Project Report

Lossy Compression of Sentinel-1 SAR Data Using SimCLR-based Neural Embeddings

1. Introduction

In recent years, the volume of Earth observation (EO) data has significantly increased due to the frequent acquisitions by satellite constellations such as Sentinel-1. This growth has led to the need for effective and efficient data compression techniques that maintain essential information for downstream tasks such as land use classification and change detection. This project proposes a hybrid approach combining traditional image preprocessing with contrastive self-supervised learning (SimCLR) to compress Sentinel-1 SAR data into 1024-dimensional embeddings.

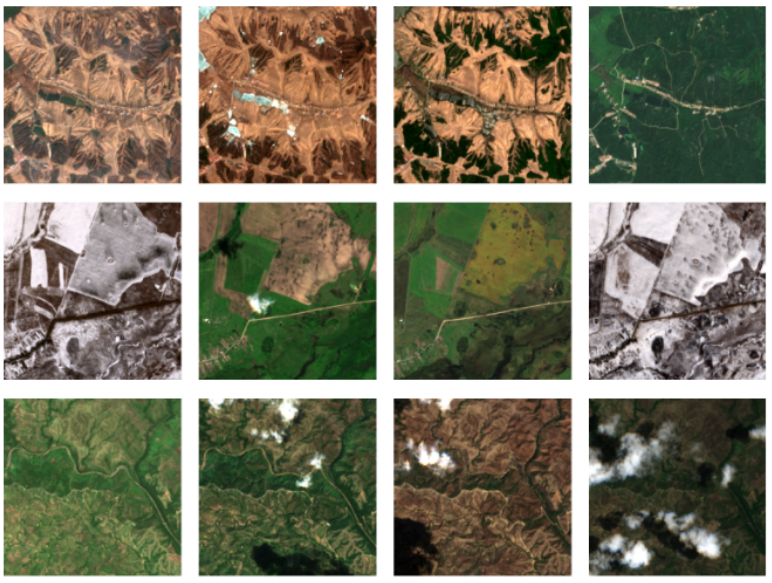


Fig: Sentinel1 SAR-Images

2. Dataset Description

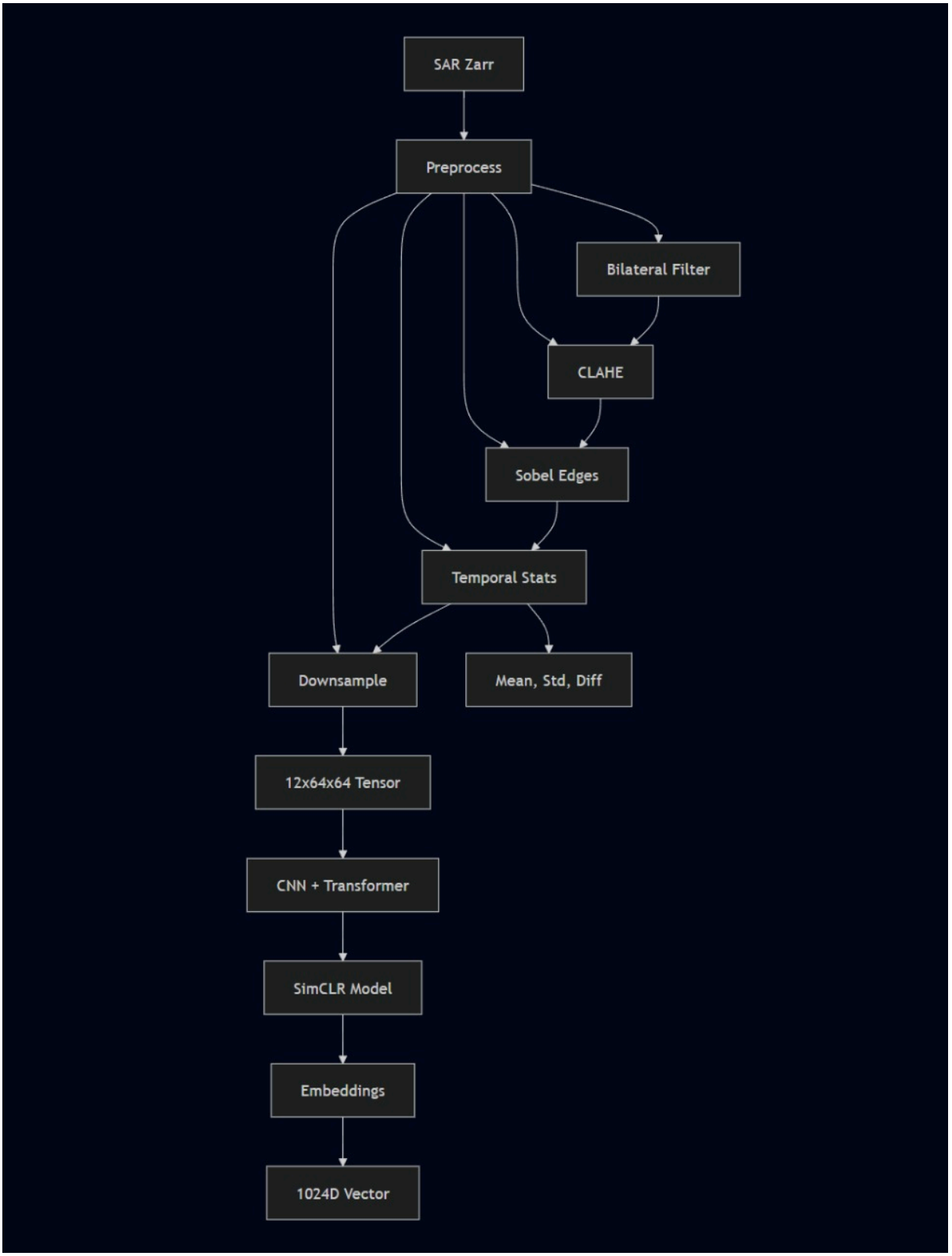
We used the Sentinel-1 Ground Range Detected (S1GRD) dataset provided in the SSL4EO-S12-v1.1 benchmark. The dataset consists of:

