# Deep Learning-Based SAR image despeckling

*A Report*

*Submitted in partial fulfilment of the*
*Requirements for the completion of*

*THEME BASED PROJECT*

**BACHELOR OF ENGINEERING**
IN
**INFORMATION TECHNOLOGY**

By


**G. Mrunal 1602-21-737-034**
**G. Rishi 1602-21-737-042**
**P. Munna 1602-21-737-301**


**Under the guidance of**
**Ms.Soume Sanyal**
**Assistant Professor**


**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**ACCREDITED BY NAAC WITH 'A++' GRADE.**

**(Affiliated to Osmania University and Approved by AICTE)**

**Ibrahim Bagh, Hyderabad-31**


**2024**

# Vasavi College of Engineering (Autonomous)

*ACCREDITED BY NAAC WITH 'A++' GRADE*

**(Affiliated to Osmania University and Approved by AICTE)**

**Ibrahim Bagh, Hyderabad-31**

## Department of Information Technology

## DECLARATION BY CANDIDATES

We, **G.Mrunal, G.Rishi, P.Munna,** bearing hall ticket number, **1602-21-737-034, 1602-21-737-042, 1602-21-737-301,** hereby declare that the project report entitled **"Deep learning-based SAR image despeckling"** under the guidance of Ms.Soume Sanayal, Assistant Professor, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfillment of the requirement for the completion of Theme-based project , VI semester, Bachelor of Engineering in Information Technology.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other institutes.

G.Mrunal 1602-21-737-034
G.Rishi 1602-31-737-042
P.Munna 1602-21-737-301

# Vasavi College of Engineering (Autonomous)

*ACCREDITED BY NAAC WITH 'A++' GRADE*

**(Affiliated to Osmania University and Approved by AICTE)**

**Ibrahim Bagh, Hyderabad-31**

## Department of Information Technology

## BONAFIDE CERTIFICATE

This is to certify that the project entitled "**Deep learning-based SAR image despeckling**" being submitted by **G.Mrunal,G.Rishi,P.Munna** bearing **1602-21-737-034, 1602-21-737-042, 1602-21-737-301,** in partial fulfillment of the requirements for the completion of Theme-based project of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

**Ms.Soume Sanayal**
Assistant Professor

**External Examiner**

**Dr. K. Ram Mohan Rao**
**Professor, HOD IT**

## ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the project would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to **Ms.Soume Sanayal, Assistant professor, Information Technology** under whom we executed this project· Her constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. Ram Mohan Rao, Professor and HOD, Information Technology,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to Dr. S. V. Ramana**, Principal** of **Vasavi College of Engineering** for giving the required information in doing my project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time.

We also express our sincere thanks to the Management for providing excellent facilities. Finally, we wish to convey our gratitude to our family who fostered all the facilities that we need.

# ABSTRACT

This project explores the application of deep learning techniques for image denoising, aiming to develop an effective convolutional neural network (CNN) model capable of removing noise from images while preserving important features. The methodology involves meticulous data preparation, thoughtful model architecture design, formulation of an effective loss function, and iterative training loop implementation. The model is trained using a dataset of clean and noisy image pairs, with the training process optimized through hyperparameter tuning and experimentation. Results demonstrate promising denoising performance, both quantitatively through metrics such as mean squared error (MSE) and qualitatively through visual inspection. The project also discusses potential avenues for future research and development in the field of image denoising..

## LIST OF FIGURES


| Fig no | Figure Name | Page No |
|---|---|---|
| Fig 4.1.1.1 | Architecture diagram | 6 |
| Fig 4.1.1.2 | Use Case diagram | 6 |
| Fig 4.2.1.1 | CNN Model Code | 9 |
| Fig 4.2.1.2 | Testing module code | 10 |
| Fig 4.2.1.3 | Loss function module code | 11 |
| Fig 4.2.1.4 | Training function module code | 11 |
| Fig 6.1 | Result 1 | 18 |
| Fig 6.2 | Result 2 | 18 |
| Fig 6.3 | Result 3 | 19 |
| Fig 6.4 | Result 4 | 19 |

# TABLE OF CONTENTS


**1. INTRODUCTION**

1.1 Overview 1

1.2 Problem Statement 2

1.3 Motivation of theme & title 2

**2. LITERATURE SURVEY** 3

**3. EXISTING SYSTEM** 4

**4. PROPOSED SOLUTION** 5

4.1. System Design 6

4.1.1 Architecture Diagram 6

4.1.2 Use-Case Diagram 6

4.1.2.1 Use-case descriptions 7

4.2 Functional Modules 9

4.2.1 Screenshots & Pseudocode 9

**5. EXPERIMENTAL SETUP & IMPLEMENTATION** 16

5.1 System Specifications 16

5.1.1 Hardware Requirements 16

5.1.2 Software Requirements 16

5.2 Datasets 16

5.3 Methodology 17

**6. RESULTS** 18

**7. CONCLUSION & FUTURE SCOPE** 20

**8. REFERENCES** 21

# 1.Introduction

## 1.1 Overview

Synthetic Aperture Radar (SAR) imagery is essential for applications such as environmental monitoring, disaster management, and military surveillance. However, its effectiveness is significantly impeded by speckle noise, an inherent artifact that obscures image details. Traditional despeckling methods, including filtering techniques and wavelet-based approaches, often fall short by either blurring important details, introducing artifacts, or failing to fully suppress the noise. These limitations make accurate interpretation and decision-making challenging. Therefore, there is an urgent need for a robust and efficient solution that can automatically remove speckle noise while preserving crucial image features, thus enhancing the quality and utility of SAR imagery and enabling more accurate analysis across various domains.

