# Post-Disaster Damage Assessment for Nepal Earthquake Using Satellite Imagery

## Overview

This guide provides a comprehensive approach to conducting post-disaster damage assessment for the November 2023 Jajarkot earthquake in Nepal (continuing impacts into 2024) using satellite imagery and Google Earth Engine.

## Event Details

- **Date**: November 3, 2023 (11:47 PM local time)
- **Magnitude**: 6.4 ML (5.7 Mw)
- **Epicenter**: Ramidanda, Jajarkot District (28°50'24''N, 82°11'24''E)
- **Affected Districts**: Jajarkot, Rukum West, and Salyan (primarily)
- **Impact**: 154 deaths, 366+ injured, 75,000+ houses affected

## Step 1: Data Sources

### Satellite Imagery Sources

#### 1. **Sentinel-2 (ESA)**

- Resolution: 10m (optical)
- Revisit time: 5 days
- Free access via Google Earth Engine
- Best for: Overall damage assessment, landslide detection

#### 2. **Landsat 8/9 (USGS)**

- Resolution: 30m (optical)
- Revisit time: 16 days
- Free access via Google Earth Engine
- Best for: Large-scale changes, thermal analysis

#### 3. **Planet Labs**

- Resolution: 3-5m
- Daily revisit

- Requires subscription (academic access available)
- Best for: Detailed building damage

**4. Maxar (WorldView, GeoEye)**

- Resolution: 0.3-0.5m
- On-demand tasking
- Commercial (may have open data for disasters)
- Best for: Very detailed damage assessment

### Data Access Platforms

1. **Google Earth Engine (GEE)**
   - Primary platform for analysis
   - Free access with Google account
   - Sign up: https://earthengine.google.com
2. **Copernicus Emergency Management Service**
   - May provide rapid mapping products
   - https://emergency.copernicus.eu
3. **USGS EarthExplorer**
   - For downloading raw imagery
   - https://earthexplorer.usgs.gov
4. **NASA Disasters Portal**
   - https://disasters.nasa.gov

## Step 2: Python Environment Setup

python

```
# Install required packages
pip install earthengine-api
pip install folium
pip install geemap
pip install rasterio
pip install geopandas
pip install scikit-learn
pip install opencv-python
```

### Step 3: Google Earth Engine Authentication

```python
import ee
import geemap
import folium
import datetime
import pandas as pd
import numpy as np

# Authenticate and initialize Earth Engine
ee.Authenticate()
ee.Initialize()
```

### Step 4: Define Area of Interest (AOI)

```python
# Define the affected area coordinates
# Jajarkot epicenter and surrounding areas
epicenter = ee.Geometry.Point([82.19, 28.84])

# Create a buffer around epicenter (50km radius for initial assessment)
aoi = epicenter.buffer(50000).bounds()

# Alternatively, define specific district boundaries
jajarkot_coords = [
    [82.0, 28.7],
    [82.4, 28.7],
    [82.4, 29.0],
    [82.0, 29.0],
    [82.0, 28.7]
]
aoi_polygon = ee.Geometry.Polygon(jajarkot_coords)
```

### Step 5: Retrieve Pre and Post-Earthquake Imagery

```python
# Define date ranges
earthquake_date = '2023-11-03'

# Pre-earthquake period (1-2 months before)
pre_start = '2023-09-01'
pre_end = '2023-11-02'

# Post-earthquake period (immediately after)
post_start = '2023-11-04'
post_end = '2023-11-30'

# Function to get Sentinel-2 imagery
def get_sentinel2_image(start_date, end_date, aoi):
    # Load Sentinel-2 Surface Reflectance collection
    collection = ee.ImageCollection('COPERNICUS/S2_SR') \
        .filterBounds(aoi) \
        .filterDate(start_date, end_date) \
        .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))

    # Get median composite to reduce clouds
    image = collection.median().clip(aoi)

    # Select relevant bands
    return image.select(['B4', 'B3', 'B2', 'B8', 'B11', 'B12'])

# Get pre and post imagery
pre_image = get_sentinel2_image(pre_start, pre_end, aoi)
post_image = get_sentinel2_image(post_start, post_end, aoi)
```

