing the image.

An image affected by salt and pepper noise will exhibit dark pixels within bright regions and vice versa. It can occur due to memory cell failures, synchronization errors during image digitization or transmission, analog-to-digital converter errors, or bit errors during transmission.

An example of salt and pepper noise is as follows:



Figure 1.3. Salt-and-Pepper Noise affect

### 1.4.3 Speckle Noise

Speckle noise [8] refers to a specific type of noise characterized by a granular pattern commonly observed in satellite images. Its removal poses a significant challenge and remains a relatively unexplored issue. This type of noise is predominantly encountered in satellite images obtained through Synthetic Aperture Radar (SAR), a specialized radar system. Speckle noise is considered an undesirable artifact resulting from the random interference between coherent signals reflected by numerous scatterers present on the Earth's surface, occurring on a scale comparable to the wavelength of the incident radar wave.

Speckle noise is represented as multiplicative noise, meaning that the resulting signal is obtained by multiplying the speckle signal with the original noise.

Let's consider  $I(i, j)$  as the distorted pixel of an observed image, and  $S(i, j)$  as the noise-free image pixel that we aim to restore. According to the multiplicative noise model,

$$I(i, j) = S(i, j) * N(i, j) \quad (1.3)$$

In which  $N(i, j)$  depicts the multiplicative noise with unit mean and standard deviation. Example of Speckle Noise:

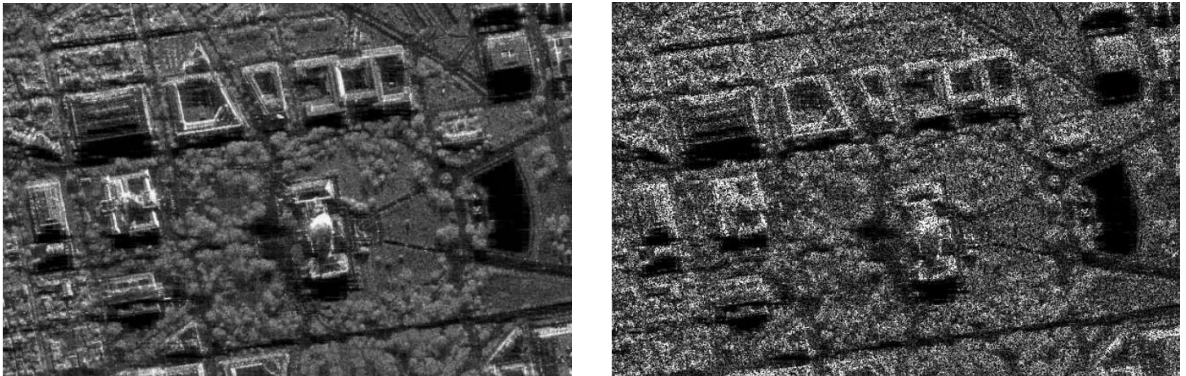


Figure 1.4. Original Image and Speckled Image compare

To mitigate image noise, several techniques are employed in remote sensing. One commonly used method is image filtering [3], which aims to remove or reduce the noise while preserving the essential features of the image. Filtering techniques such as mean filtering, median filtering, Gaussian filtering, and wavelet filtering are applied to remove different types of noise, depending on their characteristics [8].

Additionally, advanced algorithms and image processing techniques are developed to enhance the quality of remote sensing data. These algorithms involve sophisticated statistical methods, machine learning approaches, and spectral analysis techniques to distinguish between noise and actual signal, thereby improving the accuracy and reliability of the data.

The importance of addressing image noise in remote sensing cannot be overstated. By reducing noise and improving the quality of remote sensing data, researchers, scientists, and decision-makers can make more accurate and informed interpretations, leading to better insights, reliable assessments, and effective decision-making processes.

## 1.5 Consideration of modern systems and methods of noise filtering

In a digital camera, if the light, which enters the lens, misaligns with the sensors, it will create image noise. Every type of electronic device receives and transmits some noise and sends it on to what it is creating. When the images are transmitted over channels, they are corrupted with impulse noise due to noisy channels. This impulse noise consists of large positive and negative spikes.

The positive spikes have values much larger than the background and thus they appear as bright spots, while the negative spikes have values smaller than the background and they appear as darker spots. Both the spots [5] for the positive and negative spikes are visible to the human eye. Also, Gaussian type of noise affects the image. Thus, filters are required for removing noises before processing. There are many kinds of filters as linear mean filter, median filter, wiener filter and Gaussian Filter. Due filtering process, the three primaries (R, G and B) are done separately. It is followed by some gain to compensate for attenuation resulting from the filter.

The filtered primaries are then combined to form the colored image.

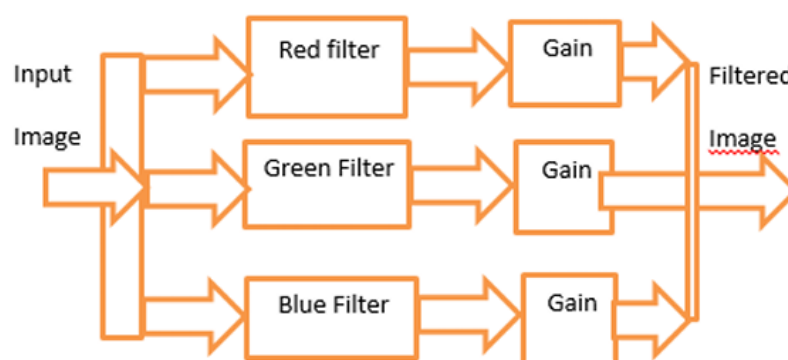


Figure 1.5. Filtering process



### 1.5.1 Mean Filter

The Mean Filter (MF) [9] is a straightforward and easily implemented linear filter used for the purpose of smoothing images, thereby reducing the level of intensity variation between adjacent pixels. Its primary application is noise reduction in images. The fundamental concept behind mean filtering involves replacing each pixel value in an image with the average (mean) value of its neighboring pixels, including the pixel itself. This process effectively eliminates pixel values that deviate significantly from their surrounding values. Mean filtering is commonly referred to as a convolution filter, as it utilizes a kernel that determines the shape and size of the neighborhood considered for calculating the mean value.

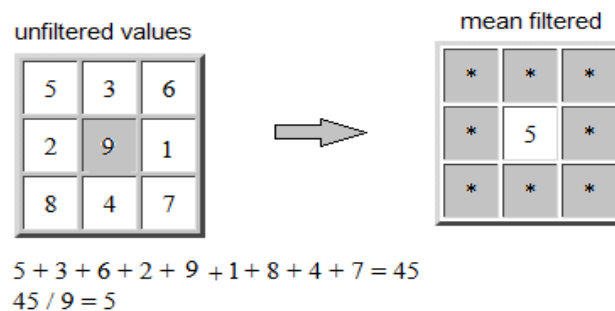


Figure 1.6. Schematic representation of mean filtering in a 3X3 kernel

### 1.5.2 Standard Median Filter

The median filter (SMF) is a type of non-linear filter [10] used to adjust the average intensity of an image when the spatial distribution of noise within the image is not symmetrical across the window. Its primary function is to decrease the variability of intensities present in the image. Unlike other filters, the median filter operates on the spatial characteristics of the image by employing a 2-D mask that is applied to every pixel in the input image. Applying the mask involves centering it on a pixel, assessing the brightness values of the pixels it covers, and determining the median brightness value.

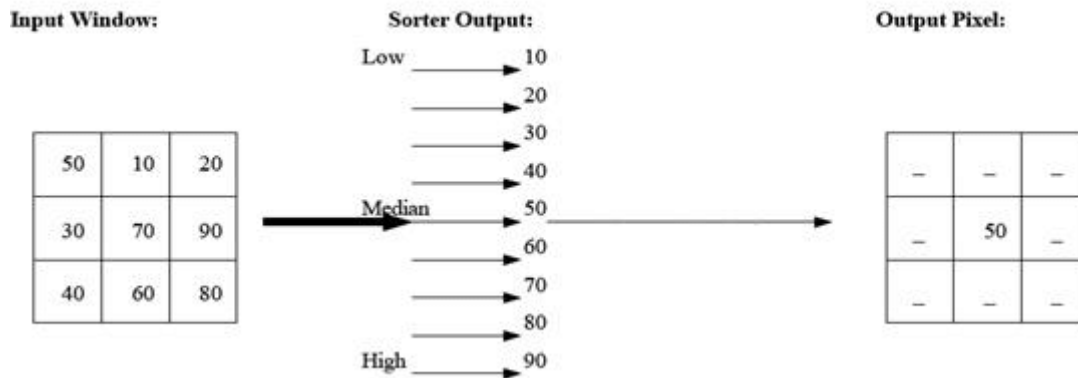


Figure 1.7. A graphical depiction of the standard median filter operation

### 1.5.3 Adaptive Wiener Filter

Adaptive Wiener Filter (AWF) [11] is a type of linear filter, to an image. It adjusts itself based on the local variance of the image, resulting in less smoothing for images with large variance and more smoothing for images with small variance. This approach often yields superior outcomes compared to linear filtering.

The adaptive nature of the filter makes it more discerning than a standard linear filter, preserving important features like edges and high-frequency components in the image. Furthermore, the wiener function takes care of all the necessary computations and implements the filter for an input image, eliminating the need for additional design tasks. However, wiener function requires more computational time compared to linear filtering.

The effectiveness of the Wiener filter is particularly prominent when dealing with constant-power ("white") additive noise, such as Gaussian noise. An alternative method for noise reduction involves evolving the image through an anisotropic diffusion process, which can be seen as a smoothing partial differential equation akin to the heat equation. This technique, known as anisotropic diffusion, provides another means of removing noise from an image.

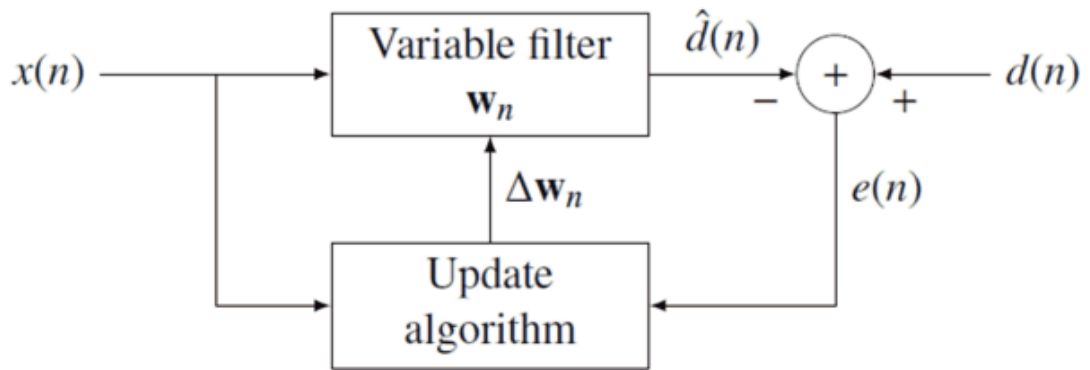


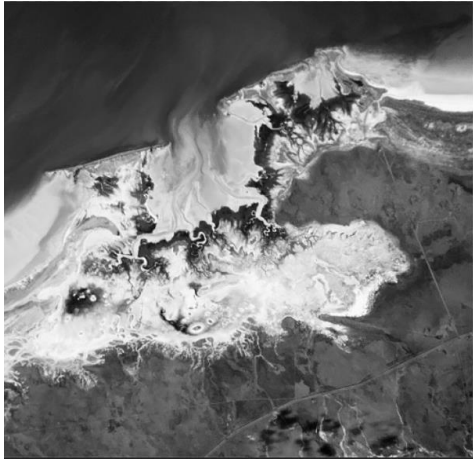
Figure 1.8. Block diagram of Adaptive Wiener Filter

"k" - refers to the sample number; "x" - represents the reference input; "x" - denotes the recent values of; "x," "d" - signifies the desired input, "W" - indicates the filter coefficients; "ε" - represents the error output; "f" - filter impulse response, "\*" - denotes convolution;  $\Sigma$  - signifies summation, upper box -linear filter, and lower box - adaption algorithm.

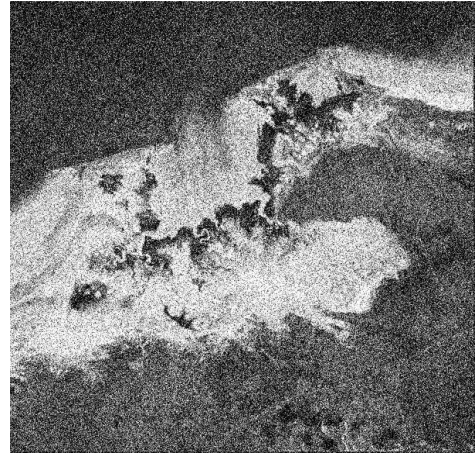
#### 1.5.4 Gaussian Filter

The Gaussian low pass filter (GF) is widely used in image processing and is known for its ability to respond to sudden changes in the input signal. Its design focuses on minimizing overshoot and optimizing the transition time when applied to a step function input. By utilizing the principles of Gaussian smoothing, this filter effectively eliminates noise and blurring from images.