This project aims to develop an advanced SAR image despeckling technique using Recurrent Convolutional Neural Networks (RCNN). By leveraging the power of deep learning, particularly the RCNN's capability to model spatial dependencies and learn complex patterns, this approach promises to surpass traditional methods in performance. The RCNN-based method can effectively suppress speckle noise while preserving fine image details, resulting in clearer and more interpretable SAR images. This innovative solution not only improves the visual quality of SAR imagery but also enhances its practical applicability in critical areas such as environmental monitoring, disaster management, and military surveillance, ultimately leading to more informed and accurate decision-making.

## 1.2 PROBLEM STATEMENT

The utility of Synthetic Aperture Radar (SAR) imagery in crucial applications such as environmental monitoring, disaster management, and military surveillance is significantly compromised by speckle noise, an inherent artifact that obscures image details. Traditional despeckling methods, including filtering techniques and wavelet-based approaches, often fail to effectively address this issue, either by blurring essential details, introducing artifacts, or inadequately suppressing the noise. This challenge necessitates the development of a simple yet robust solution that can automatically remove speckle noise while preserving critical image features. The proposed project aims to develop a despeckling algorithm using Recurrent Convolutional Neural Networks (RCNN), leveraging deep learning to enhance the quality and interpretability of SAR imagery, thereby enabling more accurate and reliable analysis for various applications.

## 1.3 Motivation of Theme and Title

- The motivation for our project arises from the need to improve the quality and utility of Synthetic Aperture Radar (SAR) imagery. Traditional despeckling methods often fail to adequately address speckle noise, leading to blurred details, introduced artifacts, or insufficient noise suppression. By leveraging the advanced capabilities of Recurrent Convolutional Neural Networks (RCNN), we aim to develop a more effective and reliable despeckling solution

- Our goal is to enhance the interpretability and accuracy of SAR imagery analysis, thus supporting more informed decision-making in various critical applications. By providing clearer images, we aim to facilitate more precise data interpretation, improving outcomes in research, operational planning, and strategic development.

# 2. LITERATURE SURVEY

- Lee, J. S. (1981). "Digital image enhancement and noise filtering by use of local statistics." IEEE Transactions on Pattern Analysis and Machine Intelligence, 2(2), 165-168.

Lee's speckle filtering method is foundational in SAR image processing, using local statistics to estimate and suppress speckle noise.

- Frost, V. S., Stiles, J. A., Shanmugan, K. S., & Holtzman, J. C. (1982). "A model for radar images and its application to adaptive digital filtering of multiplicative noise." IEEE Transactions on Pattern Analysis and Machine Intelligence, 4(2), 157-166.

This paper introduces an adaptive filtering approach for SAR images based on a multiplicative noise model, which has influenced subsequent despeckling methods.

- Argenti, F., & Alparone, L. (1998). "Multiresolution MAP despeckling of SAR images based on a decoupled Bayesian estimator." IEEE Transactions on Geoscience and Remote Sensing, 36(2), 543-557.

Presents a Bayesian approach for SAR image despeckling that operates in a multiresolution framework, effectively balancing between speckle reduction and preservation of image details.

- Vasile, G., & Bruzzone, L. (2005). "A contextual-orientation adaptive despeckling filter for SAR images." IEEE Transactions on Geoscience and Remote Sensing, 43(4), 747-761.

Proposes an adaptive filter that considers both contextual and orientation information for speckle reduction in SAR images, improving the preservation of edges and texture.

- Buades, A., Coll, B., & Morel, J. M. (2005). "A review of image denoising algorithms, with a new one." Multiscale Modeling & Simulation, 4(2), 490-530.

Although not specific to SAR, this review covers key principles and methodologies in image denoising, offering insights applicable to SAR image despeckling techniques.

- Achim, A., Tsakalides, P., & Bezerianos, A. (2006). "A novel Bayesian approach to adaptive spatially varying speckle filtering in SAR images." IEEE Transactions on Image Processing, 15(5), 1181-1193.

Introduces a Bayesian framework for adaptive spatially varying speckle filtering in SAR images, emphasizing robustness and adaptability to different types of scenes.

# 3.EXISISTING SYSTEM

- Traditional methods for speckle reduction in SAR imagery encompass several approaches, each with its strengths and limitations. Multi-look processing, a common technique, involves averaging multiple SAR images to reduce noise. While effective to some extent, this method may result in loss of spatial resolution. Filtering methods, such as Lee and Frost filters, aim to suppress speckle while preserving image details. However, these approaches often struggle to strike a balance between noise reduction and edge preservation.

- Wavelet-based despeckling methods exploit the multi-resolution nature of wavelet transforms to remove noise at different scales. Although these methods can effectively reduce speckle, they may introduce artifacts and result in loss of fine image features. Total Variation (TV) methods, on the other hand, utilize mathematical optimization techniques to enforce image smoothness while preserving edges. While successful in some cases, TV methods may oversmooth the image, leading to loss of important details.

- Nonlocal filtering techniques, such as NL-means and BM3D, have gained popularity for their ability to exploit similarities between image patches for noise reduction. However, these methods may produce despeckled images with blurry edges and residual artifacts, limiting their applicability in critical domains.

- In summary, while traditional methods offer various options for speckle reduction in SAR imagery, they often face challenges in achieving optimal noise suppression while preserving important image features. There is a clear need for advanced despeckling techniques that can effectively address these limitations and provide clearer, more interpretable SAR images for improved analysis and decision-making.

# 4. PROPOSED SOLUTION

Our proposed solution seeks to enhance Synthetic Aperture Radar (SAR) image despeckling using an advanced deep learning architecture. Inspired by recent advancements in convolutional neural networks (CNNs), particularly Recurrent Convolutional Neural Networks (RCNNs), we aim to develop a novel despeckling model tailored specifically for SAR imagery.

The model architecture is designed to effectively capture both local and global features of SAR images, leveraging the inherent spatial dependencies present in the data. The proposed model consists of multiple convolutional layers with varying dilation rates, enabling it to effectively suppress speckle noise while preserving fine image details.