## Step 6: Calculate Damage Indices

python

```python
# 1. Normalized Difference Vegetation Index (NDVI) Change
def calculate_ndvi(image):
    return image.normalizedDifference(['B8', 'B4']).rename('NDVI')

pre_ndvi = calculate_ndvi(pre_image)
post_ndvi = calculate_ndvi(post_image)
ndvi_change = post_ndvi.subtract(pre_ndvi).rename('NDVI_change')

# 2. Normalized Burn Ratio (NBR) for landslide detection
def calculate_nbr(image):
    return image.normalizedDifference(['B8', 'B12']).rename('NBR')

pre_nbr = calculate_nbr(pre_image)
post_nbr = calculate_nbr(post_image)
nbr_change = pre_nbr.subtract(post_nbr).rename('dNBR')

# 3. Built-up Area Change Detection
def calculate_built_index(image):
    # Normalized Difference Built-up Index (NDBI)
    return image.normalizedDifference(['B11', 'B8']).rename('NDBI')

pre_built = calculate_built_index(pre_image)
post_built = calculate_built_index(post_image)
built_change = post_built.subtract(pre_built).rename('NDBI_change')
```

## Step 7: Classify Damage Levels

python

```python
# Define damage thresholds
def classify_damage(change_image, index_name):
    if index_name == 'dNBR':
        # NBR difference thresholds for severity
        classified = ee.Image(0) \
            .where(change_image.lt(0.1), 1) \
            .where(change_image.gte(0.1).And(change_image.lt(0.27)), 2) \
            .where(change_image.gte(0.27).And(change_image.lt(0.44)), 3) \
            .where(change_image.gte(0.44).And(change_image.lt(0.66)), 4) \
            .where(change_image.gte(0.66), 5)

        return classified.rename('damage_class')

    elif index_name == 'NDVI_change':
        # NDVI change thresholds
        classified = ee.Image(0) \
            .where(change_image.gt(-0.1), 1) \
            .where(change_image.lte(-0.1).And(change_image.gt(-0.25)), 2) \
            .where(change_image.lte(-0.25).And(change_image.gt(-0.4)), 3) \
            .where(change_image.lte(-0.4), 4)

        return classified.rename('vegetation_loss')

# Classify damage
damage_nbr = classify_damage(nbr_change, 'dNBR')
vegetation_loss = classify_damage(ndvi_change, 'NDVI_change')
```

## Step 8: Landslide Detection

python

```python
# Detect potential landslides
def detect_landslides(pre_image, post_image, slope):
    # Calculate spectral change
    spectral_change = post_image.subtract(pre_image).pow(2).reduce(ee.Reducer.sum()).sqrt()

    # Combine with slope information
    landslide_susceptibility = spectral_change.multiply(slope.divide(90))

    # Threshold for landslide detection
    landslides = landslide_susceptibility.gt(0.3)

    return landslides.rename('landslide_areas')

# Get slope data from SRTM
srtm = ee.Image('USGS/SRTMGL1_003')
slope = ee.Terrain.slope(srtm)

# Detect landslides
landslides = detect_landslides(pre_image, post_image, slope)
```

### Step 9: Building Damage Assessment

python

```python
# For building damage, we need higher resolution imagery
# This example uses Sentinel-2, but ideally use Planet or Maxar data

def assess_building_damage(pre_image, post_image):
    # Calculate texture features for built areas
    gray_pre = pre_image.select('B4')
    gray_post = post_image.select('B4')

    # Calculate GLCM texture
    glcm_pre = gray_pre.glcmTexture(size=3)
    glcm_post = gray_post.glcmTexture(size=3)

    # Compare contrast changes (damaged buildings show different texture)
    contrast_change = glcm_post.select('B4_contrast').subtract(
        glcm_pre.select('B4_contrast')
    )

    # Threshold for damage
    building_damage = contrast_change.gt(10).rename('building_damage')

    return building_damage

building_damage = assess_building_damage(pre_image, post_image)
```