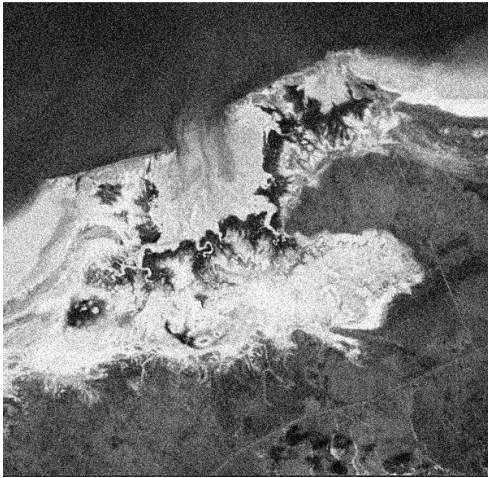
Gaussian filters [12] are commonly function by convolving an image with a Gaussian kernel, which is a weighted averaging function. The Gaussian kernel assigns higher weights to the central pixels and gradually decreases the weights for pixels further away. This property ensures that the filter successfully reduces high-frequency noise while preserving the overall structure and fine details of the image.



(a) Original



(b) 60% Noisily



(c) AWF



(d) GF



(e) MF



(f) SMF

Figure 1.9. Representation of all filters work

## **1.6 Conclusion**

Hyperspectral imaging has brought about a great revolution in the field of remote sensing, ushering in an era of unparalleled detail and accuracy in data collection. This cutting-edge technology has the remarkable ability to detect radiation across an extensive range of wavelengths, thereby enabling the construction of intricate hyperspectral data cubes and the identification of unique spectral signatures. Applications of hyperspectral imaging span a wide array of domains, encompassing agriculture, military operations, and medical advancements.

Within the realm of ongoing research, one particularly fascinating and promising avenue of exploration lies in the domain of filtering hyperspectral images. Researchers continuously dedicate their efforts to studying and developing a myriad of primary filtration methods, all in the pursuit of achieving optimal results with the least possible amount of effort. Remarkably, the scientific community is increasingly turning to the utilization of neural networks, which seamlessly integrate with existing approaches, thereby showcasing their exceptional efficiency and potential in this field of study.

## **2 WAVELETS TRANSFORMS IN DENOISING**

Wavelets are mathematical functions that divide a given function or set of data into different frequency components, and then study each component with a resolution matched to its scale. They have advantages over traditional methods in analyzing physical situations where the signal contains discontinuities and sharp spikes.

The appeal of wavelet-based image filtering [13] stems from the multiresolution nature of wavelet transforms. This property enables wavelets to analyze different frequency components with different resolutions, capturing both global (low frequency) and detailed (high frequency) information about an image.

Wavelets can represent an image as a sum of "wavelet coefficients," [14] each corresponding to a certain frequency range at a certain location in the image. By manipulating these coefficients, we can selectively enhance or suppress features in different frequency ranges and at different scales, providing a high degree of control over the filtering process.

### **2.1 Wavelets Transforms**

Wavelets proven to be valuable in the field of image noise filtering. Unlike traditional methods that focus solely on frequency analysis, wavelets offer a more comprehensive approach by considering both frequency and spatial information.

A wavelet is a type of oscillation that resembles a wave but decays rapidly and has a mean value of zero. In contrast to sinusoids, which extend infinitely, wavelets have a finite duration.

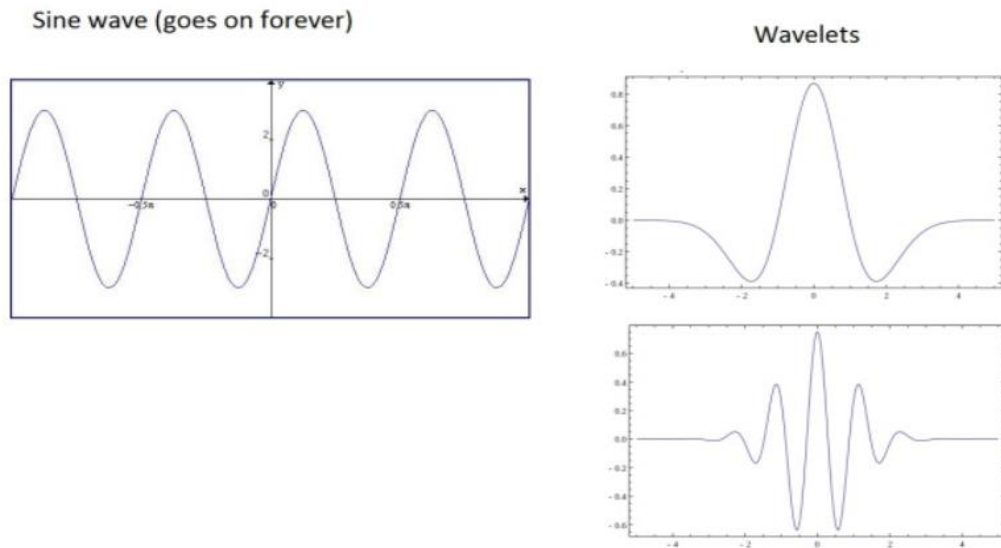


Figure 2.1. Wave and Wavelet compare

When analyzing a signal or function  $f(t)$ , it is often beneficial to express it as an expansion, which can provide a more effective analysis.

$$\Psi_{j,k}(t) = 1/j\psi\left(t - \frac{k}{j}\right), j, k \in \mathbb{R} \quad (2.1)$$

The equation depicts the wavelet expansion functions, where the function represents the specific wavelet function used, and  $j$  and  $k$  denote the scaling and shifting parameters, respectively.

In the context of image noise filtering, wavelet analysis [13] allows for the decomposition and analysis of signals at different scales. This decomposition helps in effectively separating noise from the actual image content. By examining the image at multiple scales, wavelet analysis can capture fine details as well as overall features.

Coefficients that represent the signal at different scales and positions provide insights into the presence of noise in specific frequency bands. The sparsity of noise in the wavelet domain is exploited to selectively remove unwanted noise while retaining the structure and details of the image.

The effectiveness of wavelet-based image noise filtering [19] depends on the choice of wavelet function and thresholding strategy. Different wavelet families offer different levels of smoothness and frequency localization, allowing for flexibility in adapting to specific image characteristics. Additionally, the selection of an appropriate thresholding method, such as hard or soft thresholding, determines the extent of noise suppression and preservation of image details

## 2.2 Discrete Wavelet Transform

The wavelet [19] is derived from a scaling function that characterizes its scaling properties. The requirement for the scaling functions to be orthogonal to their discrete translations imposes certain mathematical constraints, which are widely discussed, such as in the context of the dilation equation. These conditions are commonly mentioned in various sources to ensure the appropriate properties of the wavelet.

$$\phi(x) = \sum_{k=-\infty}^{\infty} a_k \phi(s_x - k) \quad (2.2)$$

In the given equation, S represents a scaling factor typically chosen as two. Additionally, it is important to ensure that the area between the function is normalized, and the scaling function is orthogonal to its integer translates.

$$\int_{-\infty}^{\infty} \phi(x)\phi(x + \iota) dx = \delta_{0,\iota} \quad (2.3)$$

After incorporating additional conditions to address the non-uniqueness of the solution, it is possible to derive the outcomes for all the given equations. These outcomes consist of a finite set of coefficients denoted as  $a_k$ , which not only determine



the scaling function but also yield the wavelet. The wavelet can be derived from the scaling function through the following relationship:

$$\psi(x) = \sum_{k=-\infty}^{\infty} (-1)^k a_{N-1-k} \psi(2x - k) \quad (2.4)$$

When considering an even integer  $N$ , the collection of wavelets creates an orthonormal basis that is utilized for signal decomposition.

### 2.3 Single Level Decomposition

Based on Figure 2.2, the signal  $s(t)$  data [16] is represented by  $cA$ . This data undergoes processing through two distinct filters: a high-pass filter and a low-pass filter. It is important to note that these filters deviate from the conventional filters typically used in signal processing. Instead, they generate wavelet coefficients. Adhering to the Nyquist theorem, the data is upsampled by a factor of 2. This process yields two types of wavelet coefficients: Detailed Coefficients ( $cD$ ) and Approximation Coefficients ( $cA$ ), which respectively contain high-frequency and low-frequency information.

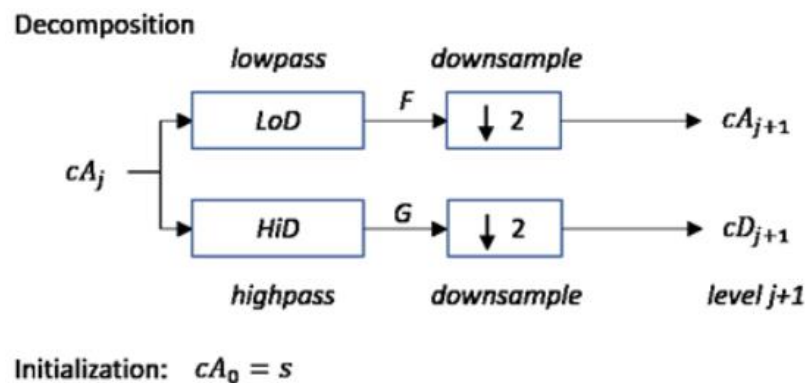


Figure 2.2. One-Dimensional DWT

## 2.4 Multi-level Decomposition

Multi-level decomposition [15], also often referred to as multilevel modeling, hierarchical modeling, or nested design, is a statistical method used for handling data that is organized at more than one level or that has a hierarchical or clustered structure. These methods recognize the existence of data hierarchies by allowing for residual components at each level in the hierarchy.

While a single-level decomposition is sufficient for edge detection, which is valuable for extracting features in image processing, multilevel decomposition offers superior data compression capabilities. This is particularly beneficial when dealing with noisy images. At the initial level of decomposition, the detected edges may be difficult to discern. However, as the number of levels increases, the subsequent levels contain progressively more valuable data with reduced noise.

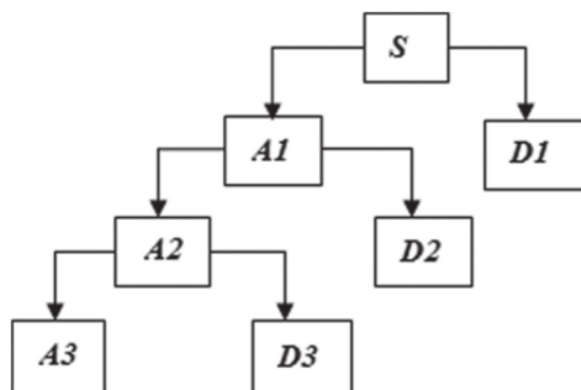


Figure 2.3. Wavelet Decomposition tree

## 2.5 Thresholding in Wavelets

Once the image has been decomposed into wavelet coefficients, the denoising process involves applying a thresholding operation [18, 19] to the detail coefficients. Thresholding involves identifying coefficients that exceed a certain threshold value and then either removing or shrinking them. By effectively suppressing the noisy coefficients while retaining the significant image details, the

DWT-based de-noising technique achieves a balance between noise reduction and preservation of important image features. Subsequently, thresholding methods are employed to mitigate the noise within each of these coefficients. Two types of thresholding methods are commonly used: hard and soft thresholding's.

**Soft thresholding.** The soft thresholding function operates by examining the argument and determining whether its absolute value is smaller than a specified threshold. If the absolute value is indeed smaller, the argument is converted to zero.

However, if the absolute value exceeds the threshold, the argument is reduced by the threshold value. This function can be mathematically represented as follows:

$$\eta_T(x) = \text{sgn}(x) * \max(|x| - T, 0) \quad (2.5)$$

**Hard thresholding.** In contrast to soft thresholding, hard thresholding is a discontinuous method. It retains the input coefficients if they are greater than the threshold value, but sets the coefficient values to zero if they are below the threshold value. The function for hard thresholding can be expressed as follows:

$$\psi_t(x) = x * 1 * 1\{|x| > T\} \quad (2.6)$$

Thresholding selectively eliminates noise from the detailed coefficients at each level, while leaving the approximation coefficients unaffected.

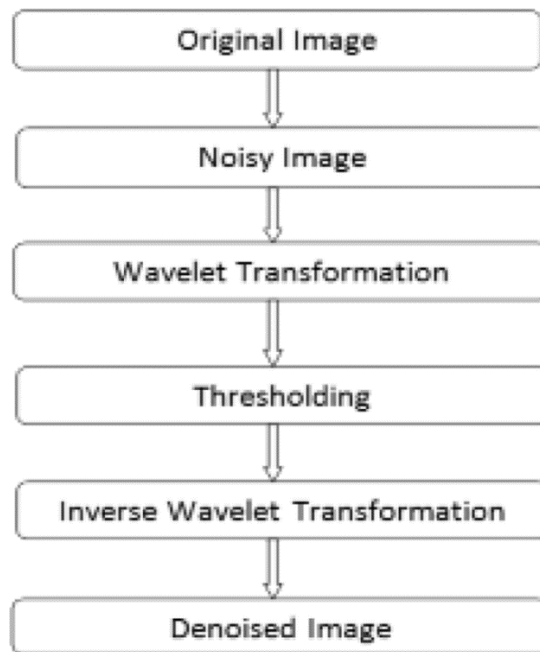


Figure 2.4. Image de-noising using wavelet transform algorithm

## 2.6 Inverse Wavelet Transform

The Inverse Wavelet Transform (IWT) [14, 19] is a mathematical operation that reverses the process of the Wavelet Transform. The IWT, also known as the synthesis step, reconstructs the original signal or image from its wavelet coefficients.

The process of the IWT involves combining the wavelet coefficients obtained from the Wavelet Transform back into the original signal or image. It follows a hierarchical structure, starting from the highest frequency components and gradually reconstructing the lower frequency components.

The IWT can be performed using different wavelet basis functions. These wavelet functions determine the time-frequency resolution and other properties of the reconstructed signal.

The IWT is particularly useful in applications such as signal compression, de-noising, image reconstruction, and feature extraction. By performing the IWT, one can obtain a representation of the original signal or image in the time or spatial domain, allowing further analysis or visualization.

