

Session 2: Advanced Palawan Land Cover Lab

Multi-temporal Classification and Change Detection

Stylianos Kotsopoulos
EU-Philippines CoPhil Programme

Session Overview

Duration: 2 hours
Type: Advanced Hands-on Lab
Focus: Real-world NRM Application
Study Area:
Palawan Biosphere Reserve (11,655 km²)

Prerequisites: Session 1 completed, GEE authenticated

What You'll Learn:

- Advanced feature engineering (GLCM texture)
- Multi-temporal composites (dry/wet season)
- Hyperparameter optimization
- Deforestation detection (2020-2024)
- Protected area monitoring

Part A: Advanced Feature Engineering

Beyond Spectral Indices

Session 1 Features: - Spectral bands (B2-B12) - Vegetation indices (NDVI, EVI) - Water indices (NDWI, MNDWI)
- Built-up index (NDBI)

Session 2 Advanced Features: - **Texture** (GLCM) - **Temporal** (seasonal composites) - **Topographic** (DEM-derived)

Goal: Improve from 82% → 87%+ accuracy

GLCM Texture Features

What is GLCM?

Gray-Level Co-occurrence Matrix measures spatial relationships between pixel pairs.

Why Use It?

- Distinguishes **primary vs secondary forest** (canopy structure differences)
- Separates **urban from bare soil** (heterogeneity patterns)
- Identifies **mangrove stands** (unique texture signature)
- Adds context that spectral values alone miss

GLCM Texture Metrics

Contrast - Measures local variation - High for heterogeneous areas (urban, mixed forest) - Low for uniform areas (water, dense forest)

Entropy - Measures randomness - High for complex textures - Low for regular patterns

Correlation - Measures pixel relationships - Detects linear structures - Useful for roads, rivers

Homogeneity - Measures uniformity - High for smooth surfaces - Low for rough textures

GEE GLCM Implementation

```
1 # Extract NIR texture features
2 image_nir = composite.select('B8')
3
4 # Calculate GLCM (3x3 window)
5 texture = image_nir.glcmTexture(size=3)
6
7 # Select key metrics
8 contrast = texture.select('B8_contrast')
9 entropy = texture.select('B8_ent')
10 correlation = texture.select('B8_corr')
11 homogeneity = texture.select('B8_idm')
12
13 # Add to feature stack
14 features = features.addBands([contrast, entropy,
15                               correlation, homogeneity])
```

Computational Note: GLCM is intensive - use 3x3 windows for large areas

Multi-temporal Composites

Philippine Seasons

Dry Season (Dec-May)

- Less cloud cover (best for mapping)
- Maximum agricultural activity
- Forest at baseline state
- Ideal for structural analysis

Best for: - Forest type classification - Infrastructure detection - Land cover baseline

Wet Season (Jun-Nov)

- More cloud challenges
- Maximum vegetation vigor
- Rice fields flooded/growing
- Seasonal wetlands visible

Best for: - Agricultural identification - Crop phenology - Irrigated vs rainfed - Wetland mapping

Temporal Indices

NDVI Difference (Wet - Dry):

```
1 # Create seasonal composites
2 dry_composite = s2.filterDate('2024-01-01', '2024-05-31').median()
3 wet_composite = s2.filterDate('2024-06-01', '2024-11-30').median()
4
5 # Calculate NDVI for each
6 dry_ndvi = dry_composite.normalizedDifference(['B8', 'B4'])
7 wet_ndvi = wet_composite.normalizedDifference(['B8', 'B4'])
8
9 # Temporal difference
10 ndvi_diff = wet_ndvi.subtract(dry_ndvi)
```

Interpretation: - **Positive (>0.2):** Seasonal crops (rice) 🌾 - **Near zero (-0.1 to 0.1):** Evergreen forest 🌳 - **Negative (<-0.1):** Dry season crops, deciduous

Topographic Features

Why Add Elevation Data?