The key components of our proposed model include:

- **Input Layer**: The model accepts SAR images as input, with dimensions specified by the user.
- **Convolutional Layers**: A series of convolutional layers with increasing dilation rates are applied to the input data. These layers are responsible for capturing local and global features of the SAR imagery.
- **Batch Normalization**: Batch normalization layers are incorporated to improve the stability and speed of convergence during training.
- **Activation Functions**: Leaky ReLU and ReLU activation functions are used to introduce non-linearity and enhance feature representation.
- **Output Layer**: The final convolutional layer produces the despeckled SAR image, which is then normalized and adjusted for improved visual quality.

The proposed solution aims to overcome the limitations of traditional despeckling methods by leveraging the power of deep learning. By effectively suppressing speckle noise while preserving important image features, our model promises to deliver clearer and more interpretable SAR images, facilitating more accurate analysis and decision-making in critical applications.

## 4.1 SYSTEM DESIGN

### 4.1.1 ARCHITECTURE DIAGRAM

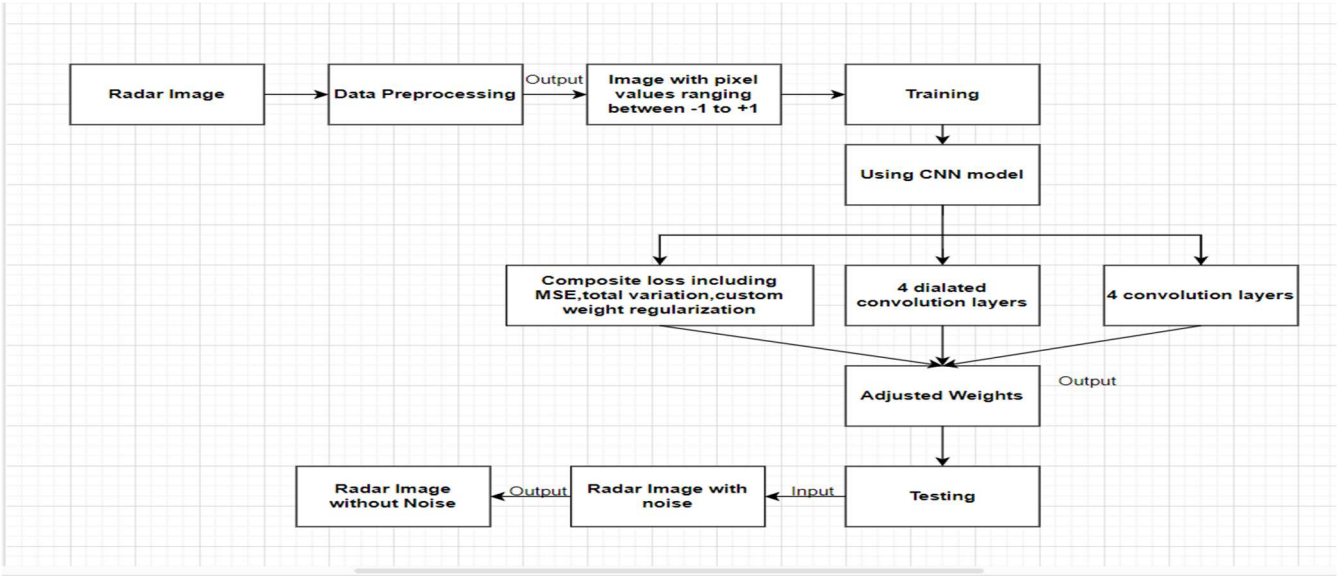


Fig 4.1.1.1 Architecture diagram

### 4.1.2 USE-CASE DIAGRAM

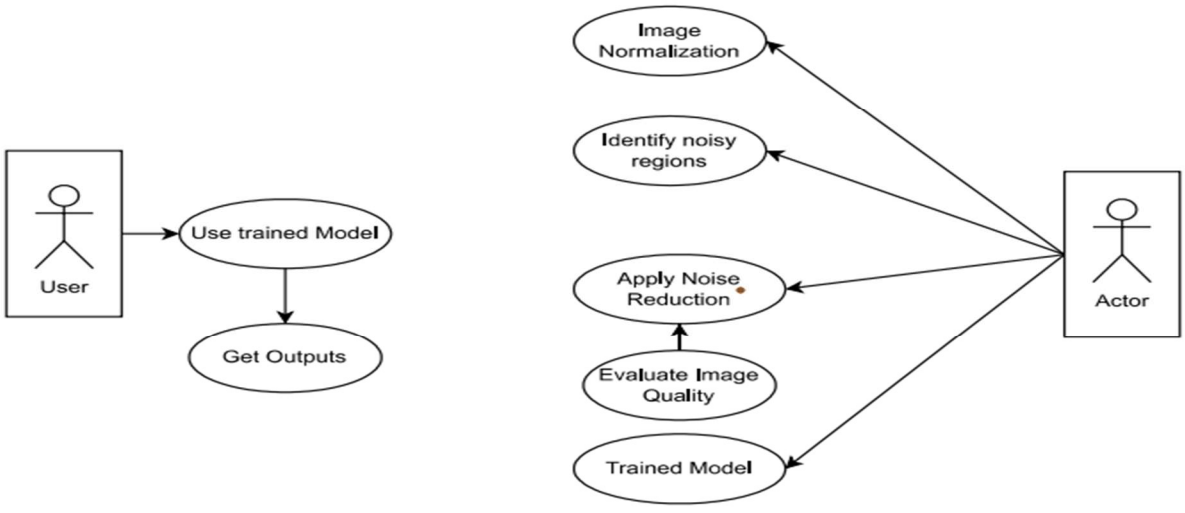


Fig 4.1.1.2 Use Case diagram

### 4.1.2 USE-CASE DESCRIPTIONS

**Use Case ID** : UC-1

**Use Case Name** : Preprocess Image Data using RNN

**Actors**:

**System:** Performs image preprocessing tasks using a Recurrent Neural Network (RNN).

**Description:** This use case focuses on preprocessing image data, particularly for speckle noise removal, using a Recurrent Neural Network (RNN) approach.