Wavelet Transform and its inverse, the IWT, form a pair of operations that preserve the information of the original signal or image. This property makes them valuable tools in various fields of signal and image processing.

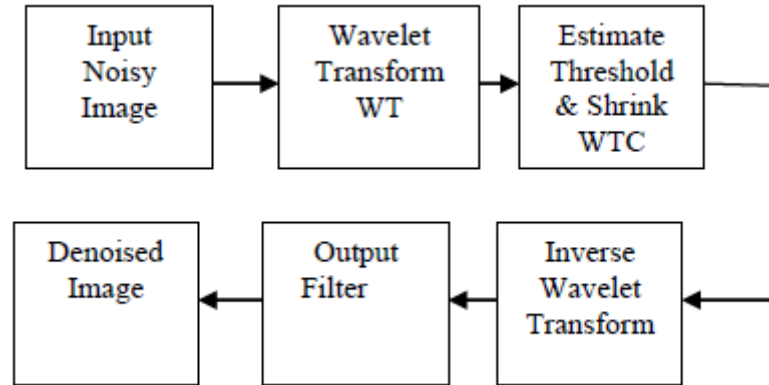


Figure 2.6. Basic scheme of IWT

## 2.7 Conclusion

Wavelet transforms, particularly in the form of Discrete Wavelet Transform (DWT) and Inverse Wavelet Transform (IWT), have emerged as indispensable tools in numerous applications including signal and image processing, data compression, and noise reduction. Wavelet transforms, compared to classical Fourier transforms, offer superior flexibility and precision as they allow for both time and frequency localization, thereby providing a multi-resolution analysis.

The DWT, by dividing a signal into a coarse approximation and detail information, serves to efficiently capture both high-frequency and low-frequency components of a signal. This division helps isolate the noise from the signal in many real-world applications, making DWT especially beneficial for noise reduction and signal enhancement. The IWT, being the reverse process of DWT, is used to reconstruct the original signal from wavelet coefficients, ensuring that no information is lost during the process of transformation and reconstruction.

The process of thresholding, which follows the DWT, allows for further noise reduction. This is achieved by setting all wavelet coefficients that are less than

a certain threshold to zero, effectively eliminating negligible details that are often attributed to noise.

Finally, both single-level and multilevel decompositions play a significant role in the practical application of wavelet transforms. Single-level decomposition divides the signal into one level of approximation and detail, while multilevel decomposition further divides the approximation, providing a deeper level of signal analysis.

### **3 NEURAL NETWORKS IN DIGITAL FILTERING**

Neural networks have emerged as powerful tools in various fields of computer science and artificial intelligence, including image processing and filtering. It's used in a lot of areas, but in image noise filtering especially, neural networks have shown remarkable performance.

Traditional approaches to image noise filtering [21] involve handcrafted algorithms that rely on specific assumptions about the noise characteristics. However, these methods may struggle to handle complex noise patterns and variations in different image domains. Neural networks, on the other hand, offer a data-driven approach to noise filtering, allowing for more robust and adaptive solutions.

In previous years, significant advancements have been made in leveraging neural networks for image noise filtering tasks. Convolutional neural networks (CNNs) have been particularly successful in learning and extracting meaningful features from noisy images. By training on large datasets containing both noisy and clean images, these networks can effectively learn the underlying noise patterns and enhance the quality of the corrupted images.

Moreover, the flexibility and scalability of neural networks enable the development of more sophisticated models for image noise filtering.

#### **3.1 Using Deep Learning neural networks**

Deep learning, [20] a subset of machine learning, leverages artificial neural networks to automatically learn hierarchical representations of data. With the availability of large-scale annotated datasets and computational resources, deep learning has revolutionized the field of image processing. Traditional image denoising methods often rely on handcrafted features and assumptions about noise characteristics, limiting their effectiveness. In contrast, deep learning approaches

can automatically learn noise patterns from large amounts of training data, enabling more accurate and adaptive noise filtering.

To perform image noise filtering, deep learning models are trained using pairs of noisy and clean images. The goal is to teach the network to capture the complex mapping between noisy input images and corresponding clean outputs. The training process involves optimizing the network parameters based on a specified loss function, typically minimizing the pixel-wise difference between the network's output and the clean target image. Especially in this case, the use of deep convolutional neural networks (CNNs) is common due to their ability to efficiently capture spatial correlations in images.

One of the key advantages of deep learning for image noise filtering is its end-to-end learning capability. Unlike traditional methods that involve multiple stages (such as noise estimation and filtering), deep learning models can directly map noisy images to de-noised outputs. By learning an end-to-end mapping, the network can implicitly model noise characteristics and adaptively remove noise while preserving important image details. This end-to-end approach eliminates the need for handcrafted features and intermediate steps, leading to more efficient and effective noise filtering.

Deep learning techniques are versatile and can be trained to handle various types of image noise, including additive noise (such as Gaussian noise), Poisson noise, salt-and-pepper noise, and more. By exposing the network to diverse noise patterns during training, it can learn to generalize and effectively remove different types of noise in real-world scenarios.

Another notable advantage of deep learning-based image noise filtering is its potential for real-time and parallel processing. With the availability of modern graphics processing units (GPUs) and specialized hardware accelerators, deep learning models can be efficiently implemented and deployed for fast and parallel computation. This enables real-time noise filtering, making it feasible for applications that require low-latency processing.



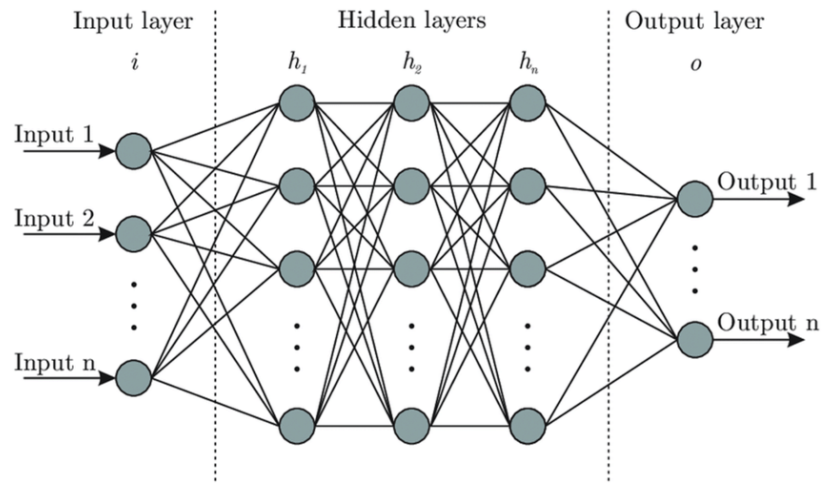


Figure 3.1. Simple Deep Learning network scheme

### 3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning algorithms designed to process and analyse visual data efficiently. They are inspired by the organization of the visual cortex in the human brain, allowing them to capture spatial dependencies and hierarchical representations in images. Unlike traditional neural networks, CNNs exploit the concept of convolution to perform localized feature extraction.

CNNs consist of multiple convolutional layers that extract various features from the input images. Each layer comprises learnable filters or kernels that convolve across the input, performing element-wise multiplications and summations. The resulting feature maps highlight specific patterns, edges, or textures at different spatial scales. These convolutional layers enable CNNs to automatically learn relevant image representations for noise filtering.

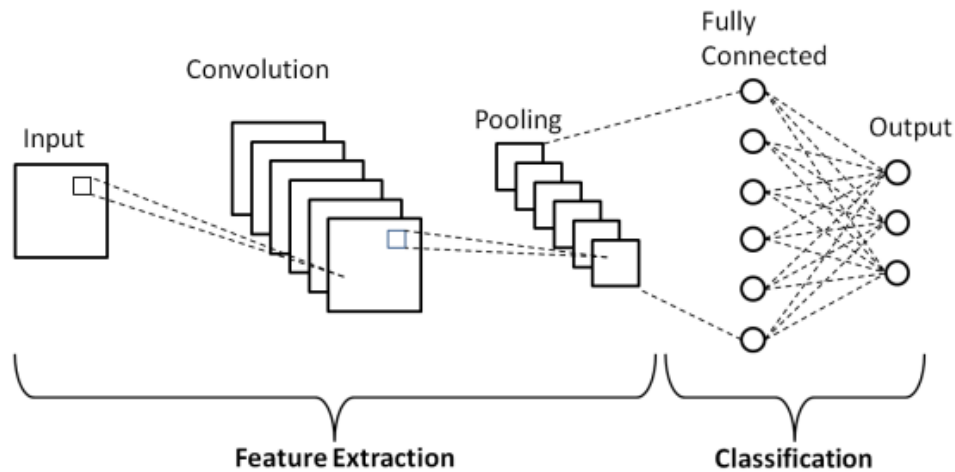


Figure 3.2. Convolutional Neural Network scheme

The architecture of a Convolutional Neural Network (CNN) consists of two primary components.

- Convolutional tool that effectively separates and identifies various features present in an image, a process commonly known as Feature Extraction. This feature extraction network typically comprises multiple pairs of convolutional or pooling layers.
- Fully connected layer that takes the output generated by the convolution process and employs it to predict the class of the image based on the features extracted in earlier stages.
- The CNN model's objective in feature extraction is to reduce the number of features within a dataset. It accomplishes this by generating new features that encapsulate the essential information contained in the original set of features. The CNN architecture diagram typically illustrates the presence of multiple CNN layers, each contributing to the overall feature extraction process.

### 3.2.1 Convolution Layers

A CNN architecture consists of three fundamental types of layers: convolutional layers, pooling layers, and fully – connected layers (FCL). These layers are stacked together to form the structure of the CNN. However, apart from these three layers, two additional crucial components contribute to the overall functionality of the network [20]. These components are the dropout layer and the activation function, each playing a significant role in the CNN’s performance.

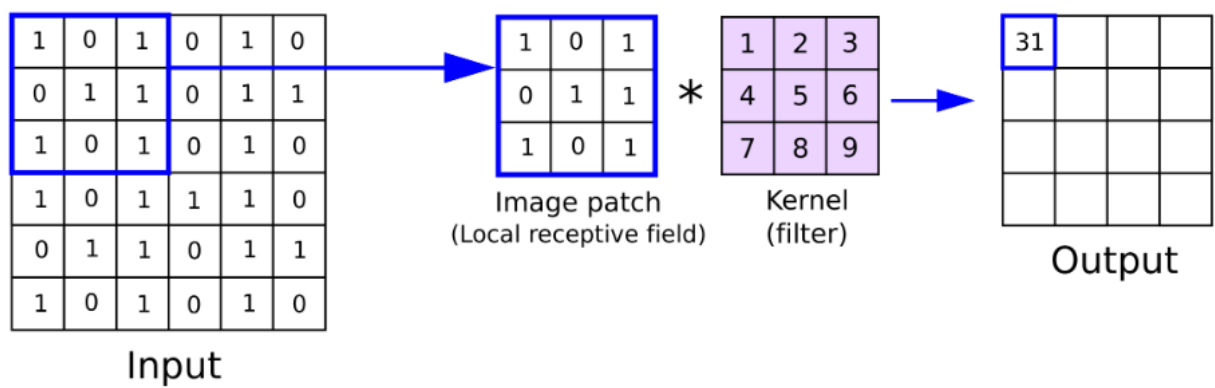


Figure 3.3. Convolution between a kernel (filter) and an image

The initial layer in the convolutional neural network (CNN) serves as the primary stage for extracting diverse features from input images. Within this layer, the convolution operation takes place, involving the input image and a filter of specific dimensions, typically  $M \times M$ . Through the process of sliding the filter across the input image, a dot product is computed between the filter and different segments of the input image, relative to the size of the filter ( $M \times M$ ).

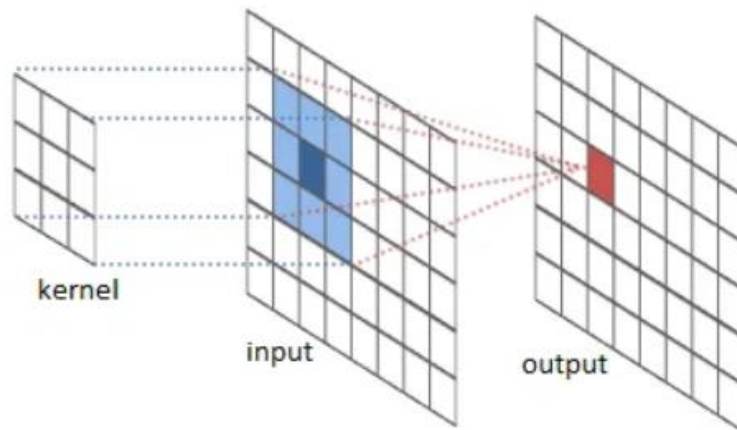


Figure 3.4. Convolution process

The resulting output is referred to as the feature map, which provides valuable information regarding various characteristics of the image, such as corners and edges. Subsequently, this feature map is passed on to subsequent layers to further learn and identify additional features within the input image.

The convolution layer within a CNN effectively transmits the outcome to the subsequent layer after performing the convolution operation on the input. Convolutional layers in CNNs offer significant advantages by preserving the spatial relationship between pixels, ensuring that important spatial details are retained throughout the network's processing.

### 3.2.2 Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The main purpose of this layer is to reduce the size of the convolved feature map, thereby minimizing computational costs. It achieves this by reducing the interconnections between layers and independently operating on each feature map. Various types of Pooling operations exist, depending on the chosen method. The primary function of Pooling is to condense the information extracted by the convolution layer.