## Step 10: Visualization

python

```python
# Create visualization parameters
vis_params_rgb = {
    'bands': ['B4', 'B3', 'B2'],
    'min': 0,
    'max': 3000,
    'gamma': 1.4
}

damage_vis = {
    'min': 1,
    'max': 5,
    'palette': ['green', 'yellow', 'orange', 'red', 'darkred']
}

# Create map
Map = geemap.Map(center=[28.84, 82.19], zoom=10)

# Add layers
Map.addLayer(pre_image, vis_params_rgb, 'Pre-earthquake')
Map.addLayer(post_image, vis_params_rgb, 'Post-earthquake')
Map.addLayer(damage_nbr, damage_vis, 'Damage Severity')
Map.addLayer(landslides.selfMask(), {'palette': 'red'}, 'Landslides')
Map.addLayer(building_damage.selfMask(), {'palette': 'purple'}, 'Building Damage')

# Add legend
legend_dict = {
    'No damage': 'green',
    'Low': 'yellow',
    'Moderate': 'orange',
    'High': 'red',
    'Severe': 'darkred'
}
Map.add_legend(legend_dict=legend_dict)

Map
```

## Step 11: Export Results

python

```python
# Export damage assessment to Google Drive
export_task = ee.batch.Export.image.toDrive(
    image=damage_nbr,
    description='Jajarkot_earthquake_damage_assessment',
    folder='earthquake_assessment',
    fileNamePrefix='damage_severity_2023',
    region=aoi,
    scale=10,
    maxPixels=1e13
)

export_task.start()

# Export statistics
def calculate_damage_statistics(damage_image, aoi):
    # Calculate area for each damage class
    pixel_area = ee.Image.pixelArea()

    areas = damage_image.addBands(pixel_area).reduceRegion(
        reducer=ee.Reducer.sum().group(
            groupField=0,
            groupName='damage_class'
        ),
        geometry=aoi,
        scale=100,
        maxPixels=1e13
    )

    return areas.getInfo()

stats = calculate_damage_statistics(damage_nbr, aoi)
print("Damage Statistics:", stats)
```

## Step 12: Generate Report

python

```python
# Create damage assessment report
import matplotlib.pyplot as plt
from datetime import datetime

def generate_report(stats, aoi_name="Jajarkot District"):
    # Create report
    report = f"""
    POST-EARTHQUAKE DAMAGE ASSESSMENT REPORT
    =======================================
    Location: {aoi_name}, Nepal
    Event Date: November 3, 2023
    Analysis Date: {datetime.now().strftime("%Y-%m-%d")}

    DAMAGE SUMMARY:
    - Total Affected Area: {sum(stats.values())/1e6:.2f} km²
    - Severe Damage: {stats.get(5, 0)/1e6:.2f} km²
    - High Damage: {stats.get(4, 0)/1e6:.2f} km²
    - Moderate Damage: {stats.get(3, 0)/1e6:.2f} km²
    - Low Damage: {stats.get(2, 0)/1e6:.2f} km²
    - No/Minimal Damage: {stats.get(1, 0)/1e6:.2f} km²

    RECOMMENDATIONS:
    1. Priority areas for immediate response: Severe and High damage zones
    2. Conduct ground truthing in moderate damage areas
    3. Monitor landslide-prone areas for continued risk
    4. Assess infrastructure damage using higher resolution imagery
    """

    return report
```

## Additional Resources

### Ground Truth Data Sources

- OpenStreetMap building footprints
- Government damage assessments
- UN OCHA situation reports
- Local NGO field reports

### Advanced Techniques

1. **Machine Learning Classification**
   - Random Forest for damage classification
   - Deep learning with CNNs for building damage
2. **SAR Analysis**
   - Use Sentinel-1 SAR data for all-weather monitoring
   - Coherence analysis for structural damage
3. **Multi-temporal Analysis**
   - Track recovery progress over time
   - Monitor reconstruction efforts

### Validation Methods

- Compare with field survey data
- Cross-reference with other damage maps
- Calculate accuracy metrics (overall accuracy, kappa coefficient)

## Code Repository Structure

```
nepal_earthquake_assessment/
├── data/
│   ├── aoi/
│   ├── ground_truth/
│   └── exports/
├── scripts/
│   ├── 01_data_acquisition.py
│   ├── 02_preprocessing.py
│   ├── 03_damage_analysis.py
│   ├── 04_visualization.py
│   └── 05_reporting.py
├── notebooks/
│   └── damage_assessment_workflow.ipynb
├── results/
│   ├── maps/
│   ├── statistics/
│   └── reports/
└── README.md
```

## Important Notes

1. Always validate results with ground truth data

2. Consider seasonal variations (monsoon effects)
3. Account for topographic shadows in mountainous terrain
4. Coordinate with local authorities for data sharing
5. Follow ethical guidelines for disaster data use</content>