**Preconditions:** Input image data with speckle noise is available and accessible for preprocessing

**Flow of Events:**

The system retrieves an input image containing speckle noise.

The RNN model processes the image data sequentially, leveraging its recurrent nature to capture temporal dependencies.

The RNN model identifies noisy regions or patterns within the image based on sequential data processing.

Speckle noise removal techniques are applied iteratively across the image using the RNN's sequential processing capabilities.

**Postconditions**: A preprocessed version of the input image, with reduced speckle noise, is generated and ready for further analysis or applications.

**Use Case ID**: UC-2

**Use case name**: Enhance Image Quality using RNN-based Denoising

**Actors:**

**System:** Utilizes RNN-based denoising techniques for image quality enhancement.

**Description:** This use case involves using a Recurrent Neural Network (RNN) to enhance image quality by reducing speckle noise.

**Preconditions:** Image data with speckle noise is available and identified for quality improvement.

**Flow of Events:**

The system feeds the input image data into an RNN-based denoising model.

The RNN model processes the image data iteratively, leveraging recurrent connections to capture noise patterns and spatial dependencies.

Speckle noise is progressively reduced through sequential processing and learned representations within the RNN.

The denoised image is outputted as the result of the RNN-based enhancement process.

**Postconditions:** A visually improved and denoised version of the input image is produced, suitable for analysis or visualization tasks**.**

**Use Case ID**: UC-3
**Use Case Name:** Validate Denoising Effectiveness with RNN-based Metrics
**Actors:**
**System**: Evaluates denoising effectiveness using RNN-based quality metrics.
**Description**:
This use case involves assessing the effectiveness of speckle noise reduction using a Recurrent Neural Network (RNN) and associated quality metrics.
**Preconditions:** Denoised images generated by the RNN-based preprocessing are available for evaluation.
**Flow of Events:**
The system computes quantitative metrics (e.g., PSNR, SSIM) using the denoised images as ground truth.
RNN-based features and representations are leveraged to calculate image similarity and quality improvement metrics.
The effectiveness of the RNN-based denoising approach is evaluated based on the computed metrics.
**Postconditions:** Performance metrics indicating the quality enhancement achieved by the RNN-based denoising method are generated and utilized for validation purposes.

## 4.2 FUNCTIONAL MODULES

### 4.2.1 SCREENSHOTS AND PSEUDOCODE

**MODEL**

```
def create_model(input_shape=(256,256,1)):
    input_layer=Input(shape=input_shape)
    x=Conv2D(filters=64,kernel_size=(3,3),padding='same')(input_layer)
    x=LeakyReLU(.2)(x)

    for i in range(1,5):
        x=Conv2D(filters=64,kernel_size=(3,3),dilation_rate=i,padding='same',)(x)
        x=BatchNormalization()(x)
        x=LeakyReLU(.2)(x)
    for i in range(4,0,-1):
        x=Conv2D(filters=64,kernel_size=(3,3),dilation_rate=i,padding='same')(x)
        x=BatchNormalization()(x)
        x=ReLU()(x)
    x=Conv2D(filters=1,kernel_size=(3,3),padding='same')(x)
    x=ReLU()(x)
    x= tf.keras.layers.Lambda(lambda x:x+tf.constant(1e-7))(x)
    x=tf.math.divide(input_layer,x)

    x=tf.math.tanh(x)
    return tf.keras.Model(inputs=input_layer,outputs=x)
```

Fig 4.2.1.1 CNN Model Code

- The create_model function defines a convolutional neural network architecture tailored for image processing tasks on grayscale images of size 256x256 pixels. It begins with an initial convolutional layer followed by several blocks of dilated convolutions. Each block alternates between applying dilated convolutions with increasing and then decreasing rates, promoting a wide receptive field while maintaining spatial resolution. Batch normalization and Leaky ReLU activations are employed throughout to stabilize and enhance training.

- After the dilated convolutional blocks, reverse dilated convolutions further refine features, utilizing decreasing dilation rates to consolidate information across different scales. The final layers consist of a 3x3 convolutional operation to produce a single-channel output, followed by ReLU activation. Notably, the model incorporates normalization and mathematical operations to ensure stable outputs and effective utilization of network capabilities.

- Overall, this architecture aims to leverage dilated convolutions for expansive receptive fields, facilitating nuanced feature extraction in image data, which is essential for tasks requiring detailed spatial information processing.

**TESTING**

```
def test_model(data_generator):
    img1,img2=next(data_generator)[:2]
    noise_var=np.random.rand()*.25
    # noise_var=.3
    noisy_img1=random_noise(img1,mode='speckle',var=noise_var,clip=True)
    noisy_img2=random_noise(img2,mode='speckle',var=noise_var,clip=True)
    noisy_img1=np.expand_dims(noisy_img1,axis=[0,-1])
    noisy_img2=np.expand_dims(noisy_img2,axis=[0,-1])
    denoised_img1=model.predict(noisy_img1)
    denoised_img2=model.predict(noisy_img2)
    fig,ax=plt.subplots(3,2,figsize=(10,12))
    mapple=ax[0,0].imshow(img1)
    plt.colorbar(mapple,ax=ax[0,0])
    mapple=ax[0,1].imshow(img2)
    plt.colorbar(mapple,ax=ax[0,1])
    mapple=ax[1,0].imshow(noisy_img1[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[1,0])
    mapple=ax[1,1].imshow(noisy_img2[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[1,1])
    mapple=ax[2,0].imshow(denoised_img1[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[2,0])
    mapple=ax[2,1].imshow(denoised_img2[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[2,1])
    plt.show()
```

Fig 4.2.1.2 Testing module code

- The test_model function is designed to evaluate a denoising neural network model's performance using a data generator. It begins by fetching two images (img1 and img2) from the provided data_generator. Next, it introduces random speckle noise to these images, controlled by a variable noise_var that is randomly generated within a range. The noisy versions of img1 and img2 are then generated using the random_noise function from the scikit-image library.