In Max Pooling, the largest element from the feature map is selected. Average Pooling computes the average of elements within a predetermined image section. Sum Pooling calculates the total sum of elements in the predefined section. The Pooling Layer commonly acts as a connector between the Convolutional Layer and the Fully Connected Layer (FCL).

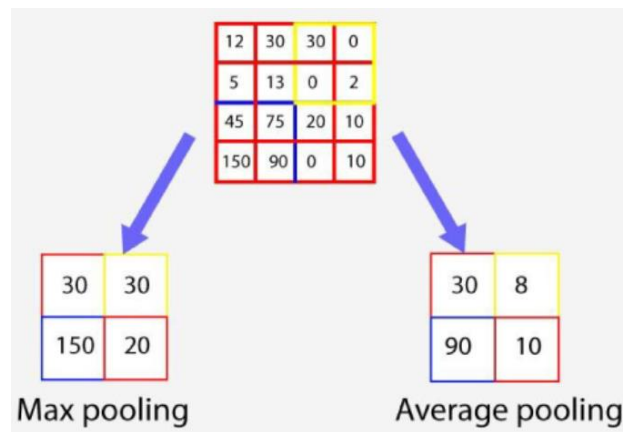


Figure 3.5. Matrix formation using Max-pooling and average pooling

The Convolutional Neural Network (CNN) model enables the extraction of generalized features by the convolution layer. This allows the network to independently recognize the extracted features. Consequently, this aids in reducing computations within the network.

### 3.2.3 Fully Connected Layer

The Fully Connected layer (FCL) serves as a connection between neurons from different layers in a neural network. Typically, these layers are positioned before the output layer and are found towards the end of a Convolutional Neural Network (CNN) architecture.

The Fully Connected layer comprises weights, biases, and neurons. Its purpose is to receive the flattened input image from the preceding layers and process it. This involves performing mathematical operations through a series of FCL. At this stage, the classification process begins to unfold. Connecting two FCL is

advantageous as it generally yields better performance compared to a single connected layer. These layers within a CNN contribute to reducing the level of human supervision involved in the network's operations.

### **3.2.4. Dropout**

Typically, connecting all features to the fully connected (FCL) layer of a model can lead to overfitting on the training dataset. Overfitting occurs when a model performs exceptionally well on the training data but struggles when presented with new, unseen data.

To address this issue, a dropout layer is often employed. This layer randomly drops a subset of neurons from the neural network during the training process, effectively reducing the model's size. By setting a dropout rate of 0.3, approximately 30% of the nodes are randomly dropped from the neural network.

Introducing a dropout layer helps improve the performance of machine learning models by mitigating over fitting. By making the network simpler through dropout, the model becomes more robust and less prone to over-relying on specific features or patterns observed in the training data.

### **3.2.5 Activation Functions**

The activation function stands as one of the crucial parameters within the CNN model, enabling it to learn and approximate complex relationships among network variables. Essentially, it plays a role in determining which information within the model should be activated in the forward direction, while disregarding others towards the end of the network.

By introducing non-linearity, the activation function enhances the network's capabilities. Among the commonly utilized activation functions are Softmax and Sigmoid. Each of these functions serves specific purposes. In the case of a binary

classification CNN model, Sigmoid and Softmax functions are preferred, while Softmax is generally used for multi-class filtering. In simpler terms, activation functions in a CNN model decide whether a neuron should be activated or not. They ascertain the significance of the input to the predictive process through mathematical operations.

### 3.3 U-Net architecture for CNN

The U-Net architecture [23], originally proposed for biomedical image segmentation, has gained popularity in various image processing tasks, including image noise filtering. The architecture derives its name from its characteristic U-shaped structure, which consists of an encoder path and a decoder path.

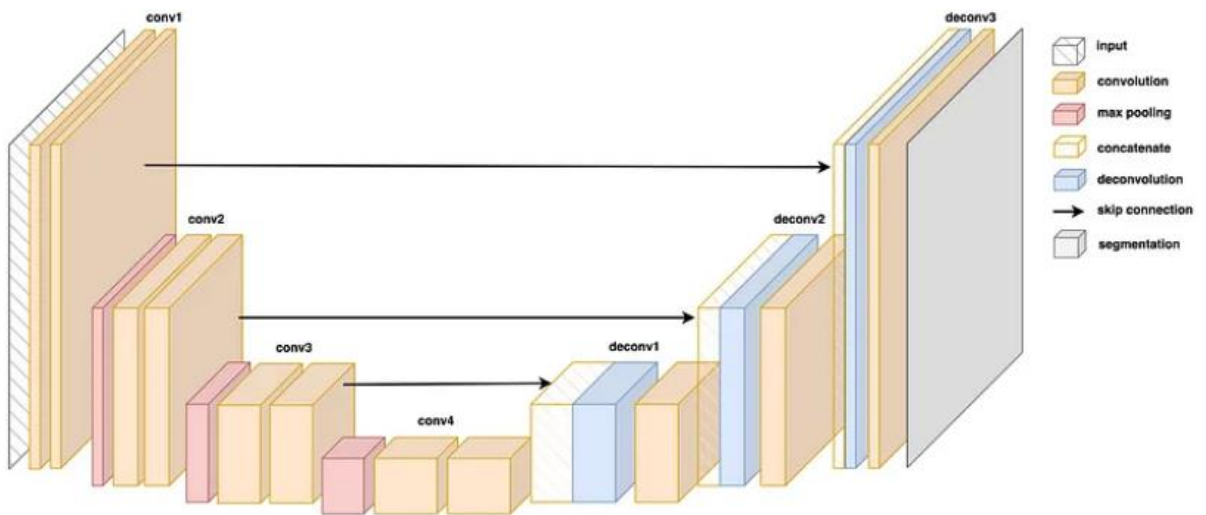


Figure 3.6. Representation of U-Net architecture

#### 3.3.1 Encoder Path

The encoder path of the U-Net architecture [20, 23] is responsible for capturing hierarchical features from the input image. It consists of several convolutional layers, each followed by a non-linear activation function. The encoder gradually reduces the spatial resolution of the input image while increasing the number of feature channels, enabling the extraction of abstract representations.

The encoder path consists of a series of convolutional layers. Each convolutional layer applies a set of filters to the input image, convolving them across the spatial dimensions. The filters learn to extract different features, such as edges, textures, or patterns, from the image. As the encoder progresses deeper into the network, the number of filters typically increases, allowing for a more complex and abstract representation of the input.

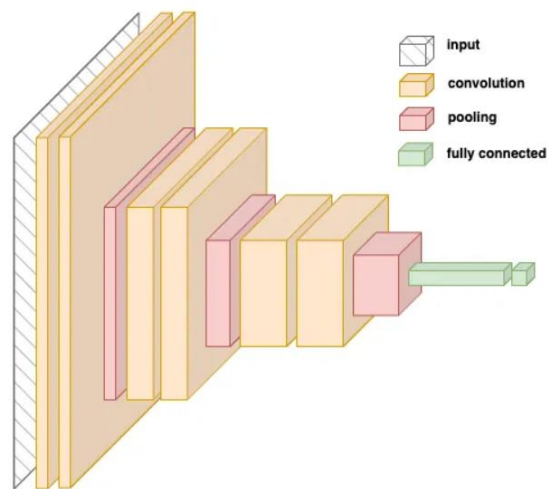


Figure 3.7. Encoder Path used for image filtering

The process involves utilizing convolutional layers and pooling layers to achieve this. The convolution layer operates by traversing across each pixel in an image, employing a mapping or kernel. This mapping is acquired during the model-training phase. Afterwards, through the utilization of a predetermined function, the pooling layers effectively decrease the output's dimensionality.



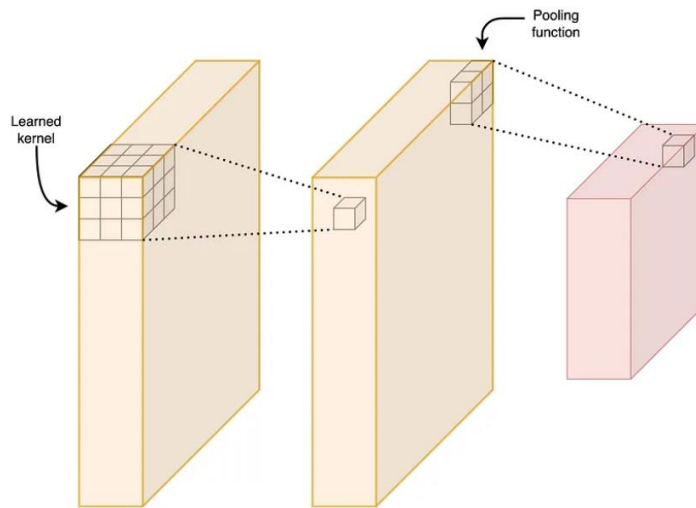


Figure 3.8. Convolutional and pooling layers

By combining multiple convolutional and pooling layers, we can extract finer details, progressing from edges and colors to objects like houses, trees, and vehicles. The network learns which features are important for filtering and creates a compact representation of the image.

### 3.3.2 Decoder Path

The decoder path of the U-Net architecture aims to recover the spatial resolution and reconstruct the filtered image based on the encoded features. It consists of upsampling operations, which progressively increase the spatial resolution, and concatenation of feature maps from the corresponding encoder path layers. This design allows the decoder to exploit both low-level and high-level features, facilitating accurate reconstruction of the filtered image.

As illustrated in Figure 4.7, this section start after the conv4 block. In the decoding process, the objective is to reconstruct an image using the compact representation. Similar to the encoder, the decoder has convolutional blocks, and it's own deconvolution layers that stands to enhance the image's quality and clarity.

### 3.3.3 Skip connections

A distinctive feature of the U-Net architecture is the presence of skip connections [24] that connect the corresponding encoder and decoder path layers. These connections allow for the direct propagation of spatial information from the encoder to the decoder, enabling precise localization of image details during the reconstruction process. Skip connections help to overcome the information loss that can occur in deeper layers and improve the model's ability to preserve fine image structures.

What is critical is that the position of the feature retrieved by convolutional layers be passed. That is, the skip connections indicate the network where the features in the picture came from.

The process involves the concatenation of the final layer in the convolutional block with the initial layer of the corresponding deconvolutional block. It is important to note that the U-Net architecture is designed to be symmetrical, ensuring that the dimensions of the opposing layers are identical. This symmetry facilitates the seamless combination of these layers into a unified tensor. Subsequently, the convolution operation is performed conventionally by applying the kernel across the concatenated tensor.

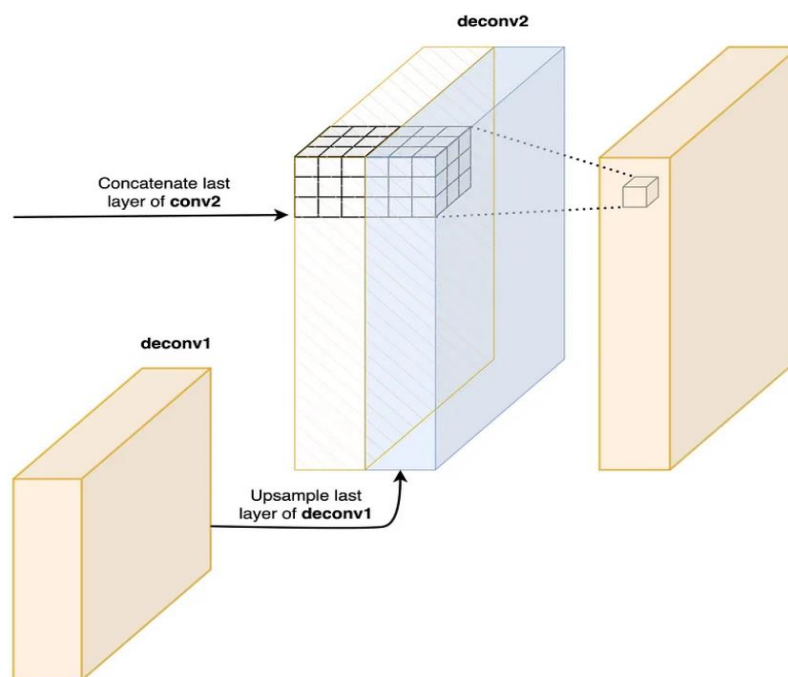


Figure 3.7. Concatenating layers

At the core of the U-Net architecture [20] lies the fundamental concept of concatenation. This operation serves to merge two crucial elements of information:

- Feature extraction: The features extracted from the preceding layer are transmitted to the up sampled layer (depicted in blue).
- Feature localization: The positional information of the feature is relayed from the corresponding convolutional layer (highlighted in orange).

By effectively amalgamating these distinct pieces of information, can be enhance the efficacy of semantic models and concurrently diminish the quantity of data necessary for training the network

### **3.4 Wavelets basis Neural network**

Neural networks combined with wavelets have emerged as a powerful approach for image noise filtering. This hybrid technique leverages the strengths of both neural networks and wavelet analysis to achieve effective noise filtering results.

In this approach, a neural network is trained to learn the relationship between the wavelet coefficients of noisy images and their corresponding clean versions. The network takes the noisy wavelet coefficients as input and outputs the denoised wavelet coefficients. By learning from a large dataset of noisy-clean image pairs, the neural network learns to generalize and effectively remove noise from unseen images.

Wavelets play a crucial role in this process by providing a multi-scale analysis of the image. The image is decomposed into different frequency bands using the wavelet transform, and the neural network operates on the wavelet coefficients at each scale. This allows the network to capture both local and global image details, which can aid in better noise estimation and removal.

The hybrid neural network-wavelet approach offers several advantages. Firstly, the wavelet decomposition provides a sparse representation of the image, which helps in reducing the computational complexity of the neural network. The

neural network focuses on learning the relationship between the wavelet coefficients, rather than operating on the entire image, which can lead to more efficient and faster filtering.

