**CHAPTER 1 :**

**INTRODUCTION**

Synthetic Aperture Radar (SAR) imaging has become pivotal in remote sensing for terrain classification, disaster management, and military applications. However, SAR’s speckle noise and varying texture signatures make classification non-trivial.

This project aims to develop an end-to-end system that semantically segments SAR images into multiple classes. From denoising and edge labeling to a trained multiclass model, this project demonstrates a complete deep learning pipeline for semantic segmentation on radar-based datasets.

**1.1 Overview:**

SAR imagery is a key modality in remote sensing due to its ability to penetrate cloud cover and provide surface details regardless of lighting. Its applications are diverse, spanning across military surveillance, agriculture monitoring, environmental studies, and disaster mapping.

**1.2 Applications of SAR Imagery:**

In defense scenarios, SAR helps identify urban layouts, movement pathways, or structural anomalies. In agriculture and environmental monitoring, SAR data allows classification of vegetation cover, water bodies, and barren zones. However, the presence of noise and variability in intensity patterns poses substantial difficulty in applying conventional classification methods.

**1.3 Project Motivation and Approach:**

This project introduces a machine learning pipeline designed to handle SAR-specific challenges and segment images into multiple semantic classes. The innovation lies in blending classical image processing (Gabor filtering, edge detection) with patch-wise training of a deep learning model, enabling it to learn high-level representations from limited yet information-rich data.

## CHAPTER 2:

## PROBLEM DEFINITION

SAR image interpretation is a challenging problem due to:

* High speckle noise.
* Limited labelled data.
* Multi-texture representations.

The aim is to process SAR data into clean, classifiable segments using automated learning techniques with edge and texture as primary features.

**2.1 Challenges in SAR Image Segmentation**:

SAR images are rich in structural and textural information, but they are often corrupted by speckle noise and have high dynamic range variations. Manual interpretation of such images is not scalable, and existing algorithms either require large annotated datasets or fail to distinguish similar texture classes under noise.

**2.2 Objective**:

The problem addressed here is the development of an efficient, scalable, and robust segmentation framework for SAR data that performs well on high-resolution images with limited data availability. The aim is to:

* Design a pre-processing pipeline to extract meaningful features from noisy SAR images.
* Generate accurate pixel-level labels using unsupervised clustering.
* Train a semantic segmentation model capable of multiclass classification with limited data.

**2.3 Methodology Highlights**

By leveraging techniques like Gabor-based texture extraction, edge morphology, and centroid-driven clustering, we address both the noise problem and the label generation bottleneck. Further, patch-based training allows the deep learning model to generalize better despite constrained GPU memory and dataset size.

## CHAPTER 3:

## DATASET

**3.1 Data Source and Structure:**

The SAR dataset used in this project was confidentially provided by the organization. It contains a set of high-resolution grayscale SAR images representing a variety of land cover types.

**3.2 Label Generation Strategy:**

Due to security constraints, the dataset is not publicly available and must be handled under institutional guidelines. The dataset includes raw SAR captures without labels. As such, custom labels were generated using Gabor filtering and edge-based morphological analysis, followed by clustering using KMeans.

**3.3 Tiling Strategy:**

Each image was of large dimensions (~1600×3500), requiring the use of image tiling during model training to process smaller patches while maintaining full spatial resolution. This approach helps in creating a balanced dataset where each patch contains adequate semantic information without overwhelming memory constraints.

**3.4 MATLAB Pre-processing**

Edge Detection: Canny + morphological filtering.

Gabor Labeling: Gabor filter bank applied to extract texture features.

KMeans Clustering: Clusters used for semantic labeling.

## CHAPTER 4: TECHNOLOGIES USED

### 4.1 MATLAB for Preprocessing

MATLAB was employed for classical image processing and pre-labeling due to its robust built-in functions and ease of matrix manipulation. Key functions used include:

* **imguidedfilter** – to perform edge-preserving smoothing for speckle noise reduction.
* **wdencmp** – wavelet decomposition and compression for noise suppression.
* **imgaborfilt** – applies a bank of Gabor filters to extract directional texture features.
* **edge** – Canny edge detector used for capturing structural outlines.

MATLAB scripts were used to generate three outputs per SAR image: a denoised image, edge map, and Gabor-based pseudo-label mask. These outputs were exported as .png images for downstream deep learning use.

### 4.2 Python for Model Design and Training

The deep learning pipeline was implemented in Python using:

* **Keras with TensorFlow backend** – for defining, training, and evaluating the U-Net architecture.
* Custom data generators – for loading, augmenting, and feeding tiles during training.
* Training loop with callbacks for early stopping and model checkpointing.