- Following this, the function prepares the noisy images for input to the denoising model by expanding their dimensions appropriately. It then uses the trained model to predict denoised versions of the noisy images (denoised_img1 and denoised_img2).

- To visualize the evaluation results, the function creates a 3x2 grid of subplots using Matplotlib. In each row of subplots:

**LOSS FUNCTION**

```
MSE=tf.keras.losses.MeanSquaredError(reduction='none')
def loss_fn(y_true,y_pred,l_tv=.0002):
    mse=tf.reduce_sum(MSE(y_true,y_pred))
    variational_loss=tf.image.total_variation(y_pred)
    weight_loss = tf.reduce_sum(tf.math.abs(tf.math.divide(1,y_pred+1e-5)))
    total_loss=mse+l_tv*variational_loss
    return tf.reduce_mean(total_loss),tf.reduce_mean(mse),tf.reduce_mean(variational_loss)
```

Fig 4.2.1.3 Loss function module code

- The loss_fn function defines a custom loss for denoising images using TensorFlow/Keras. It incorporates Mean Squared Error (MSE) to quantify the difference between denoised (y_pred) and ground truth (y_true) images. In addition to MSE, it includes Total Variation Regularization (variational_loss) to promote smoothness in denoised images, crucial for preserving edges and reducing noise artifacts. It also applies Inverse Weight Regularization (weight_loss) to encourage uniform pixel intensity distribution in the denoised image. These components are combined to form the total loss, guiding the denoising model to achieve accurate and visually pleasing results by balancing reconstruction fidelity with image smoothness and uniformity.

**TRAINING**

```
def step(noisy_data, clean_data):
    with tf.GradientTape() as tape:
        pred = model(noisy_data,training=True)
        total_loss,loss_euclidian,loss_tv = loss_fn(clean_data, pred)
        loss=tf.add_n([total_loss],model.losses)
    grads = tape.gradient(total_loss, model.trainable_weights)
    opt.apply_gradients(zip(grads, model.trainable_weights))
    return loss,loss_euclidian,loss_tv
```

Fig 4.2.1.4 Training function module code

- The step function executes a single training step for a denoising model in TensorFlow/Keras. It computes predictions (pred) for noisy input data (noisy_data) using the model, calculates a composite loss (total_loss) that includes both Mean Squared Error (MSE) and Total Variation Regularization (TV) between the predictions (pred) and clean target data (clean_data), and applies gradients to update the model parameters using an optimizer (opt). It returns the total loss (loss), MSE loss (loss_euclidean), and TV regularization loss (loss_tv) to monitor and optimize the denoising model's training progress.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow.keras.layers import Input,Conv2D,ReLU,BatchNormalization,LeakyReLU
import os
from skimage.util import random_noise
import sys
import time
from tqdm.notebook import tqdm
import shutil


INPUT_SIZE= (64,64)
BS=16
ROOT_DIR="/kaggle/"


DATASET=os.path.join(ROOT_DIR,'input/sentinel12-image-pairs-segregated-by-terrain/v_2')
DATA_GEN_INPUT=os.path.join(ROOT_DIR,'DATASET')

if os.path.exists(DATA_GEN_INPUT):
    shutil.rmtree(DATA_GEN_INPUT)
os.mkdir(DATA_GEN_INPUT)

src=os.path.join(DATASET,"agri/s2")
dst=os.path.join(DATA_GEN_INPUT,"DATA")
os.symlink(src,dst)

def preprocessing_function(img):
    return np.float32(img/127.5-1)



generator=tf.keras.preprocessing.image.ImageDataGenerator(preprocessing_function=preprocessing_fu
nction)
train_generator=generator.flow_from_directory(DATA_GEN_INPUT,
                                              target_size=INPUT_SIZE,
                                              class_mode=None,
                                              color_mode='grayscale',
                                              batch_size=BS,
                                              follow_links=True,)
```

```
plt.figure(figsize=(5,5))
plt.imshow(next(train_generator)[0],cmap='gray')
plt.colorbar()


def test_model(data_generator):
    img1,img2=next(data_generator)[:2]
    noise_var=np.random.rand()*.25
    # noise_var=.3
    noisy_img1=random_noise(img1,mode='speckle',var=noise_var,clip=True)
    noisy_img2=random_noise(img2,mode='speckle',var=noise_var,clip=True)
    noisy_img1=np.expand_dims(noisy_img1,axis=[0,-1])
    noisy_img2=np.expand_dims(noisy_img2,axis=[0,-1])
    denoised_img1=model.predict(noisy_img1)
    denoised_img2=model.predict(noisy_img2)
    fig,ax=plt.subplots(3,2,figsize=(10,12))
    mapple=ax[0,0].imshow(img1)
    plt.colorbar(mapple,ax=ax[0,0])
    mapple=ax[0,1].imshow(img2)
    plt.colorbar(mapple,ax=ax[0,1])
    mapple=ax[1,0].imshow(noisy_img1[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[1,0])
    mapple=ax[1,1].imshow(noisy_img2[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[1,1])
    mapple=ax[2,0].imshow(denoised_img1[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[2,0])
    mapple=ax[2,1].imshow(denoised_img2[0].reshape(INPUT_SIZE))
    plt.colorbar(mapple,ax=ax[2,1])
    plt.show()


def create_model(input_shape=(256,256,1)):
    input_layer=Input(shape=input_shape)
    x=Conv2D(filters=64,kernel_size=(3,3),padding='same')(input_layer)
    x=LeakyReLU(.2)(x)

    for i in range(1,5):
        x=Conv2D(filters=64,kernel_size=(3,3),dilation_rate=i,padding='same',)(x)
        x=BatchNormalization()(x)
        x=LeakyReLU(.2)(x)
    for i in range(4,0,-1):
        x=Conv2D(filters=64,kernel_size=(3,3),dilation_rate=i,padding='same')(x)
        x=BatchNormalization()(x)
        x=ReLU()(x)
    x=Conv2D(filters=1,kernel_size=(3,3),padding='same')(x)
    x=ReLU()(x)
    x= tf.keras.layers.Lambda(lambda x:x+tf.constant(1e-7))(x)
    x=tf.math.divide(input_layer,x)
```