Secondly, the wavelet decomposition enables the neural network to handle different types of noise at various scales. Wavelets can effectively capture noise characteristics in different frequency bands, allowing the neural network to adapt its de-noising strategy accordingly.

Learned neural network can be applied to real-time de-noising tasks since it operates in the wavelet domain, where the filtering process is usually faster compared to pixel-based approaches.

Currently, approaches are being actively developed to improve the efficiency of wavelet transformations using neural networks. The direction is newly emerged and has many ways of further development.

### **3.5 Conclusion**

Neural networks, specifically convolutional neural networks (CNNs), have shown great promise in noise filtering applications. The ability of CNNs to automatically learn hierarchical representations from data makes them well-suited for handling complex noise patterns in various domains, including image and signal processing.

U-Net is a specific CNN architecture that has gained popularity in the field of image segmentation, which is closely related to noise filtering tasks. It is characterized by a U-shaped architecture that includes both a contracting path and an expansive path. The contracting path captures context and spatial information through convolutional and pooling operations, while the expansive path enables precise localization of the objects or features being segmented.

The U-Net architecture has demonstrated excellent performance in various remote sensing image denoising tasks, where noise filtering is often a crucial preprocessing step. The network's ability to learn intricate features and capture fine details makes it particularly effective for denoising applications.

## **4 DIGITAL NOISE FILTERING REALISATION ON HYPERSPPECTRAL IMAGES**

Image restoration is a crucial and long-standing problem in remote sensing, aiming to recover the original clean image ( $x$ ) from its degraded observation ( $y$ ). Over the years, numerous methods have been proposed to address image restoration, considering both prior modeling and discriminative learning approaches. In recent times, convolutional neural networks (CNNs) have gained significant attention and have achieved remarkable performance in various image restoration tasks, including image denoising [20].

The widespread adoption of CNNs in image restoration can be attributed to two main factors. Firstly, existing CNN-based solutions have demonstrated substantial improvements over other methods, particularly in simple tasks like image denoising. Secondly, recent studies have shown that CNN-based denoisers can be integrated into model-based optimization methods to tackle more complex image restoration tasks [21, 22]. This integration has further encouraged the extensive use of CNNs in this field. Essentially, in image restoration, a CNN functions as a mapping from a degraded observation to a latent clean image. To ensure consistent input and output image sizes, a common strategy involves employing a fully convolutional network (FCN) by removing pooling layers.

Typically, a larger receptive field, which considers more spatial context, enhances the restoration performance. However, for FCNs without pooling, enlarging the receptive field can be achieved either by increasing the network depth or by using larger filters. It is worth noting that these approaches also lead to higher computational costs.

### **4.1 Cascade Wavelets CNN model introduction**

In this research paper, I am introducing a novel model called the Cascade Wavelet CNN (CWCNN) with the aim of enhancing the receptive field to achieve a

better balance between performance and efficiency. CWCNN model is built upon the well-known U-Net architecture, which consists of a contracting subnetwork and an expanding subnetwork. In the contracting subnetwork, incorporate the discrete wavelet transform (DWT) as a replacement for traditional pooling operations. By using DWT, we ensure that the downsampling process is reversible, thus preserving all the information within the feature maps.

In the expanding subnetwork, we employ the inverse wavelet transform (IWT) to upsample low-resolution feature maps and obtain high-resolution counterparts. This enables to enhance the representation of features while minimizing the computational burden.

The main principle underlying CWCNN architecture is the incorporation of a CNN block following each level of DWT. In Fig. 4.1, we can observe that each CNN block consists of four layers of FCN, omitting pooling, and utilizes all subband images as inputs. In contrast, distinct CNNs are employed for the low-frequency and high-frequency bands within the deep convolutional framelets [20]. It is important to note that the subband images following DWT remain interdependent, and disregarding their interdependence could have detrimental effects on the restoration performance.

When the input image  $x$  comes, are employing a two-dimensional discrete wavelet transform (2D DWT) [22] using four convolutional filters: a low-pass filter  $f_{LL}$ , and three high-pass filters  $f_{LH}$ ,  $f_{HL}$ , and  $f_{HH}$ . By this, image  $x$  decompose into the four subband images:  $x_{LL}$ ,  $x_{LH}$ ,  $x_{HL}$ , and  $x_{HH}$ . These four filters have predetermined parameters and utilize a convolutional stride of 2 during the transformation process. Take the Haar wavelet as an illustration, where these four filters are specifically defined.

$$f_{LL} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, f_{LH} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}, f_{HL} = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}, f_{HH} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (4.1)$$

Vectors  $f_{LL}$ ,  $f_{LH}$ ,  $f_{HL}$ , and  $f_{HH}$  are mutually perpendicular to each other and collectively form a  $4 \times 4$  invertible matrix. The DWT operation can be defined as follows:  $x_{LL}$  is obtained by convolving  $x$  with  $f_{LL}$  and then downsampling the result by a factor of 2. Similarly,  $x_{LH}$  is obtained by convolving  $x$  with  $f_{LH}$  and downsampling,  $x_{HL}$  is obtained by convolving  $x$  with  $f_{HL}$  and downsampling, and  $x_{HH}$  is obtained by convolving  $x$  with  $f_{HH}$  and downsampling. Here, the symbol  $\otimes$  represents the convolution operator, and  $\downarrow 2$  indicates the standard downsampling operation with a factor of 2. In simple terms, the DWT utilizes four fixed convolution filters with a stride of two to perform downsampling. Additionally, as per the principles of the Haar transform [56], the  $(i, j)$ -th values of  $x_{LL}$ ,  $x_{LH}$ ,  $x_{HL}$ , and  $x_{HH}$  after a 2D Haar transform can be expressed as

$$\begin{cases} x_{LL}(i, j) = x(2i - 1, 2j - 1) + x(2i - 1, 2j) + x(2i, 2j - 1) + x(2i, 2j) \\ x_{LH}(i, j) = -x(2i - 1, 2j - 1) - x(2i - 1, 2j) + x(2i, 2j - 1) + x(2i, 2j) \\ x_{HL}(i, j) = -x(2i - 1, 2j - 1) + x(2i - 1, 2j) - x(2i, 2j - 1) + x(2i, 2j) \\ x_{HH}(i, j) = x(2i - 1, 2j - 1) - x(2i - 1, 2j) - x(2i, 2j - 1) + x(2i, 2j). \end{cases} \quad (4.2)$$

The downsampling process is utilized in order to maintain the biorthogonal characteristic of the DWT. This property ensures that the original image, denoted as  $x$ , can be reconstructed accurately through the IWT without any loss of information. This reconstruction can be expressed as  $x = \text{IWT}(x_{LL}, x_{LH}, x_{HL}, x_{HH})$ , where  $x_{LL}$ ,  $x_{LH}$ ,  $x_{HL}$ , and  $x_{HH}$  represent the wavelet coefficients obtained from the DWT. Specifically, for the Haar wavelet, the IWT can be defined as follows:

$$\begin{cases} x(2i - 1, 2j - 1) = (x_{LL}(i, j) - x_{LH}(i, j) - x_{HL}(i, j) + x_{HH}(i, j))/4 \\ x(2i - 1, 2j) = (x_{LL}(i, j) - x_{LH}(i, j) + x_{HL}(i, j) - x_{HH}(i, j))/4 \\ x(2i, 2j - 1) = (x_{LL}(i, j) + x_{LH}(i, j) - x_{HL}(i, j) - x_{HH}(i, j))/4 \\ x(2i, 2j) = (x_{LL}(i, j) + x_{LH}(i, j) + x_{HL}(i, j) + x_{HH}(i, j))/4 \end{cases} \quad (4.3)$$

In order to facilitate subsequent processing the subband images  $x_{LL}$ ,  $x_{LH}$ ,  $x_{HL}$ , and  $x_{HH}$  can be sequentially decomposed using discrete wavelet transformation

(DWT). To obtain the outcomes of a two-level WPT, DWT is independently applied to decompose each subband image  $x_i$  (where  $i = LL, LH, HL, \text{ or } HH$ ) into four subband images:  $x_{i,LL}$ ,  $x_{i,LH}$ ,  $x_{i,HL}$ , and  $x_{i,HH}$ . This recursive process can be extended to obtain the results of three or more levels of WPT.

Each layer of the CNN block comprises convolution using  $3 \times 3$  filters (Conv), batch normalization (BN), and rectified linear unit (ReLU) [24] operations. For the last layer of the final CNN block, Conv without BN and ReLU is employed to predict the residual image. Fig 4.2 provides an overview of the comprehensive architecture of CWCNN, which encompasses both a contracting subnetwork and an expanding subnetwork. In essence, CWCNN introduces modifications to the U-Net framework from three distinct perspectives.

Denote by  $\Theta$  the network parameters of CWCNN, and  $F(y; \Theta)$  be the network output. Let  $\{(y_i, x_i)\}_{i=1}^N$  be a training set, with  $y_i$  representing the  $i$ -th input picture and  $x_i$  being the associated ground-truth image. The learning goal function for CWCNN is then given by

$$\mathcal{L}(\Theta) = \frac{1}{2N} \sum_{i=1}^N \|F(y_i, \Theta) - x_i\|_F^2 \quad (4.4)$$

To train the CWCNN, we employ the ADAM algorithm [25] which minimizes the objective function.

Program realisation for hyperspectral image filtering is done by Python and usage of deep learning and data processing libraries. For creation and working with CCN - basis of CWCCN model, was used TensorFlow as the most reliable. Spectral library was used for opening and reading the hyperspectral image file, displaying images, and saving created RGB images and it's cumulative histograms which neural network will use to work with. Utility libraries like Matplotlib, Panda and NumPy are employed for data visualization representation and processing, respectively.



The key functions in the whole program are:

1. `hdr.read_bands([nr, ng, nb])`: Function is creating a standard RGB image from the hyperspectral image by using the bands corresponding to the red, green, and blue wavelengths.
2. `cv2.calcHist()`: The function calculated histogram of the image. A histogram of shows the distribution of pixel intensities (number of pixels of each intensity value from 0 to 255).
3. `train_test_split(noisy_array_paths, gt_array_paths, test_size, random_state)`: This function the splits dataset into training and testing sets.
4. `create_model()` :This function creates a sequential model with encoder-decoder architecture style, where the upsampling is done by `Conv_bkloc`: downsampling is done by DWT in the first half, then upsampling is done in the second half by IWT.
5. `model.fit()` : Function trains the model for a fixed number of epochs. It uses training and validation data, with the previously defined steps per epoch and callbacks.

All of the above functions are part of a sequential system that takes care of loading data, preprocessing datasets, building a neural network model, and then training and validating it using wavelet transforms. The loss and accuracy curves are plotted using the Matplotlib library and the model is saved in .h5 format. Developed model onsistent with the stated theoretical base, effectively copes with the task of noise filtering on satellite hyperspectral images and can be improved in the future.

A complete implementation for training a noise filtering model on hyperspectral images is provided in Appendix B.

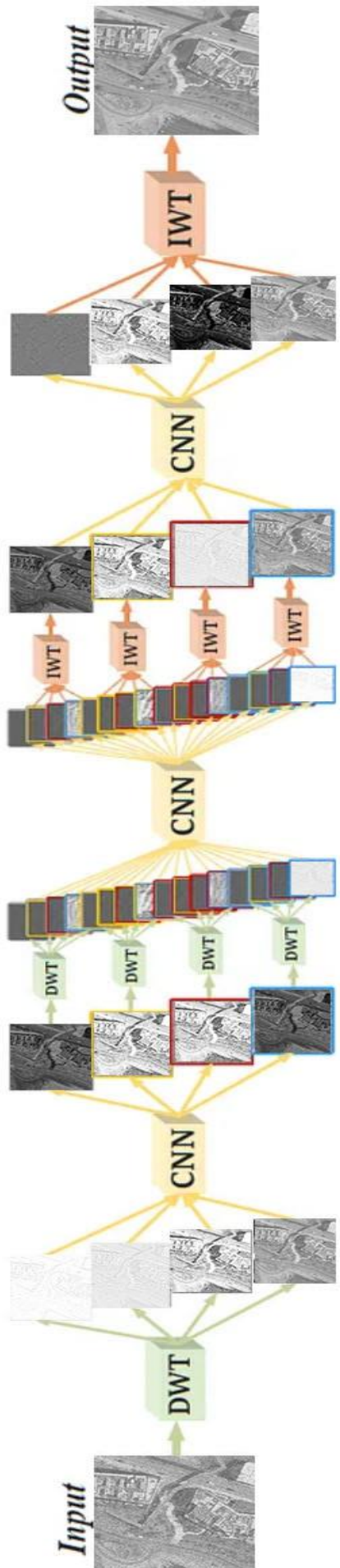


Figure 4.1. CWCNN filtering process

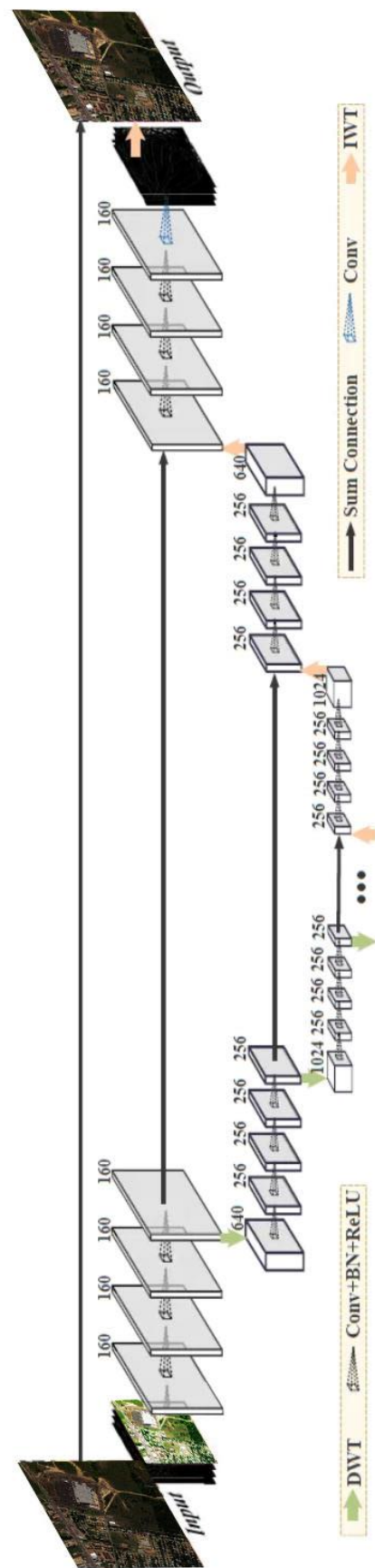


Figure 4.2. CWCNN architecture

## 4.2 Data preparation and model training details

One open source hyperspectral imaging dataset, Urban\_210, was used for the study. The image consists of a 307 by 307 pixel grid, where each pixel represents an area of 2x2 square meters. There are a total of 210 different wavelengths in this image, ranging from 400 nm to 2500 nm, resulting in a spectral resolution of 10 nm. However, some channels (specifically channels 1-4, 76, 87, 101-111, 136-153, and 198-210) were excluded due to the presence of dense water vapor and weathering. As a result, we have 162 remaining channels to work with.