The model was trained using patch-wise inputs and optimized with categorical cross-entropy. TensorFlow enabled GPU acceleration for faster convergence.

### 4.3 Supporting Python Libraries

Several additional libraries supported the end-to-end pipeline:

* **OpenCV**: For reading/writing images, tiling large SAR images, and merging predictions.
* **NumPy**: For handling pixel data and matrix reshaping.
* **scikit-image**: For preprocessing, label encoding, and evaluation utilities such as label2rgb.
* **Albumentations** or custom augmentation code: To perform real-time image transformations like flips, rotations, and brightness shifts.

### 

### 4.4 Visualization and Version Control

* **Matplotlib** was used to visualize patches, ground truths, predictions, and overlay masks with custom colormaps.
* **Git** was used for version control and collaboration, allowing snapshot-based tracking of preprocessing, model scripts, and outputs.

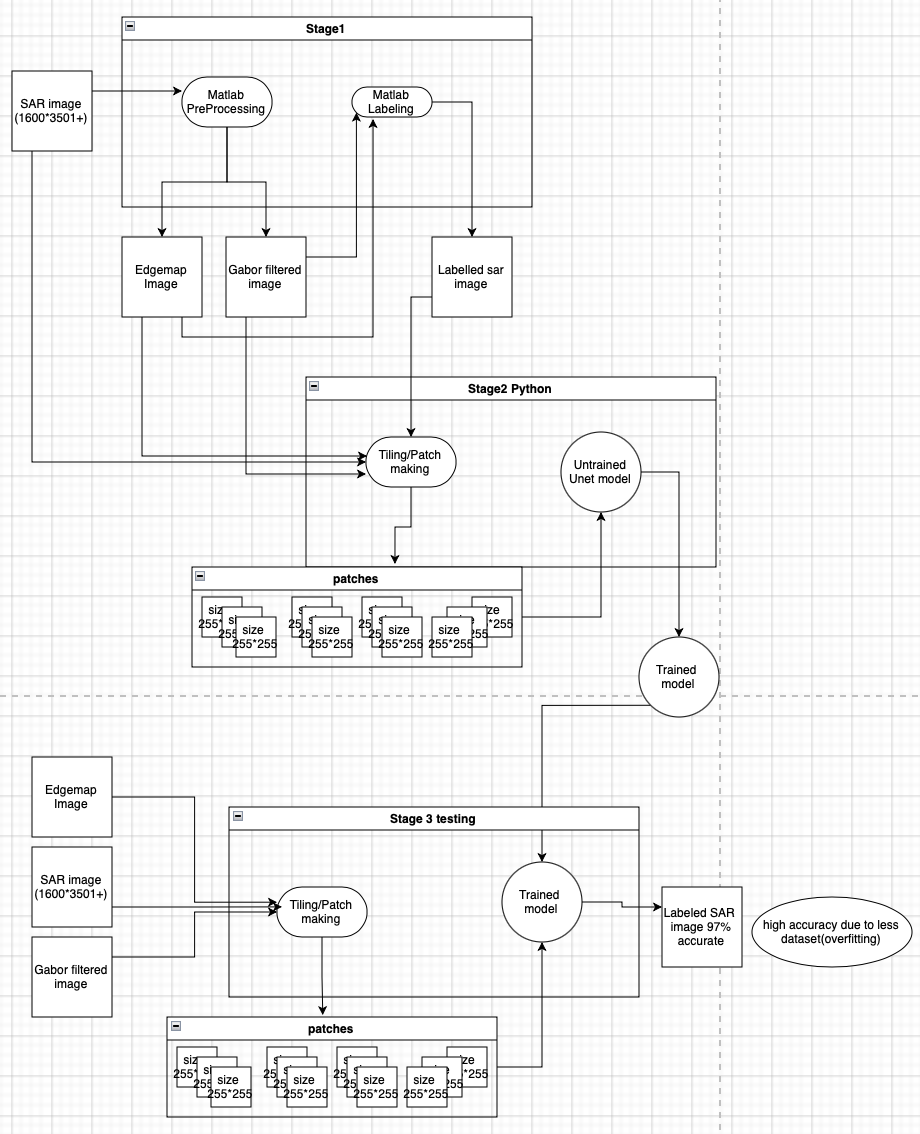


figure 4.1

## 

## 

## 

## 

## CHAPTER 5:

## SYSTEM DESIGN AND ARCHITECTURE

### 5.1 System Requirements

* **Hardware**:  
    - Minimum: 16 GB RAM, NVIDIA GPU (4GB+ VRAM)  
    - Recommended: 32 GB RAM, NVIDIA RTX GPU (8GB+ VRAM)
* **Software**:  
    - MATLAB (R2022b or later)  
    - Python 3.10+  
    - TensorFlow 2.12+, Keras, OpenCV  
    - scikit-image, NumPy, Matplotlib  
    - Git for version control

### 5.2 High-Level System Modules

The complete pipeline is composed of three independent modules:

* **Pre-processing and Label Generation (MATLAB):**  Denoising, edge detection, and Gabor-based pseudo-label generation.
* **Training Data Construction (Python):**  Stacking SAR, edge, and label channels into 256×256 tiles for model training.
* **Model Training and Inference (Python):**  Patch-wise training of a custom U-Net model for semantic segmentation.