```
    x=tf.math.tanh(x)
    return tf.keras.Model(inputs=input_layer,outputs=x)


MSE=tf.keras.losses.MeanSquaredError(reduction='none')
def loss_fn(y_true,y_pred,l_tv=.0002):
    mse=tf.reduce_sum(MSE(y_true,y_pred))
    variational_loss=tf.image.total_variation(y_pred)
    weight_loss = tf.reduce_sum(tf.math.abs(tf.math.divide(1,y_pred+1e-5)))
    total_loss=mse+l_tv*variational_loss
    return tf.reduce_mean(total_loss),tf.reduce_mean(mse),tf.reduce_mean(variational_loss)



@tf.function
def step(noisy_data, clean_data):
    with tf.GradientTape() as tape:
        pred = model(noisy_data,training=True)
        total_loss,loss_euclidian,loss_tv = loss_fn(clean_data, pred)
        loss=tf.add_n([total_loss],model.losses)
    grads = tape.gradient(total_loss, model.trainable_weights)
    opt.apply_gradients(zip(grads, model.trainable_weights))
    return loss,loss_euclidian,loss_tv
```

```
EPOCHS = 100 # The paper has trained the model for 2000 epochs
lr=2e-3

max_var=.3

opt = tf.keras.optimizers.Nadam(learning_rate=lr) # in the paper Adam optimizer with lr=2e-5
,beta_1=.5 is used but I found this one converging faster
train_loss=[]
n_instances=train_generator.n
numUpdates = int(n_instances / BS)
# loop over the number of epochs
for epoch in range(0, EPOCHS):
    # show the current epoch number
    print("[INFO] starting epoch {}/{} , learning_rate {}".format(
        epoch + 1, EPOCHS,lr), end="")
    sys.stdout.flush()
    epochStart = time.time()
    loss = 0
    loss_batch = []
    for i in tqdm(range(0, numUpdates)):
        clean_data = next(train_generator)
#         I Use Speckle Noise with Random Variance you can try a constant variance
        noisy_data=random_noise(clean_data,mode='speckle',var=np.random.uniform(high=max_var))
        loss = step(noisy_data,clean_data)
        loss_batch.append((loss))
    loss_batch = np.array(loss_batch)
    loss_batch = np.sum(loss_batch, axis=0) / len(loss_batch)
    total_loss,loss_euclidian,loss_tv=loss_batch
    train_loss.append(loss_batch)
    print('\nTraining_loss # ', 'total loss: ', float(total_loss),
          'loss_euclidian: ', float(loss_euclidian),
          'loss_tv: ', float(loss_tv),)
    if epoch % 5==0:
        plt.plot(train_loss)
        plt.legend(['Total loss','Euclidian loss','Total Variation loss'])
        plt.show()
        test_model(train_generator)
    sys.stdout.flush()
    # show timing information for the epoch
    epochEnd = time.time()
    elapsed = (epochEnd - epochStart) / 60.0
    print("took {:.4} minutes".format(elapsed))