The training of the network involved utilizing mini-batches, where each batch contained 256 examples. The training process was iterated over 100 epochs, incorporating batch normalization and data augmentation techniques. These methods were employed to enhance the neural network's performance, stability, and accuracy.

All the experimental work was conducted in the Colab Research environment, utilizing a computing setup equipped with an A100 GPU and 24 GB of RAM. In order to process the neural network, the hyperspectral image was resized to dimensions of 256x256 and encompassed 3 RGB channels.

Let's look at the code in Appendix A, where we can follow the process of preparing the Urban\_210 dataset for further work with the CWCNN neural network. For a more precise image filtering process, bitmap graphics with a high color depth are best suited, which can offer us a hyperspectral image format (RGB). The code then reads the number of rows, columns, and channels (spectral channels) in the image. It then reads the image data for the RGB bands and displays the RGB image.

We read the number of rows, columns and channels (spectral channels) in an image using ENVI library. Then, the image data for the RGB stripes is displayed as an RGB image.

Hyperspectral imaging captures images in numerous narrow and contiguous spectral bands, resulting in high-dimensional data. To convert a hyperspectral image to a multispectral representation, a subset of spectral bands, corresponding to the

RGB bands, is selected. This reduction in dimensionality simplifies the data and enables visualization using familiar color.

Histogram equalization is then applied to the multispectral image to enhance contrast. The process involves three main steps:

- **Normalized Cumulative Histogram:** The histogram of the image is calculated and normalized by dividing the bin counts by the total number of pixels. A cumulative histogram is computed by summing the normalized histogram values.
- **Pixel Mapping Lookup Table:** A lookup table is created to map each original intensity value to its corresponding equalized value. This mapping is obtained by multiplying the cumulative histogram values by the maximum intensity value (255) and rounding to the nearest integer.
- **Transformation:** The image is flattened into a 1D list, and each pixel value is replaced with its equalized value based on the lookup table. The list is then reshaped back into the original image shape.
- The result is an equalized image with improved contrast and visual quality. This allows you to visualize and analyze spectral data that would otherwise be invisible to the eye.

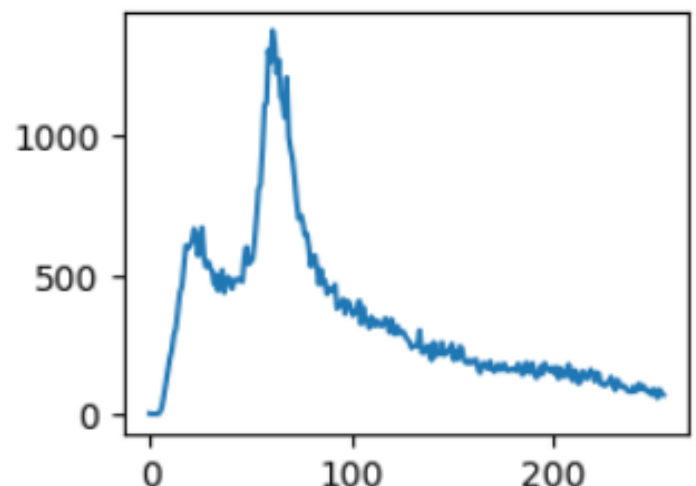


Figure 4.3. Representation of Multispectral image from hyperspectral and it's Histogram

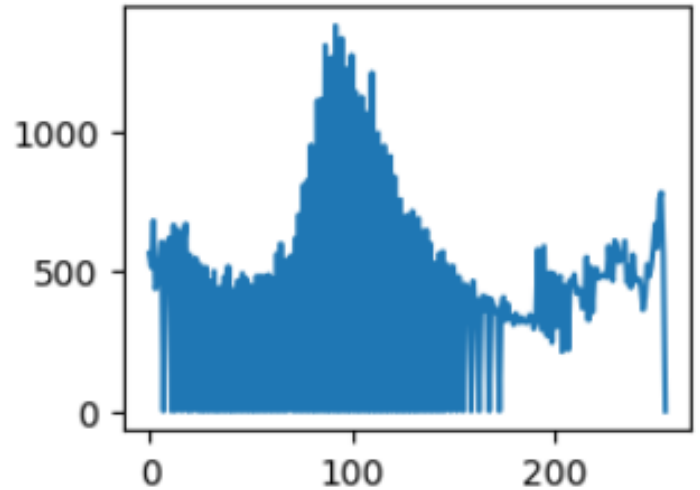


Figure 4.4. Multispectral image and applied Histogram equalization

When we examine these images, we observe the result displayed in Figure 4.3, along with a comprehensive explanation and evaluation of the model's parameters. The output represents a concise overview of a specialized Convolutional Neural Network (CNN) structure tailored for the purpose of effectively filtering out noise artifacts from images.

Based on the provided model summary, here is the description of each section of the neural model:

- **Input Layer:** The input layer receives the input images with a shape of (256, 256, 3). The images used in this study have spatial dimensions of the feature maps 256x256, with 3 channels representing different shades of colors, which is the standard for color images. The decision to use a size of 256x256 is based on previous research findings indicating that patches of this size provide sufficient spatial context for making accurate predictions. To handle the high dimensionality of the images and generate additional training examples, patches are extracted from the entire image and utilized for training the model.

- **Conv\_block:** This section consists of a Convolutional layer, which applies four convolutional operations with 64 filters and a kernel size of 3x3. Each convolutional operation is followed by a ReLU] activation function. The output

shape of this section is (256, 256, 64). This section will be repeated and increase filters numbers two times, but decrease spatial dimensions in a same amount ,because of DWT\_downsampling. Since Conv\_block6, model start using IWT\_upsampling doing reverse process to DWT until Conv\_block 9 takes the same value as origin image.

- DWT\_downsampling: This section performs down-sampling using a downsampling layer on each conv\_block, which reduces the spatial dimensions of the feature maps by a factor of 2, end will repeat every time soon. The output shape after down-sampling is (128, 128, 256).

- Batch Normalization: Batch normalization is applied to normalize the activations of the Conv\_block\_4 layer, resulting in (16, 16, 512) output shape.

- Conv2D\_: A convolution layer that operates in 2 dimensions (height and width). It is used here for further spatial feature extraction from the reshaped data.

- IWT\_upsampling: This section performs inverse wavelet transform (IWT) up-sampling on the feature maps, restoring their spatial dimensions to the original size before down-sampling. The output shape is None (depends on the input size).


- Add: The upsampled feature maps are added element-wise to the feature maps from the corresponding DWT\_downsampling layers, promoting skip connections. The output shape is (32, 32, 512).

The total number of trainable parameters in the model is 58,814,019. During training, the weights and biases in this network are updated. This means that all parameters in the network are trainable and do not remain constant throughout the training.

```

Model: "model"

```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	
conv_block (Conv_block)	(None, 256, 256, 64)	75648	['input_1[0][0]']
dwt_downsampling (DWT_downsampling)	(None, 128, 128, 256)	0	['conv_block[0][0]']
conv_block_1 (Conv_block)	(None, 128, 128, 128)	590208	['dwt_downsampling[0][0]']
dwt_downsampling_1 (DWT_downsampling)	(None, 64, 64, 512)	0	['conv_block_1[0][0]']
conv_block_2 (Conv_block)	(None, 64, 64, 256)	2360064	['dwt_downsampling_1[0][0]']
dwt_downsampling_2 (DWT_downsampling)	(None, 32, 32, 1024)	0	['conv_block_2[0][0]']
conv_block_3 (Conv_block)	(None, 32, 32, 512)	9438720	['dwt_downsampling_2[0][0]']
dwt_downsampling_3 (DWT_downsampling)	(None, 16, 16, 2048)	0	['conv_block_3[0][0]']
conv_block_4 (Conv_block)	(None, 16, 16, 512)	14157312	['dwt_downsampling_3[0][0]']
batch_normalization_20 (Batch Normalization)	(None, 16, 16, 512)	2048	['conv_block_4[0][0]']
conv_block_5 (Conv_block)	(None, 16, 16, 512)	7079424	['batch_normalization_20[0][0]']
conv2d_24 (Conv2D)	(None, 16, 16, 2048)	9439232	['conv_block_5[0][0]']
iwt_upsampling (IWT_upsampling)	(None, 32, 32, 512)	0	['conv2d_24[0][0]']
add (Add)	(None, 32, 32, 512)	0	['iwt_upsampling[0][0]', 'conv_block_3[0][0]']
conv_block_6 (Conv_block)	(None, 32, 32, 512)	7079424	['add[0][0]']
conv2d_29 (Conv2D)	(None, 32, 32, 1024)	4719616	['conv_block_6[0][0]']
iwt_upsampling_1 (IWT_upsampling)	(None, 64, 64, 256)	0	['conv2d_29[0][0]']
add_1 (Add)	(None, 64, 64, 256)	0	['iwt_upsampling_1[0][0]', 'conv_block_2[0][0]']
conv_block_7 (Conv_block)	(None, 64, 64, 256)	1770240	['add_1[0][0]']
conv2d_34 (Conv2D)	(None, 64, 64, 512)	1180160	['conv_block_7[0][0]']
iwt_upsampling_2 (IWT_upsampling)	(None, 128, 128, 128)	0	['conv2d_34[0][0]']
add_2 (Add)	(None, 128, 128, 128)	0	['iwt_upsampling_2[0][0]', 'conv_block_1[0][0]']
conv_block_8 (Conv_block)	(None, 128, 128, 128)	442752	['add_2[0][0]']
conv2d_39 (Conv2D)	(None, 128, 128, 256)	295168	['conv_block_8[0][0]']
iwt_upsampling_3 (IWT_upsampling)	(None, 256, 256, 64)	0	['conv2d_39[0][0]']
add_3 (Add)	(None, 256, 256, 64)	0	['iwt_upsampling_3[0][0]', 'conv_block[0][0]']
conv_block_9 (Conv_block)	(None, 256, 256, 64)	110784	['add_3[0][0]']
conv2d_44 (Conv2D)	(None, 256, 256, 128)	73856	['conv_block_9[0][0]']
conv2d_45 (Conv2D)	(None, 256, 256, 3)	387	['conv2d_44[0][0]']

```

=====
Total params: 58,815,043
Trainable params: 58,814,019
Non-trainable params: 0

```

Figure 4.5. Results of model parameters for Urban\_F210 dataset

### 4.3 Filtration results

The complete implementation can be found in Appendix B. The overall method described involves the following steps: data preparation, model loading, prediction generation, performance evaluation, and result visualization.

- Data Augmentation. Data augmentation techniques such as flipping, rotating, adjusting the hue, brightness, saturation, and contrast of the images are used to create variations of the images to had more data for training

- After image preparation done in previous section, we can work with neural network now. It consist of 3 repeatable parts

- Encoder: The encoder part of the model consists of several Convolutional Blocks (Conv\_blocks) followed by DWT\_downsampling layers. Each Conv\_block consists of multiple 2D convolutional layers, each followed by a Rectified Linear Unit (ReLU) activation function. The purpose of the encoder is to extract hierarchical features from the input image.

- Intermediate Convolutional Block: After the last Conv\_block in the encoder, there is an additional Conv\_block without downsampling, followed by a Conv2D layer with 2048 filters and a 3x3 kernel size.

- Decoder: The decoder part of the model consists of several IWT\_upsampling layers followed by Conv\_blocks. The IWT\_upsampling layers perform inverse wavelet transformations to upsample the feature maps. The feature maps are then combined with the corresponding feature maps from the encoder using skip connections (via Add() operation). The purpose of the decoder is to reconstruct the high-resolution image from the learned features.



Given that the model generates a probability distribution for our image, we proceed by selecting the class that exhibits the highest probability as our predicted image.

After all, we receive output data in form of prediction image, which represent noise filtering efficiency of model. Compare this to the noisy image and ground truth version, it can be seen visual difference that show network work.