### 5.3 Input Pipeline and Data Preparation

Each SAR image is divided into 256×256 patches. These patches are composed of multi-channel inputs: grayscale SAR, edge maps, and Gabor-based clusters, forming inputs of shape (256, 256, 2) or (256, 256, 3).

### 5.4 U-Net Architecture Overview

The U-Net model is a fully convolutional network with an encoder-decoder design and skip connections to preserve spatial information during upsampling.

#### 1 Encoder (Contracting Path)

* Convolution layers (3×3) + ReLU
* Batch Normalization
* MaxPooling (2×2)
* Dropout for regularization

#### 

#### 2 Decoder (Expanding Path)

* Transposed convolutions (up-convolutions)
* Skip connections from encoder layers
* Convolution → BatchNorm → Activation

#### 3 Output Layer

* 1×1 convolution for channel projection
* Softmax activation
* Output shape: (256, 256, number of classes)

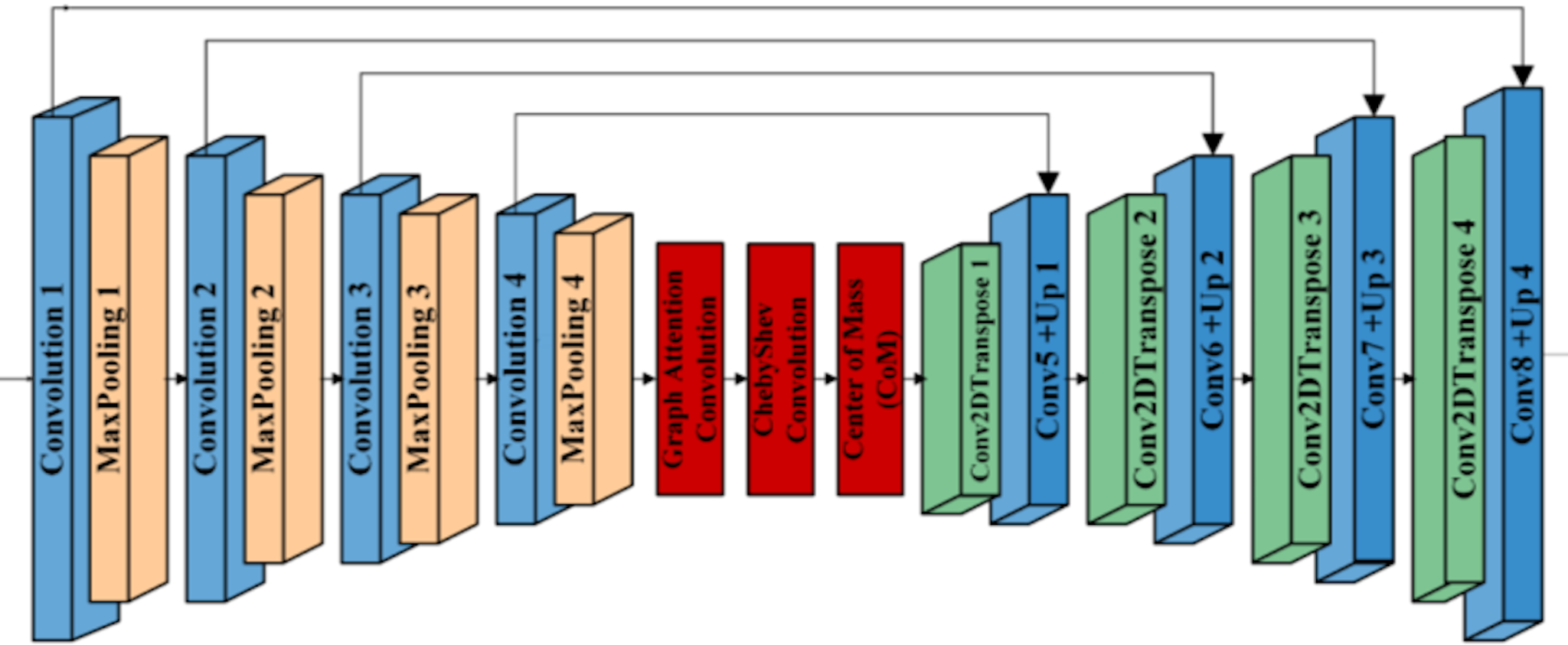


figure 5.1

### 5.5 U-Net Modifications

* Support for multi-channel SAR input
* Reduced depth to prevent overfitting
* Dropout regularization
* Real-time augmentation integrated in the generator