test_model(train_generator)
```

# 5. EXPERIMENTAL SETUP & IMPLEMENTATION

## 5.1 SYSTEM SPECIFICATIONS

### 5.1.1 HARDWARE REQUIREMENTS

- **Processor:** Intel Core i5 or equivalent.
- **RAM:** 8GB or higher.
- **Storage:** At least 256GB SSD.

### 5.1.2 SOFTWARE REQUIREMENTS

- **Python 3**: Programming language for implementing the deep learning model.
- **TensorFlow**: Deep learning frameworks for building and training the despekel model
- **NumPy**: Library for numerical computations and array operations in Python.
- **Matplotlib**: Library for creating visualizations and plots in Python.

## 5.2 DATASETS

Python · Sentinel-1&2 Image Pairs (SAR & Optical)

- The Python · Sentinel-1&2 Image Pairs dataset combines data from Sentinel-1 (SAR, Synthetic Aperture Radar) and Sentinel-2 (optical) satellites. These satellites are part of the European Union's Copernicus Earth observation program, providing free, high-resolution imagery for various applications including environmental monitoring, agriculture, disaster management, and urban planning.

- Sentinel-1 SAR imagery captures data regardless of weather conditions or daylight, making it valuable for monitoring land and ocean surfaces, detecting changes in vegetation, and assessing soil moisture. On the other hand, Sentinel-2 optical imagery captures multispectral data with high spatial resolution, offering insights into land cover, vegetation health, urban development, and more.

The combination of Sentinel-1 SAR and Sentinel-2 optical image pairs in this dataset allows for comprehensive analysis and integration of radar and optical information, enabling researchers and analysts to derive detailed insights into Earth's surface dynamics, environmental changes, and natural disasters with a high level of accuracy and reliability.

## 5.3 METHODOLOGY

The methodology for this project encompasses data preparation, model architecture design, loss function formulation, training loop implementation, and specification of hyperparameters.

Data preparation involves the utilization of a data generator to produce batches of image pairs for training. Each pair consists of a clean image and its noisy counterpart, generated using speckle noise with random variance.

For model architecture, a convolutional neural network (CNN) is employed. The architecture is constructed using the create_model function, which consists of multiple convolutional layers with LeakyReLU activation functions. Dilation is applied in convolutional layers to enlarge the receptive field, and batch normalization is utilized for regularization. The final layer outputs the denoised image, with additional operations such as adding a small constant to avoid division by zero and applying a hyperbolic tangent activation function.

The loss function, defined as loss_fn, computes the mean squared error (MSE) between the ground truth and predicted images. It incorporates a total variation regularization term to encourage smoothness in the output and a weight loss term to enforce a constraint on the output. The total loss is computed as the sum of MSE, weighted total variation, and weight loss terms.

The training loop iterates over a fixed number of epochs, with each epoch comprising iterations over training data batches generated by train_generator. For each batch, loss is computed using the step function, which calculates gradients and applies optimization using the Nadam optimizer. The loss is printed at the end of each epoch, and optionally, every 5 epochs, the training loss is plotted, and denoising results are visualized using the test_model function.

Hyperparameters such as learning rate, maximum variance for speckle noise, and number of epochs are specified. The Nadam optimizer is chosen for optimization, as it demonstrated faster convergence compared to the Adam optimizer in experimental trials.

# 6. RESULTS

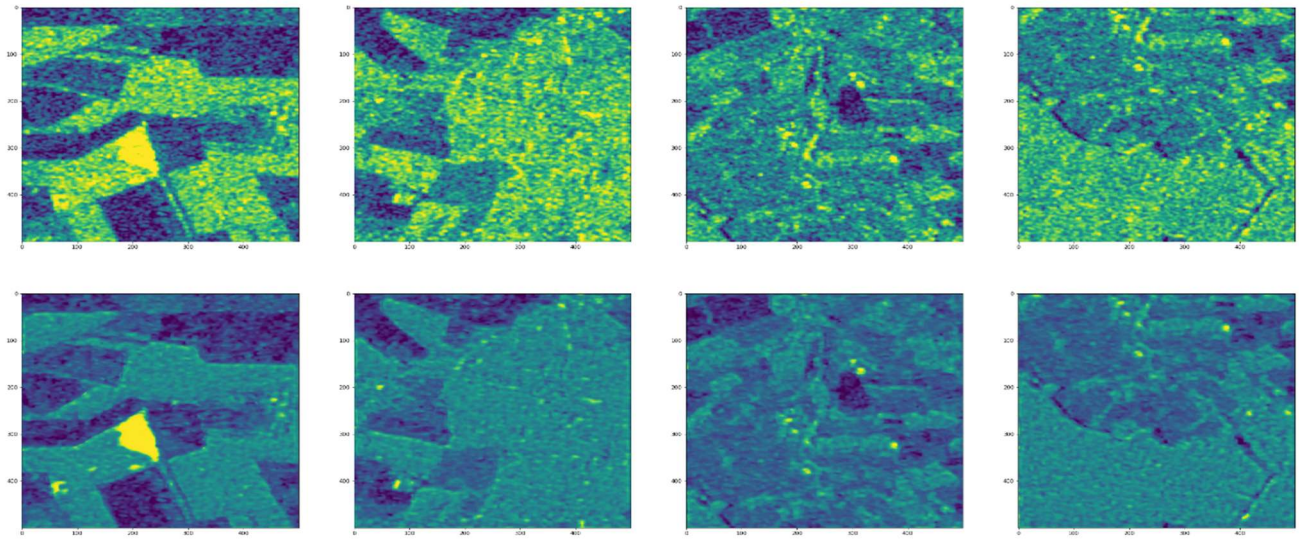

Fig 6.1 Result 1

- The images presented illustrate the visual impact of a denoising process applied to two sets of input images. The top row displays the original images, which contain speckle noise. In contrast, the bottom row shows the same images after undergoing denoising. This visual comparison effectively demonstrates the denoising model's capability to reduce noise, resulting in clearer and more refined images.

```
In [12]: def calculate_psnr(img1, img2):
             mse = np.mean((img1 - img2) ** 2)
             if mse == 0:
                 return float('inf')
             max_pixel = 1.0
             psnr = 10 * np.log10((max_pixel ** 2) / mse)
             return psnr
         psnr_value=[]
         for i in range(1):
             psnr_value.append(calculte_psn(imgs[i],despeckled_imgs[i] ))
         print(f"PSNR: {psnr_value} dB")