Figure 4.6. Ground Truth image (clear image)



(a)



(b)



(c)



(d)



(e)



(f)

Figure 4.7 Urban\_F210 image noise filtering using CWCNN model.  
(a,c,e) – represent Gaussian, Salt & Pepper and it's combine noises.  
(b,d,f) –result of filtering by CWCNN.

## 4.4 Performance evaluation

This comparison uses three separate measurements, namely the Peak Signal to Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Noise Level (Q), to evaluate the effectiveness of noise filtering in a hyperspectral image. PSNR quantifies the relationship between the maximum achievable image power and the noise power generated by the noise reduction process, while SSIM evaluates the similarity between two images in terms of their brightness, contrast and texture. The noise level (Q) corresponds to the amount of noise present in the image.

To evaluate the performance of the CWCNN model, we compared it with supervised methods such as X-y Source Pairs (no model), Mean Filter (MF), and DnCNN [27]. To provide more accurate results, three types of noise were used: Gaussian, Salt and Pepper, as well as a combination of them. Examples Fig 4.7

The data set was divided into training (20%) and test (80%) segments. The results were calculated using a publicly available code specific to the above comparative methods.

As shown in Table 4.1, PSNR, SSIM, and noise figure results are presented for various methods. CWCNN, as shown in Table 4.1, outperforms all other comparison methods for every data set.

The original x, y pairs basically compare the raw data: an image with noise without applying any filtering and Ground Truth, a crisp image.

The CWCNN architecture has much in common with the DnCNN (Denoising Convolutional Neural Network) [27] neural network and is also based on CNN at its core, but the models have gone different ways in achieving the result. When DnCNN uses the ReLU activation function, CWCNN uses the transform wavelet to get the best result, as shown in Table 4.1.

Comparison is important to understand the potential of the whole model and the ability to compete with other solutions.

MF works by replacing the value of each pixel in the image with the mean (average) value of the intensity of the pixels in its vicinity, but since this is not part of any neural network, it may not perform better than expected.

## **4.5 Conclusion**

CWCNN present a unique and effective approach to image analysis tasks by integrating wavelet transforms (DWT and IWT) with the robust capabilities of Convolutional Neural Networks (CNNs). The strengths of these two components combined often outperform either method used separately.

Wavelets provide a multi-resolution analysis of an image. This effectively captures both the coarse-scale global structures (via low-frequency components) and fine-scale local details (via high-frequency components) of an image. When used independently, wavelets are good at signal decomposition and noise reduction but lack the ability to learn and adapt to complex features in the data.

On the other hand, CNNs excel at learning hierarchical features and adapting to complex patterns in data. However, when used alone, they might struggle with multi-scale and multi-resolution image information.

## CONCLUSION

Hyperspectral images possess high-resolution data across multiple spectral ranges but are susceptible to various types of noise due to factors such as limitations in sensors, environmental influences, and transmission errors. This noise can significantly degrade the quality of these images and hinder their subsequent analysis and applications.

To address this issue, I have developed a System of Digital Filtering for Hyperspectral Images. This system incorporates a Convolutional Wavelet Convolutional Neural Network (CWCNN) that combines the reliable capabilities of Convolutional Neural Networks (CNN) with a unique approach to filtering hyperspectral images by integrating wavelet transforms, namely the Discrete Wavelet Transform (DWT) and the Inverse Wavelet Transform (IWT). The synergy between these two components often yields superior results compared to using either method individually.

Wavelets play a crucial role in image analysis by providing different resolutions. They effectively capture both the global structures at coarse scales (through low-frequency components) and the fine local details (through high-frequency components) of the image.

Empirical results of image denoising have demonstrated that CWCNNs frequently outperform traditional filtering methods and some neural networks, showcasing their effectiveness in enhancing image quality.

## REFERENCE

1. Lillesand, T., Kiefer, R. W., & Chipman, J. (2007). *Remote Sensing and Image Interpretation*. John Wiley & Sons.
2. Richards, J. A., & Jia, X. (2006). *Remote sensing digital image analysis: an introduction*. Springer.
3. J. R. Jensen, "Remote Sensing of the Environment: An Earth Resource Perspective," *Proceedings of the IEEE*, March 2013.
4. J. Bioucas-Dias, A. Plaza, G. Camps-Valls, N. Nasrabadi, P. Scheunders, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 2, pp. 6–36, June 2013.
5. P. Ghamisi, N. Yokoya, J. Li, W. Liao, S. Liu, J. Plaza, B. Rasti, and A. Plaza, "Advances in hyperspectral image and signal processing: A comprehensive overview of the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5 no 4, pp. 37–78, Dec 2017.
6. Metzler, G.R., "The Noise Figure of Radio Receivers: Y-Factor Method vs. Amplifier Method," 2020 IEEE Conference on Antenna Measurements & Application.
7. Srinivasan, K.S. and Ebenezer, D., "A New Fast and Efficient Decision-Based Algorithm for Removal of High-Density Impulse Noises," *IEEE Signal Processing Letters*, vol. 14, no. 3, March 2007.
8. Yu, L. and Acton, S.T., "Speckle Reducing Anisotropic Diffusion," *IEEE Transactions on Image Processing*, vol. 11, no. 11, Nov. 2002.
9. Coatrieux, G. and Sankur, B., "A Detailed Analysis of the Mean Filter in the Transform Domain," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 31, no. 1, Feb. 1983.
10. Astola, J. and Kuosmanen, P., "Nonlinear Filters for Image Processing," *IEEE Potentials*, vol. 15, no. 1, Feb.-March 1996.

11. Farsiu, S., Robinson, D., Elad, M., and Milanfar, P., "Adaptive Wiener Filter Super-Resolution of Color Filter Array Images," 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, 2006.
12. Lee, J.S., Kuo, W.J., Chung, P.C., and Chen, T.C., "Adaptive Gaussian Filter for Noise Reduction and Edge Detection," Proceedings of 1996 IEEE International Conference on Neural Networks, 1996.
13. A. Graps, "An Introduction to Wavelets," IEEE Computational Science and Engineering, vol. 2, no. 2, pp. 50-61, Summer 1995. DOI: 10.1109/99.388960.
14. I. Daubechies, "Ten Lectures on Wavelets," IEEE Transactions on Information Theory, vol. 36, no. 5, pp. 961-1005, September 1990. DOI: 10.1109/18.57199.
15. N. G. Kingsbury, "Image Processing with Complex Wavelets," IEEE Transactions on Image Processing, vol. 12, no. 12, pp. 1415-1425, December 2003. DOI: 10.1109/TIP.2003.819229.
16. H. Wang, X. Chen, and Q. Ding, "Single Level Decomposition-Based Time-Frequency Feature Extraction for Gear Fault Diagnosis," IEEE Transactions on Industrial Electronics, vol. 64, no. 5, pp. 3899-3909, May 2017. DOI: 10.1109/TIE.2016.2642639.
17. J. Tian, Z. Zhong, and G. Li, "A Multi-Level Decomposition Method for Image Fusion," IEEE Transactions on Geoscience and Remote Sensing, vol. 55, no. 2, pp. 805-819, February 2017. DOI: 10.1109/TGRS.2016.2614932.
18. D. L. Donoho, "De-Noising by Soft-Thresholding," IEEE Transactions on Information Theory, vol. 41, no. 3, pp. 613-627, May 1995. DOI: 10.1109/18.382009.
19. C. S. Burrus, R. A. Gopinath, and H. Guo, "Introduction to Wavelets and Wavelet Transforms: A Primer," Proceedings of the IEEE, vol. 84, no. 4, pp. 518-533, April 1996. DOI: 10.1109/5.488544.
20. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, May 2015. DOI: 10.1038/nature14539.

21. K.Zhang,W.Zuo,andL.Zhang.FFDNet:Toward a fast and flexible solution or CNN based image denoising.IEEE Transactionson Image Processing,2018.
22. A.S.LewisandG.Knowles. Image compression using the 2-D wavelet transform.IEEETransactionsonImageProcessing,1(2):244–250,1992.
23. O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Munich, Germany, October 2015. DOI: 10.1007/978-3-319-24574-4\_28.
24. R. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, pp. 770-778. DOI: 10.1109/CVPR.2016.90.
25. D. Kingma and J. Ba. Adam: A method for stochastic optimization. In International Conference for Learning Representations, 2015.
26. K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian denoiser: Residual learning of deep cnn for image denoising. IEEE Transactions on Image Processing, PP (99):1–1, 2016.



## APPENDIX A

### Data preparations for CWCNN training and testing

```
import spectral as sp
import os
from google.colab import drive
drive.mount('/content/drive')

hdr =
sp.envi.open("/content/drive/MyDrive/dataset/Urban_F210.hdr
")
rows, cols, bands = hdr.nrows, hdr.ncols, hdr.nbands
nr, ng, nb = [int(x) for x in hdr.metadata['default
bands']]

rgb = hdr.read_bands([nr, ng, nb])

sp.imshow(rgb)

import cv2
import matplotlib.pyplot as plt
import numpy as np

hist1 = cv2.calcHist([rgb],[0],None,[256],[0,256])
plt.subplot(221),plt.imshow(rgb);
plt.subplot(222),plt.plot(hist1);

img_array = rgb

"""
STEP 1: Normalized cumulative histogram
"""
#flatten image array and calculate histogram via binning
histogram_array = np.bincount(img_array.flatten(),
minlength=256)
#normalize
num_pixels = np.sum(histogram_array)
histogram_array = histogram_array/num_pixels
#cumulative histogram
chistogram_array = np.cumsum(histogram_array)
"""
STEP 2: Pixel mapping lookup table
"""
transform_map = np.floor(255 *
chistogram_array).astype(np.uint16)
```

```

"""
STEP 3: Transformation
"""
# flatten image array into 1D list
img_list = list(img_array.flatten())
# transform pixel values to equalize
eq_img_list = [transform_map[p] for p in img_list]
# reshape and write back into img_array
eq_img_array = np.reshape(np.asarray(eq_img_list),
img_array.shape)
hist2 = cv2.calcHist([eq_img_array],[0],None,[256],[0,256])

plt.subplot(223),plt.imshow(eq_img_array);
plt.subplot(224),plt.plot(hist2);

# sp.save_rgb('/content/drive/My Drive/Colab
Notebooks/rgb1_heq.tif',eq_img_array)
sp.imshow(eq_img_array, stretch=(0.02, 0.98))
plt.subplot(121),plt.imshow(rgb), plt.axis('off');
plt.subplot(122),plt.imshow(eq_img_array), plt.axis('off');
sp.save_rgb('/content/drive/My Drive/rgb_urban.tif',rgb)

"""
Data Preparation
"""
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import os
import pathlib
from matplotlib import pyplot as plt
import seaborn as sns
import pickle
import pandas as pd
import numpy as np
import cv2
import skimage
from skimage.util import random_noise
import tensorflow as tf

from tqdm.notebook import tqdm
import random

def add_noise(img):

```

```

# Getting the dimensions of the image
row , col = img.shape

# Randomly pick some pixels in the
# image for coloring them white
# Pick a random number between 300 and 10000
number_of_pixels = random.randint(300, 10000)
for i in range(number_of_pixels):

    # Pick a random y coordinate
    y_coord=random.randint(0, row - 1)

    # Pick a random x coordinate
    x_coord=random.randint(0, col - 1)

    # Color that pixel to white
    img[y_coord][x_coord] = 255

# Randomly pick some pixels in
# the image for coloring them black
# Pick a random number between 300 and 10000
number_of_pixels = random.randint(300 , 10000)
for i in range(number_of_pixels):

    # Pick a random y coordinate
    y_coord=random.randint(0, row - 1)

    # Pick a random x coordinate
    x_coord=random.randint(0, col - 1)

    # Color that pixel to black
    img[y_coord][x_coord] = 0

return img

# salt-and-pepper noise can
# be applied only to grayscale images
# Reading the color image in grayscale image
img =
cv2.imread('/content/drive/MyDrive/dataset/rgb_urban.tif',
           cv2.IMREAD_GRAYSCALE)

#Storing the image
cv2.imwrite('/content/drive/MyDrive/dataset/NOISYrgb_urban.
.tif',
           add_noise(img))

if 'rgb_urban' not in os.listdir():

```

```

!gdown
https://drive.google.com/uc?id=104UHeL1RKVb6fKaPZUIDGx0tulg
5cRCU

def get_images_paths(root_dir_ssid, root_dir_mi,
root_dir_nind):

    # Getting Urban dataset (noised and clear) images
    root = pathlib.Path(root_dir_ssid)
    img_paths = list(root.rglob("*.tif*"))
    img_paths_lst = [str(path) for path in img_paths]

    gt_lst = []
    noisy_lst= []
    for p in img_paths_lst:
        img_type = p.split("/")[-1].split('_')[-3]
        if img_type=="NOISY":
            noisy_lst.append(p)
        else
            gt_lst.append(p)

noisy_array = np.asarray(noisy_lst)
gt_array = np.asarray(gt_lst)
return noisy_array, gt_array

from sklearn.model_selection import train_test_split

noisy_array_paths, gt_array_paths =
get_images_paths("rgb_urban")

noisy_train_paths, noisy_test_paths, gt_train_paths,
gt_test_paths = train_test_split(noisy_array_paths,
gt_array_paths, test_size=0.20, random_state=42)

# Get gt_images in memory
def get_images_in_mem(images_paths):
    images_lst = []
    for img_path in tqdm(images_paths):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (256, 256))
        images_lst.append(img)
    return np.array(images_lst)

noisy_train_images = get_images_in_mem(noisy_train_paths)
noisy_test_images = get_images_in_mem(noisy_test_paths)

gt_train_images = get_images_in_mem(gt_train_paths)