### 5.6 Loss Function and Optimization

* Loss: Categorical Crossentropy
* Optimizer: Adam
* Metrics: Class-wise accuracy, Mean IoU

### 5.7 Training Process

* Train/validation split: 80:20
* Epochs: 10
* Early stopping based on validation loss
* Augmentation: flipping, rotation, brightness variation

### 5.8 Prediction and Reconstruction

* Patch-wise inference
* Reconstructed into full image using tiling metadata
* Optional filtering of noise in post-processing
* Visualized with color-mapped classes

## CHAPTER 6:

## DATA PREPROCESSING AND IMPLEMENTATION

### 6.1 MATLAB Stage

The SAR images underwent several preprocessing steps to extract meaningful features and generate training labels:

* **Denoising**:
  + imguidedfilter for speckle noise smoothing while retaining edges.
  + wdencmp for multi-scale wavelet-based denoising.
* **Edge Extraction**:
  + edge function using Canny method, followed by imclose, imfill, and bwareaopen to clean and close edge gaps.
* **Gabor Feature Map**:
  + imgaborfilt applied with various orientations and frequencies.
  + Responses smoothed with a Gaussian kernel to reduce outliers.
* **Clustering for Pseudo-labels**:
  + Gabor responses reshaped into feature vectors per pixel.
  + KMeans clustering assigns a class to each pixel.
  + Cluster ID map reshaped to original image dimensions to form the label mask.
* **Output Directory Structure**:
  + /Denoised: Noise-suppressed images
  + /Cleaned\_Edges: Binary edge maps
  + /Gabor\_Labelled: Cluster-based segmentation masks

### 

### 

### 

### 6.2 Python Stage

The labelled data from MATLAB was further processed and used for training the U-Net model:

* **Tiling**:  
  Large images and corresponding label masks were divided into fixed-size patches (256×256) to enable mini-batch training.
* **Model**:  
  A custom U-Net was built using Keras with:
  + An encoder-decoder structure with skip connections
  + Dropout and batch normalization for regularization
  + Softmax activation for pixel-wise multiclass classification
* **Training**:
  + A custom generator loaded batches with real-time augmentation
  + Trained with Adam optimizer and categorical cross-entropy loss
  + Evaluation metrics: per-class accuracy and mean IoU
* **Prediction and Reconstruction**:
  + Test images tiled similarly and passed through the trained model
  + Predicted patches were merged back to reconstruct the full-sized mask
  + Final masks were visualized using custom color maps for each class

## 

## CHAPTER 7: CONCLUSION

This project demonstrates a practical pipeline for semantic segmentation of SAR images. By combining edge detection, Gabor filtering, and deep learning, the system handles noise, class variability, and limited annotations effectively.

The customized U-Net model, aided by patch-based training and augmentation, achieves robust classification on a limited dataset. The approach proves valuable for scalable, real-time SAR interpretation systems in defense and environmental applications.

During my internship at LRDE, DRDO, I had the opportunity to work on a technically intensive project involving Multiclass semantic segmentation of SAR images using both classical and deep learning methods. While the experience was highly enriching, it also highlighted several areas for future improvement.

A key challenge encountered was overfitting during model training, primarily due to the limited size of the dataset. Although patch-wise training and data augmentation mitigated the issue to some extent, a larger and more diverse dataset, along with techniques such as transfer learning or semi-supervised learning, could further improve generalization. Additionally, I aim to enhance my skills in optimizing memory usage during batch processing, improve code modularity, and develop a deeper understanding of SAR-specific image properties and preprocessing strategies.

Despite these challenges, the internship provided a solid foundation in working with guided filtering, Gabor-based texture analysis, KMeans clustering, and the design of a custom U-Net model in Python using TensorFlow. I also gained experience in modular system design, data preparation pipelines, augmentation techniques, and performance evaluation using metrics like IoU and per-class accuracy.

Working at LRDE, a premier defense research organization, gave me valuable insights into structured research practices, reproducibility, and mission-driven problem-solving. The internship strengthened my skills in both algorithmic thinking and applied implementation, preparing me well for future academic or industry roles in image analysis and cybersecurity-based systems.

## CHAPTER 8: References