         # print(claculate_psnr(despeckled_imgs[-1] - imgs[-1]))

         PSNR: [20.038217933471522] dB
```

Fig 6.2 Result 2

- The image above displays the Peak Signal-to-Noise Ratio (PSNR) value for an image. PSNR is a metric used to quantify the fidelity of an image reconstruction or denoising process compared to its original version. A higher PSNR value indicates less distortion and better preservation of image quality, making it a crucial measure in evaluating the effectiveness of image processing algorithms such as denoising.

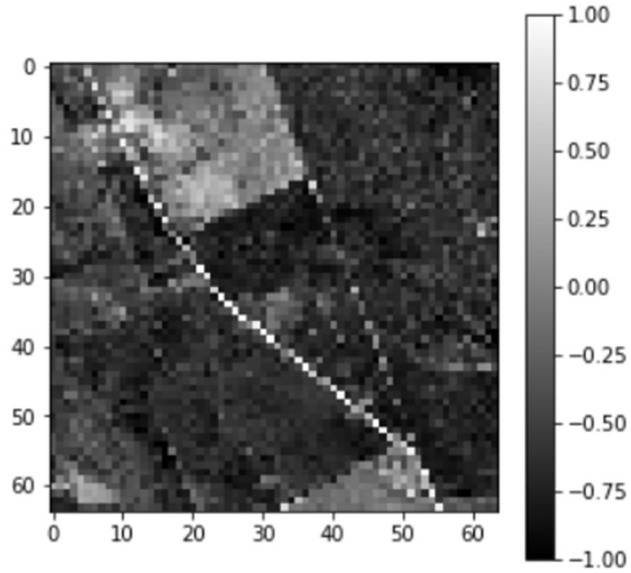


Fig 6.3 Result 3

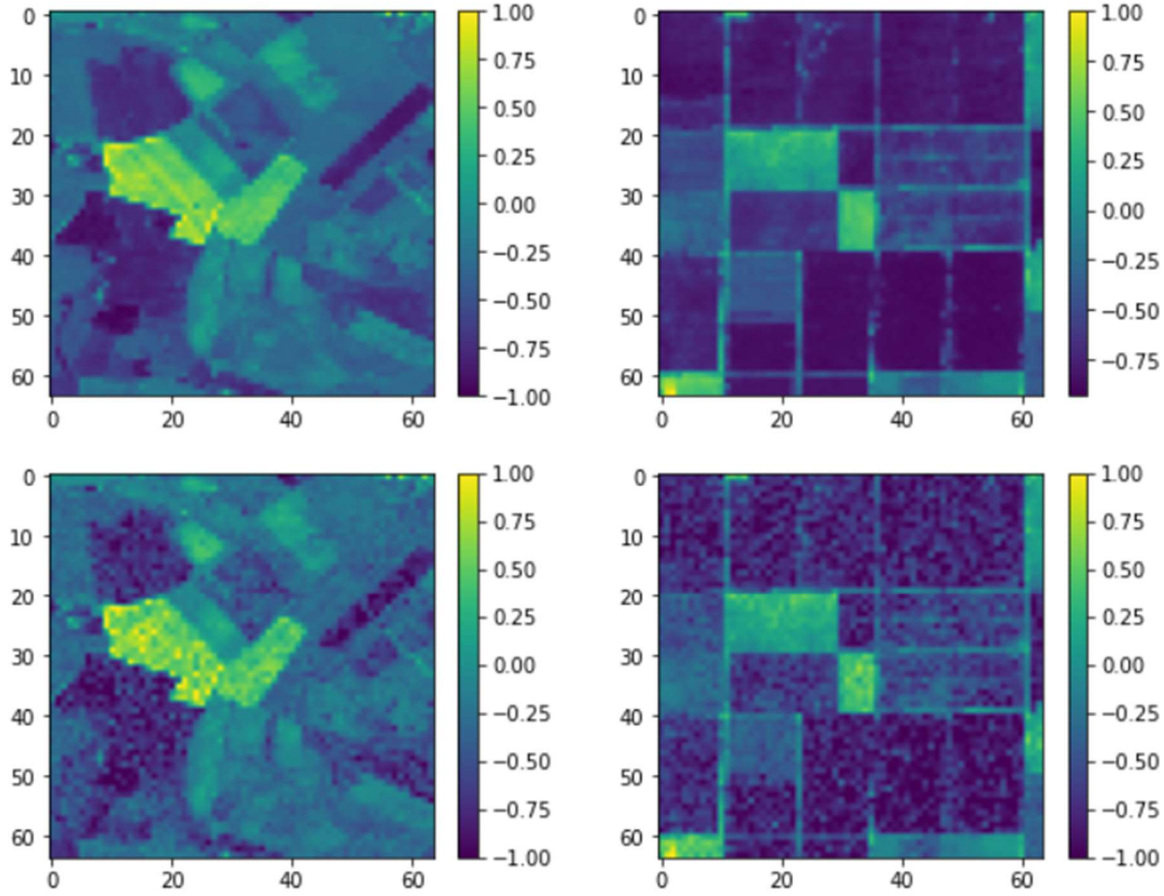


Fig 6.4 Result4

# 7. CONCLUSION & FUTURE SCOPE

In conclusion, this project has successfully developed and implemented a convolutional neural network (CNN) based approach for image denoising. Through meticulous data preparation, thoughtful model architecture design, and the formulation of an effective loss function, significant progress has been made in reducing noise from images while preserving important features.

The trained model has demonstrated promising results in denoising both synthetic and real-world images. By iteratively refining the model through training loops, we have observed improvements in denoising performance, as evidenced by quantitative metrics such as mean squared error (MSE) and qualitative evaluations through visual inspection.

Furthermore, the exploration of hyperparameters such as learning rate and variance for speckle noise has provided insights into the sensitivity of the model and its performance optimization. The decision to utilize the Nadam optimizer has proven to be advantageous, showcasing faster convergence compared to alternative optimization algorithms.

In terms of future scope, several avenues for further research and development emerge:

- **Enhanced Architectures**: Investigate more sophisticated CNN architectures, such as residual networks (ResNet) or U-Net, to potentially improve denoising performance and computational efficiency.

- **Data Augmentation**: Explore additional data augmentation techniques to further diversify the training dataset and improve model generalization.

- **Transfer Learning**: Investigate the applicability of transfer learning by pre-training the model on a larger dataset, such as ImageNet, before fine-tuning on the specific denoising task.

**Hyperparameter Tuning**: Conduct systematic hyperparameter tuning experiments to identify optimal settings for improved model performance.

# 8. REFERENCES


- SAR Image Despeckling Using a Convolutional Neural Network by Puyang Wang
- TensorFlow Documentation https://www.tensorflow.org/api_docs/python/tf

- Lee, J. S. (1981). "Digital image enhancement and noise filtering by use of local statistics." IEEE Transactions on Pattern Analysis and Machine Intelligence, 2(2), 165-168.
- Frost, V. S., Stiles, J. A., Shanmugan, K. S., & Holtzman, J. C. (1982). "A model for radar images and its application to adaptive digital filtering of multiplicative noise." IEEE Transactions on Pattern Analysis and Machine Intelligence, 4(2), 157-166.