```

```

gt_test_images = get_images_in_mem(gt_test_paths)

print(noisy_train_images.shape)
print(noisy_test_images.shape)

print(gt_train_images.shape)
print(gt_test_images.shape)

f, axarr = plt.subplots(1,2, figsize=(14,14))
axarr[0].imshow(noisy_train_images[5])
axarr[0].set_title("Noisy image")
axarr[1].imshow(gt_train_images[5])
axarr[1].title.set_text("Ground Truth image")

# Augmentation techniques
def _up_down_flip(image, label):
    image = tf.image.flip_up_down(image)
    label = tf.image.flip_up_down(label)
    return image, label

def _left_right_flip(image, label):
    image = tf.image.flip_left_right(image)
    label = tf.image.flip_left_right(label)
    return image, label

def _rotate(image, label):
    random_angle = tf.random.uniform(shape=[], minval=0,
maxval=4, dtype=tf.int32)
    image = tf.image.rot90(image, random_angle)
    label = tf.image.rot90(label, random_angle)
    return image, label

def _hue(image, label):
    rand_value = random.uniform(-1,1)
    image = tf.image.adjust_hue(image, rand_value)
    label = tf.image.adjust_hue(label, rand_value)
    return image, label

def _brightness(image, label):
    rand_value = random.uniform(-0.08,0.25)
    image = tf.image.adjust_brightness(image, rand_value)
    label = tf.image.adjust_brightness(label, rand_value)
    return image, label

def _saturation(image, label):
    rand_value = random.uniform(1, 5)
    image = tf.image.adjust_saturation(image, rand_value)
    label = tf.image.adjust_saturation(label, rand_value)

```

```

    return image, label

def _contrast(image, label):
    rand_value = random.uniform(1, 3)
    image = tf.image.adjust_contrast(image, rand_value)
    label = tf.image.adjust_contrast(label, rand_value)
    return image, label

def tf_data_generator(X, y, batch_size=32,
augmentations=None):
    dataset = tf.data.Dataset.from_tensor_slices((X, y))

# This is the main step for data generation
    dataset = dataset.shuffle(1000,
reshuffle_each_iteration=True)

    if augmentations:
        for f in augmentations:
            if np.random.uniform(0,1)<0.5:
                dataset = dataset.map(f,
num_parallel_calls=2)

    dataset = dataset.repeat()
    dataset = dataset.batch(batch_size=batch_size,
drop_remainder=True)
    dataset =
dataset.prefetch(tf.data.experimental.AUTOTUNE)
    return dataset

BATCH_SIZE=4
augmentation_lst = [_up_down_flip, _left_right_flip,
_rotate]
image_generator_train =
tf_data_generator(X=noisy_train_images, y=gt_train_images,
batch_size=BATCH_SIZE, augmentations=augmentation_lst)
image_generator_test =
tf_data_generator(X=noisy_test_images, y=gt_test_images,
batch_size=BATCH_SIZE)

image_generator_train

```

## APPENDIX B

### Software realization of using CWCNN model for image noise filtering

```
import tensorflow as tf
from tensorflow.keras import models, layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D,
Conv2DTranspose, \
                                GlobalAveragePooling2D,
AveragePooling2D, MaxPool2D, UpSampling2D, \
                                BatchNormalization,
Activation, ReLU, Flatten, Dense, Input, \
                                Add, Multiply,
Concatenate, Softmax
from tensorflow.keras import initializers, regularizers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.activations import softmax

tf.keras.backend.set_image_data_format('channels_last')
import keras.backend as K

class Conv_block(tf.keras.layers.Layer):
    def __init__(self, num_filters=200, kernel_size=3,
**kwargs):
        super().__init__(**kwargs)
        self.num_filters=num_filters
        self.kernel_size=kernel_size
        self.conv_1 = Conv2D(filters=self.num_filters,
kernel_size=self.kernel_size, padding='same')
        self.conv_2 = Conv2D(filters=self.num_filters,
kernel_size=self.kernel_size, padding='same')
        self.conv_3 = Conv2D(filters=self.num_filters,
kernel_size=self.kernel_size, padding='same')
        self.conv_4 = Conv2D(filters=self.num_filters,
kernel_size=self.kernel_size, padding='same')

        self.bn_1 = BatchNormalization()
        self.bn_2 = BatchNormalization()
        self.bn_3 = BatchNormalization()
        self.bn_4 = BatchNormalization()

    def get_config(self):
        config = super().get_config().copy()
        config.update({
            'num_filters': self.num_filters,
            'kernel_size':self.kernel_size
        })
        return config
```

```

def call(self, X):
    X = self.conv_1(X)
    # X = self.bn_1(X)
    X = ReLU()(X)
    X = self.conv_2(X)
    # X = self.bn_2(X)
    X = ReLU()(X)
    X = self.conv_3(X)
    # X = self.bn_3(X)
    X = ReLU()(X)
    # X = self.conv_4(X)
    # # X = self.bn_4(X)
    # X = ReLU()(X)

    return X

class DWT_downsampling(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, x):

        x1 = x[:, 0::2, 0::2, :] #x(2i-1, 2j-1)
        x2 = x[:, 1::2, 0::2, :] #x(2i, 2j-1)
        x3 = x[:, 0::2, 1::2, :] #x(2i-1, 2j)
        x4 = x[:, 1::2, 1::2, :] #x(2i, 2j)

        x_LL = x1 + x2 + x3 + x4
        x_LH = -x1 - x3 + x2 + x4
        x_HL = -x1 + x3 - x2 + x4
        x_HH = x1 - x3 - x2 + x4

        return Concatenate(axis=-1)([x_LL, x_LH, x_HL,
x_HH])

class IWT_upsampling(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, x):
        x_LL = x[:, :, :, 0:x.shape[3]//4]
        x_LH = x[:, :, :, x.shape[3]//4:x.shape[3]//4*2]
        x_HL = x[:, :, :, x.shape[3]//4*2:x.shape[3]//4*3]
        x_HH = x[:, :, :, x.shape[3]//4*3:]

        x1 = (x_LL - x_LH - x_HL + x_HH)/4
        x2 = (x_LL - x_LH + x_HL - x_HH)/4

```



```

x3 = (x_LL + x_LH - x_HL - x_HH)/4
x4 = (x_LL + x_LH + x_HL + x_HH)/4

y1 = K.stack([x1,x3], axis=2)
y2 = K.stack([x2,x4], axis=2)
shape = K.shape(x)
return K.reshape(K.concatenate([y1,y2], axis=-1),
K.stack([shape[0], shape[1]*2, shape[2]*2, shape[3]//4]))
# Create Model
def create_model():
    tf.keras.backend.clear_session()

    input = Input(shape=(256,256,3))

    cb_1 = Conv_block(num_filters=64)(input)
    dwt_1 = DWT_downsampling()(cb_1)

    cb_2 = Conv_block(num_filters=128)(dwt_1)
    dwt_2 = DWT_downsampling()(cb_2)

    cb_3 = Conv_block(num_filters=256)(dwt_2)
    dwt_3 = DWT_downsampling()(cb_3)

    cb_4 = Conv_block(num_filters=512)(dwt_3)
    dwt_4 = DWT_downsampling()(cb_4)

    cb_5 = Conv_block(num_filters=512)(dwt_4)
    cb_5 = BatchNormalization()(cb_5)
    cb_5 = Conv_block(num_filters=512)(cb_5)
    cb_5 = Conv2D(filters=2048, kernel_size=3, strides=1,
padding='same')(cb_5)

    up = IWT_upsampling()(cb_5)
    up = Conv_block(num_filters=512)(Add()([up, cb_4]))
    up = Conv2D(filters=1024, kernel_size=3, strides=1,
padding='same')(up)

    up = IWT_upsampling()(up)
    up = Conv_block(num_filters=256)(Add()([up, cb_3]))
    up = Conv2D(filters=512, kernel_size=3, strides=1,
padding='same')(up)

    up = IWT_upsampling()(up)
    up = Conv_block(num_filters=128)(Add()([up, cb_2]))
    up = Conv2D(filters=256, kernel_size=3, strides=1,
padding='same')(up)

    up = IWT_upsampling()(up)

```

```

    up = Conv_block(num_filters=64)(Add()([up, cb_1]))
    up = Conv2D(filters=128, kernel_size=3, strides=1,
padding='same')(up)

    out = Conv2D(filters=3, kernel_size=(1, 1),
padding="same")(up)

    return Model(inputs=[input], outputs=[out])

model = create_model()
model.summary()

# Training

steps_per_epoch_train = len(noisy_train_images)
steps_per_epoch_validation = len(noisy_test_images)

callbacks_lst = [
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
, min_lr=0.0000009, min_delta=0.0001, factor=0.70,
patience=3, verbose=1, mode='min'),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
mode='min', verbose=1, min_delta=0.0001, patience=10)
]

model.compile(loss=tf.keras.losses.MeanSquaredError(),
optimizer=Adam(learning_rate=0.0009))
model.fit(image_generator_train,
validation_data=image_generator_test,
steps_per_epoch=steps_per_epoch_train,
validation_steps=steps_per_epoch_validation,
epochs=100,
verbose=1,
callbacks=callbacks_lst)
history = model.fit(x= image_generator_train, y=
image_generator_test, batch_size=256, epochs=100,
validation_split=0.2)

# Inference
def inference_single_image(model, noisy_image):
    input_image = np.expand_dims(noisy_image, axis=0)
    predicted_image = model.predict(input_image)

    return predicted_image[0]

def inference_batch_images(model, noisy_images):

```

```

    predicted_image = model.predict(noisy_images)
    return predicted_image

def visualize_predictions(model, X_test, y_test, n):
    random_numbers = random.choices(range(X_test.shape[0]),
k=n)    # Get n random indices
    for i in random_numbers:
        noisy_image = X_test[i]
        gt_image = y_test[i]
        predicted_image = inference_single_image(model,
X_test[i])
        predicted_image/=255

        f, axarr = plt.subplots(1,3, figsize=(21,21))
        axarr[0].imshow(noisy_image)
        axarr[0].set_title("Noisy image")
        axarr[0].set_axis_off()
        axarr[1].imshow(gt_image)
        axarr[1].set_title("Ground truth image")
        axarr[1].set_axis_off()
        axarr[2].imshow(predicted_image)
        axarr[2].set_title("Predicted image")
        axarr[2].set_axis_off()

visualize_predictions(model, noisy_test_images,
gt_test_images, 10)

# Test image prediction
urban = "No"

img_path = "/content/drive/MyDrive/Colab
Notebooks/data/CWCNN/"+urban

img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (256, 256))

predicted_image = inference_single_image(model, img)
predicted_image/=255

f, axarr = plt.subplots(1,2, figsize=(14,14))
axarr[0].imshow(img)
axarr[0].title.set_text("Noisy image")
axarr[0].set_axis_off()

axarr[1].imshow(predicted_image)
axarr[1].title.set_text("Predicted image")
axarr[1].set_axis_off()

```

```

#get PSNR data
from skimage.metrics import peak_signal_noise_ratio

predicted_images = inference_batch_images(model,
noisy_test_images)
psnr_original_mean = 0
psnr_prediction_mean = 0

for gt_img, noisy_img, predicted_img in zip(gt_test_images,
noisy_test_images, predicted_images):
    psnr_original_mean += peak_signal_noise_ratio(gt_img,
noisy_img)
    psnr_prediction_mean += peak_signal_noise_ratio(gt_img,
predicted_img)

psnr_original_mean/=gt_test_images.shape[0]
psnr_prediction_mean/=gt_test_images.shape[0]
print("Original average gt-noisy PSNR ->",
psnr_original_mean)
print("Predicted average gt-predicted PSNR ->",
psnr_prediction_mean)

#get data SSIM
from skimage.metrics import structural_similarity as ssim

predicted_images = inference_batch_images(model,
noisy_test_images)
ssim_original_mean = 0
ssim_prediction_mean = 0

for gt_img, noisy_img, predicted_img in zip(gt_test_images,
noisy_test_images, predicted_images):
    ssim_original_mean += ssim(gt_img, noisy_img,
multichannel=True, data_range=noisy_img.max() -
noisy_img.min())
    ssim_prediction_mean += ssim(gt_img, predicted_img,
multichannel=True, data_range=predicted_img.max() -
predicted_img.min())

ssim_original_mean/=gt_test_images.shape[0]
ssim_prediction_mean/=gt_test_images.shape[0]
print("Original average gt-noisy SSIM ->",
ssim_original_mean)
print("Predicted average gt-predicted SSIM ->",
ssim_prediction_mean)

```

## APPENDIX C

Noise type	$\sigma$	Original X-y pairs	MF	DnCNN	CWCNN
Gaussian	5	28.4561/0.7344	30.2114/0.7967	32.4890/0.8394	35.1901/0.9210
	15	26.3779/0.6000	28.6278/0.7221	30.5713/0.7932	32.5220/0.8397
	30	24.5390/0.5513	25.7463/0.6782	28.7051/0.6520	30.3067/0.7645
Salt and Paper	5	27.6784/0.6858	31.1149/0.8183	30.7839/0.7856	33.2451/0.7998
	15	25.2800/0.6055	29.4108/0.7064	28.5467/0.6440	31.4398/0.7459
	30	23.4581/0.5367	27.6743/0.6267	27.2908/0.6290	29.8974/0.7143
Combine	5	25.4360/0.6148	29.3401/0.6873	30.5789/0.7231	32.3902/0.7640
	15	24.3781/0.5750	27.3902/0.6173	29.4306/0.6947	31.1042/0.7212
	30	22.5304/0.4864	26.2105/0.5847	27.9860/0.6703	29.5490/0.6991

Table 4.1. Filtration performance improvements by chosen model