- SAR images in Zarr format.

- Each Zarr folder contains 64 samples.
- Each sample has 4 temporal snapshots.
- Each snapshot includes 2 polarization bands: VV and VH.

3. Data Preprocessing Pipeline

We designed a preprocessing pipeline to reduce noise, enhance contrast, and extract edge features from the SAR images. These steps include:



3.1. Denoising (Bilateral Filter)

```
def bilateral_denoise(image):
    return cv2.bilateralFilter(image.astype(np.float32), d=9, sigmaColor=75,
```

```
sigmaSpace=75)
```

This preserves edges while removing noise, suitable for SAR data which contains speckle noise.

3.2. Contrast Enhancement (CLAHE)

```
def apply_clahe(image):
    norm = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX).astype
(np.uint8)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    return clahe.apply(norm)
```

CLAHE enhances local contrast, which helps improve visibility of features for learning.

3.3. Edge Detection (Sobel)

```
def sobel_edges(image):
    grad_x = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=3)
    grad_y = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=3)
    magnitude = cv2.magnitude(grad_x, grad_y)
    return magnitude
```

Sobel edge maps highlight linear features such as roads and field boundaries.

3.4. Temporal Feature Aggregation

For each sample:

- Stack 4 temporal frames.
- Extract VV/VH denoised, VV CLAHE, and Sobel edge maps.
- Generate temporal features:
    - Mean image
    - Standard deviation
    - Difference (Last - First)

3.5. Downsampling

```
def downsample(image, target_shape=(64, 64)):
    return resize(image, target_shape, mode='reflect', preserve_range=True,
anti_aliasing=True)
```

The 12-channel temporal feature cube is spatially downsampled to reduce input size.

Final Output Per Sample

- Shape: `[12, 64, 64]` (temporal stack of 3 features over 4 channels)

4. Model Architecture

We use a SimCLR-style encoder combining convolutional layers and a Transformer for embedding extraction.

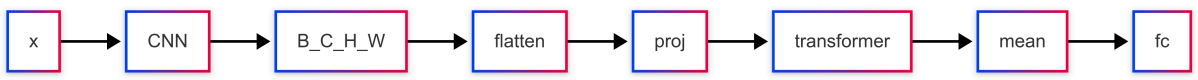
4.1. CNN Feature Extractor

- 4 convolutional layers with increasing channels: 64 → 512
- Kernel size: 3, stride: 2, padding: 1
- Each followed by BatchNorm and ReLU

4.2. Projection & Transformer

- Project 512D CNN output to 128D
- Flatten to sequence for transformer input
- Use 2-layer TransformerEncoder with 8 heads
- Final linear layer maps mean pooled sequence to 1024D compressed embedding

Model Forward Flow



5. Loss Function - NT-Xent

We use the normalized temperature-scaled cross entropy loss from SimCLR:

```
def nt_xent_loss(z_i, z_j, temperature=0.5):
    ...
    return F.cross_entropy(logits, labels)
```

This encourages representations of augmented views of the same sample to be close in embedding space.

6. Training Details

- Optimizer: Adam
- Learning Rate: 1e-4
- Batch Size: 64
- Epochs: 10
- Training on 100 folders of SAR samples

Training loop includes:

```
for x1, x2 in loader:
    z1 = model(x1)
    z2 = model(x2)
    loss = nt_xent_loss(z1, z2)
    ...
```

7. Embedding Extraction

After training, embeddings are extracted from the full dataset (398 folders):

- Forward pass only
- Outputs are saved as a CSV file for further analysis or downstream use

```
def extract_embeddings(model, dataset):
    ...
    df.to_csv("sar_embeddings.csv", index=False)
```

8. Output

- Final trained model weights
- Embeddings CSV

10. Conclusion

This project presents an efficient method to compress temporal SAR data using self-supervised learning. By integrating classical preprocessing techniques with modern deep learning models (CNN + Transformer + SimCLR), we produce compact, informative embeddings that preserve spatiotemporal patterns critical for downstream geospatial analytics.

11. Future Work

- Incorporate seasonal encoding or positional embedding to improve temporal modeling.
- Experiment with masked autoencoders for even stronger compression.
- Fine-tune on downstream tasks like land use classification or flood detection.

✅ **Outcome:** ~7000x compression ratio from full SAR cubes to 1024D embeddings, ready for scalable geospatial ML.