- **Altitude patterns:** Upland forest vs lowland agriculture
- **Slope:** Flat = agriculture, steep = forest (less accessible)
- **Aspect:** North-facing = more moisture = denser forest
- **Accessibility:** Low elevation near roads = higher deforestation risk

GEE Implementation:

```
1 # Load SRTM DEM
2 dem = ee.Image('USGS/SRTMGL1_003')
3
4 # Calculate derivatives
5 elevation = dem.select('elevation')
6 slope = ee.Terrain.slope(dem)
7 aspect = ee.Terrain.aspect(dem)
8
9 # Add to features
10 features = features.addBands([elevation, slope, aspect])
```


Palawan 8-Class Scheme

Classification Classes

Class	Description	Key Discriminators
 Primary Forest	Dense dipterocarp, closed canopy	High NDVI, low texture, high elevation
 Secondary Forest	Regenerating, mixed canopy	Moderate NDVI, medium texture
 Mangroves	Coastal, tidal	High NDVI + high NDWI + coastal
 Agricultural	Rice, coconut	Seasonal NDVI change, flat terrain
 Grassland	Open, sparse vegetation	Low-moderate NDVI, low texture
 Water	Rivers, lakes, coastal	Very low NIR, high NDWI
 Urban	Settlements, infrastructure	High NDBI, high texture, low NDVI
 Bare Soil	Mining, cleared	Bright, low NDVI, near roads

Feature Stack Summary

Complete Feature Set (~21 features):

- Spectral (6)** - B2 (Blue) - B3 (Green) - B4 (Red) - B8 (NIR) - B11 (SWIR1) - B12 (SWIR2)
- Indices (4)** - NDVI - NDWI - NDBI - EVI
- Texture (4)** - Contrast - Entropy - Correlation - Homogeneity
- Temporal (4)** - Dry NDVI - Wet NDVI - NDVI difference - Seasonal amplitude
- Topographic (3)** - Elevation - Slope - Aspect

Hyperparameter Optimization

Random Forest Parameters

Key Parameters to Tune:

Parameter	Default	Range to Test	Impact
<code>numberOfTrees</code>	100	50, 100, 200, 500	More = better (diminishing returns)
<code>variablesPerSplit</code>	\sqrt{n}	\sqrt{n} , $\log_2(n)$, $n/3$	Balance randomness vs accuracy
<code>minLeafPopulation</code>	1	1, 2, 5, 10	Higher = simpler trees
<code>bagFraction</code>	0.5	0.5, 0.7, 1.0	Sampling strategy

Cross-Validation Strategy

K-Fold Cross-Validation (k=5):

```
1 # Split training data into 5 folds
2 folds = training_data.randomColumn('fold', seed=42)
3
4 # Test each fold
5 accuracies = []
6 for i in range(5):
7     train = folds.filter(ee.Filter.neq('fold', i))
8     test = folds.filter(ee.Filter.eq('fold', i))
9
10    # Train model
11    classifier = ee.Classifier.smileRandomForest(100).train(train, 'class', bands)
12
13    # Evaluate
14    accuracy = test.classify(classifier).errorMatrix('class', 'classification').accuracy()
15    accuracies.append(accuracy)
16
17 # Average accuracy
18 mean_accuracy = sum(accuracies) / 5
```

Handling Class Imbalance

The Problem:

Real-world datasets often have imbalanced classes:

Class	Pixels	Percentage
Primary Forest	450,000	45%
Secondary Forest	200,000	20%
Agriculture	180,000	18%
Water	100,000	10%
Grassland	40,000	4%
Urban	15,000	1.5% ← Rare class
Mangroves	10,000	1% ← Very rare
Bare Soil	5,000	0.5%

Consequence: Model ignores rare classes, poor accuracy for minority classes

Class Imbalance Solutions

1. Balanced Sampling

Oversample rare classes, undersample common classes:

```
1 # Equal samples per class
2 samples_per_class = 100
3
4 balanced_training = ee.FeatureCollection([])
5 for class_id in [1, 2, 3, 4, 5, 6, 7, 8]:
6     class_samples = training.filter(
7         ee.Filter.eq('class', class_id)
8     ).randomColumn('random').limit(samples_per_class)
9     balanced_training = balanced_training.merge(class_samples)
```

Pros: Simple, effective **Cons:** May oversample noisy pixels

2. Class Weights

Give higher weight to rare classes during training:

```
1 # Calculate class weights inversely proportional to frequency
2 class_counts = {1: 450000, 2: 200000, ..., 7: 10000, 8: 5000}
3 total = sum(class_counts.values())
4 weights = {k: total/(len(class_counts)*v) for k,v in class_counts.items()}
5
6 # Not directly supported in GEE, but can weight samples
```

3. Stratified Validation

Ensure all classes in train AND test sets:

```
1 # Stratified split per class
2 training = ee.FeatureCollection([])
3 testing = ee.FeatureCollection([])
4
5 for class_id in [1, 2, 3, 4, 5, 6, 7, 8]:
6     class_data = all_data.filter(ee.Filter.eq('class', class_id))
7     class_data = class_data.randomColumn('random')
8
9     train = class_data.filter(ee.Filter.lt('random', 0.8))
10    test = class_data.filter(ee.Filter.gte('random', 0.8))
11
12    training = training.merge(train)
13    testing = testing.merge(test)
```

Out-of-Bag (OOB) Error

Built-in Validation:

Random Forest automatically provides OOB error estimate

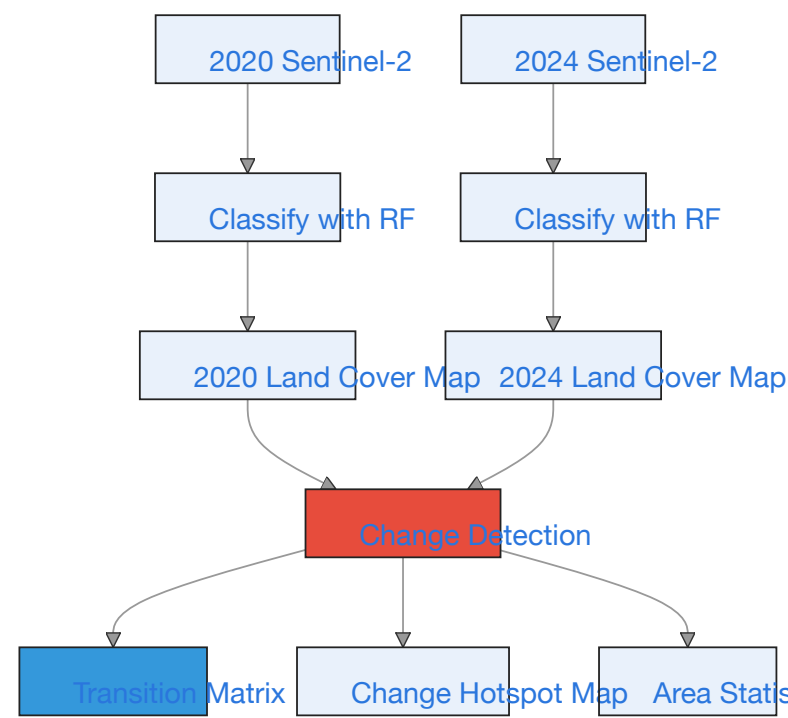
```
1 # Train with OOB
2 classifier = ee.Classifier.smileRandomForest(
3     numberOfTrees=100,
4     outOfBagMode=True # Enable OOB error calculation
5 ).train(training, 'class', bands)
6
7 # Get OOB error
8 oob_error = classifier.confusionMatrix().accuracy()
9 print('OOB Accuracy:', oob_error)
```

Advantage: No need for separate validation set (~37% of data used for OOB)

Change Detection

2020 vs 2024 Comparison

Deforestation Analysis Workflow:



Change Detection Implementation

```
1 # Classify both years with same model
2 lc_2020 = composite_2020.classify(trained_classifier)
3 lc_2024 = composite_2024.classify(trained_classifier)
4
5 # Detect changes
6 change = lc_2024.subtract(lc_2020)
7
8 # Forest loss (class 1 or 2 → any other class)
9 forest_2020 = lc_2020.lte(2) # Primary or secondary
10 forest_2024 = lc_2024.lte(2)
11 forest_loss = forest_2020.And(forest_2024.Not())
12
13 # Agricultural expansion
14 ag_gain = lc_2020.neq(4).And(lc_2024.eq(4))
15
16 # Calculate areas
17 forest_loss_area = forest_loss.multiply(ee.Image.pixelArea()).reduceRegion({
18     reducer: ee.Reducer.sum(),
```

Transition Matrix

From-To Analysis:

	→ Forest	→ Ag	→ Urban	→ Bare
Forest ↓	85%	12%	2%	1%
Ag ↓	3%	90%	5%	2%
Grassland ↓	8%	25%	60%	7%
Bare ↓	2%	10%	5%	83%

Key Insights: - 12% forest → agriculture (main driver) - 5% agriculture → urban (development) - Grassland mostly stable or converts to agriculture

Post-Processing Techniques

Why Post-Process Classifications?

Raw Classification Issues:

Common Problems: - **Salt-and-pepper noise:** Isolated misclassified pixels - **Small patches:** Below minimum mapping unit - **Edge effects:** Mixed pixels at boundaries - **Geometric errors:** Irregular shapes

Solutions: 1. Majority/Modal filter 2. Minimum mapping unit filter 3. Morphological operations 4. Boundary smoothing

Goal: Clean, cartographically appealing maps suitable for stakeholder communication

Majority Filter (Focal Mode)

Smooth noisy pixels using neighborhood majority:

```
1 # Apply 3x3 majority filter
2 classification = classification_raw.focal_mode(
3     radius=1, # 3x3 window (1 pixel in each direction)
4     kernelType='square'
5 )
6
7 # More aggressive: 5x5 filter
8 classification_smooth = classification_raw.focal_mode(
9     radius=2, # 5x5 window
10    kernelType='square'
11 )
```

Before Filtering: - Salt-and-pepper noise - Isolated pixels
- Fragmented patches

After Filtering: - Smoother appearance - More
contiguous patches - Reduced noise

Trade-off: May lose small but real features (small clearings, narrow roads)

Minimum Mapping Unit (MMU)

Remove patches smaller than threshold:

```
1 # Step 1: Connected component labeling
2 connected = classification.connectedPixelCount(maxSize=256)
3
4 # Step 2: Filter by size (e.g., MMU = 25 pixels = 0.25 ha at 10m resolution)
5 mmu_threshold = 25
6 classification_mmu = classification.updateMask(connected.gte(mmu_threshold))
7
8 # Step 3: Fill gaps with focal_mode
9 classification_clean = classification_mmu.focal_mode(radius=5, kernelType='square')
```

MMU Guidelines:

Local (1:10,000)	0.1 - 0.5 ha	Detailed management
Regional (1:50,000)	1 - 5 ha	Provincial planning
National (1:250,000)	10 - 25 ha	National reporting

Morphological Operations

Opening and Closing for shape refinement:

Opening (Erosion → Dilation):

Removes small protrusions, separates narrow connections

```
1 # Erode then dilate
2 eroded = classification.focal_min(radius=1)
3 opened = eroded.focal_max(radius=1)
```

Use for: - Breaking thin connections - Removing speckles
- Smoothing boundaries

Combined Workflow:

```
1 # Remove noise (opening) then fill gaps (closing)
2 classification_clean = classification.focal_min(1).focal_max(1) # Opening
3 classification_final = classification_clean.focal_max(1).focal_min(1) # Closing
```

Closing (Dilation → Erosion):

Fills small holes, connects nearby patches

```
1 # Dilate then erode
2 dilated = classification.focal_max(radius=1)
3 closed = dilated.focal_min(radius=1)
```

Use for: - Filling gaps - Joining nearby patches - Closing boundaries

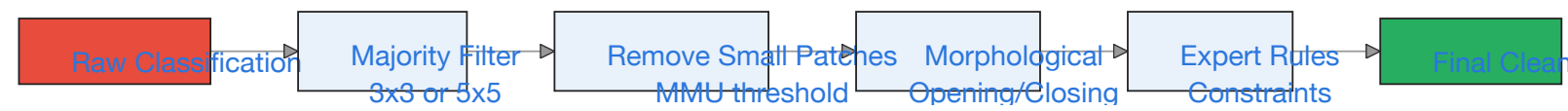
Expert Rules & Constraints

Apply domain knowledge to refine results:

```
1 # Rule 1: Mangroves only near coast
2 mangrove_class = 3
3 distance_to_coast = coastline.distance(maxDistance=5000) # 5km
4 mangrove_mask = classification.eq(mangrove_class).And(distance_to_coast.lt(2000))
5 classification = classification.where(
6     classification.eq(mangrove_class).And(distance_to_coast.gte(2000)),
7     2 # Reclassify as secondary forest
8 )
9
10 # Rule 2: Agriculture unlikely above 1000m elevation
11 ag_class = 4
12 classification = classification.where(
13     classification.eq(ag_class).And(elevation.gt(1000)),
14     5 # Reclassify as grassland
15 )
16
17 # Rule 3: Urban near roads
18 urban_class = 6
```

Post-Processing Workflow Summary

Recommended Pipeline:



Quality Control Checklist: - ✓ Visual inspection of before/after - ✓ Area statistics comparison (should be similar) - ✓ Check for over-smoothing - ✓ Validate against high-resolution imagery - ✓ Ensure ecologically plausible results

Protected Area Monitoring

Palawan Conservation Context

UNESCO Biosphere Reserve (1990)

Biodiversity: - 252 bird species (15 endemic) - 95 mammal species - Last Philippine frontier forest - Critical habitat for endangered species

Area: - Core: 3,000 km² - Buffer: 5,000 km² - Transition: 3,655 km²

Threats: - Mining (nickel, chromite) - Agricultural expansion - Infrastructure (roads, ports) - Illegal logging - Tourism pressure

Management: - DENR oversight - Strategic Environmental Plan (SEP) - Local government units (LGUs) - NGO partnerships

Encroachment Detection

Boundary Analysis:

```
1 # Load protected area boundary
2 protected_area = ee.FeatureCollection('path/to/palawan_PA')
3
4 # Create buffer zones
5 core = protected_area
6 buffer_1km = core.buffer(1000)
7 buffer_5km = core.buffer(5000)
8
9 # Detect forest loss in each zone
10 core_loss = forest_loss.clip(core)
11 buffer_loss = forest_loss.clip(buffer_5km).subtract(core_loss)
12
13 # Calculate statistics
14 core_loss_area = core_loss.multiply(ee.Image.pixelArea()).reduceRegion(...)
15 buffer_loss_area = buffer_loss.multiply(ee.Image.pixelArea()).reduceRegion(...)
16
17 # Generate alert if threshold exceeded
18 if core_loss_area > threshold:
```

Deforestation Hotspot Map

Kernel Density Analysis:

```
1 # Identify forest loss pixels
2 loss_pixels = forest_loss.selfMask()
3
4 # Convert to points
5 loss_points = loss_pixels.sample(
6     region=aoi,
7     scale=10,
8     geometries=True
9 )
10
11 # Kernel density estimation
12 hotspots = loss_points.reduceToImage(['classification'],
13                                     ee.Reducer.count())
14                                     .convolve(ee.Kernel.gaussian(500))
15
16 # Visualize
17 Map.addLayer(hotspots, {min: 0, max: 50, palette: ['white', 'yellow', 'red']},
18               'Deforestation Hotspots')
```

Use Case: Target field verification and enforcement

Expected Outcomes

Performance Targets

Accuracy Improvement:

Approach	Overall Accuracy	Kappa
Session 1 (Basic RF)	82%	0.78
+ GLCM Texture	85%	0.82
+ Multi-temporal	87%	0.84
+ Topographic	88-90%	0.86-0.88

Per-Class Targets: >85% for most classes

Common Confusions

Expected Confusion Pairs:

1. **Primary** ↔ **Secondary Forest**

- Similar spectral signature
- Texture helps but overlap exists
- Solution: Add canopy height (LiDAR)

2. **Mangrove** ↔ **Wet Season Agriculture**

- Both high NDVI + water proximity
- Solution: Temporal analysis (mangroves stable)

3. **Urban** ↔ **Bare Soil**

- Both bright in visible bands
- Solution: Texture (urban more heterogeneous)

Session Deliverables

By the end of this session:

- ✓ High-resolution land cover map (10m Palawan)
- ✓ Comprehensive accuracy report (>85%)
- ✓ Feature importance analysis
- ✓ 2020-2024 change detection map
- ✓ Deforestation statistics (hectares per class)
- ✓ Hotspot map for DENR
- ✓ Exported GeoTIFF for QGIS integration
- ✓ Area statistics CSV

NRM Applications

DENR Use Cases

Forest Monitoring: - Annual forest cover updates - REDD+ MRV compliance - Protected area assessment - Illegal logging detection

Implementation: - Automated monthly processing - Alert system for >5 ha forest loss - Integration with field teams - Reporting to central office

Local Government Applications

Land Use Planning: - Zoning map updates - Infrastructure siting (avoid sensitive areas) - Agricultural zone delineation - Tourism planning

Disaster Risk: - Flood-prone areas (based on land cover) - Landslide susceptibility (slope + forest loss) - Evacuation route planning

NGO Conservation Programs

Community Monitoring: - Train local rangers to use maps - Mobile app for ground truthing - Participatory mapping sessions - Livelihood integration (agroforestry zones)

Impact Assessment: - Baseline for conservation projects - Monitor restoration success - Detect encroachment early - Evidence for advocacy

Technical Considerations

Computational Challenges

GEE Limitations:

 Common Errors

“**Computation timed out**” - **Cause:** GLCM on large area - **Solution:** Process in tiles, export intermediates

“**Memory limit exceeded**” - **Cause:** Too many features + large AOI - **Solution:** Reduce feature count, use `.aside()` sparingly

“**User memory limit exceeded**” - **Cause:** Complex reducers - **Solution:** Simplify, use `.limit()` on collections

Optimization Strategies

Speed Up Processing:

1. **Use `.limit()`** on ImageCollections
2. **Export intermediate results** (composites, features)
3. **Reduce GLCM window** ($5 \times 5 \rightarrow 3 \times 3$)
4. **Process by tiles** for very large areas
5. **Use `.aside()`** judiciously for debugging

Example:

```
1 # Instead of:
2 composite = collection.median() # Slow
3
4 # Do:
5 composite = collection.limit(50).median() # Faster, usually sufficient
```


Summary

Key Takeaways

1. **Feature Engineering Matters:** +5-8% accuracy from texture, temporal, topographic
2. **Multi-temporal is Powerful:** Seasonal patterns reveal agriculture vs forest
3. **GLCM Adds Context:** Spatial structure complements spectral info
4. **Hyperparameter Tuning:** Small gains but worth it for production
5. **Change Detection:** Quantifies deforestation for stakeholders
6. **Real-world Impact:** This workflow used by DENR, PhilSA, NGOs

Next Steps

After Session 2

Immediate: - Complete all notebook exercises - Experiment with feature combinations - Try different AOIs in Philippines

Session 3 Preview: Deep Learning and CNNs - automatic feature learning!

[Continue to Session 3 →](#)

Questions & Resources

Documentation: - [GEE GLCM](#) - [RF Classifier](#) - [Change Detection Guide](#)

Support: - Instructor Q&A - [GEE Forum](#) - Session notebook with complete code

Session 2 - Advanced Palawan Land Cover Lab | CoPhil Training Programme