03_verify_patches

December 29, 2025

1 Verify Preprocessed Scenes

Sanity check that scene conversion is correct and random cropping works.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path, PureWindowsPath, PurePosixPath
import random
import json
import torch
import platform

# Detect environment and set path accordingly
if platform.system() == 'Windows':
    # Running from Windows Python accessing WSL files
    SCENE_DIR = Path(r'\\wsl.localhost\Ubuntu-24.
↪04\home\mike1\Arcticv2\data\ai4arctic_hugging face\npy_memmap')
else:
    # Running from WSL/Linux
    SCENE_DIR = Path('../data/ai4arctic_hugging face/npy_memmap')
    if not SCENE_DIR.exists():
        SCENE_DIR = Path('/home/mike1/Arcticv2/data/ai4arctic_hugging face/
↪npy_memmap')

print(f"Platform: {platform.system()}")
print(f"Scene directory: {SCENE_DIR}")
print(f"Directory exists: {SCENE_DIR.exists()}")

# Class names
CLASS_NAMES = [
    'OpenWater',
    'NewIce',
    'YoungIce',
    'FirstYearIce',
    'MultiYearIce',
    'GlacialIce',
]
```

```
# Colors for each class
COLORS = [
    [0, 0, 139],       # OpenWater - dark blue
    [173, 216, 230],  # NewIce - light blue
    [144, 238, 144],  # YoungIce - light green
    [255, 255, 0],     # FirstYearIce - yellow
    [255, 165, 0],     # MultiYearIce - orange
    [255, 255, 255],  # GlacialIce - white
]
COLORS = np.array(COLORS) / 255.0
```

```
Platform: Linux
Scene directory: ../data/ai4arctic_hugging face/npy_memmap
Directory exists: True
```

[2]:
```
# Check preprocessing output (memmap format: separate _image.npy and _label.npy␣
↪files)
train_images = list((SCENE_DIR / 'train').glob('*_image.npy'))
val_images = list((SCENE_DIR / 'val').glob('*_image.npy'))

print(f'Train scenes: {len(train_images)}')
print(f'Val scenes: {len(val_images)}')

# Load metadata
if (SCENE_DIR / 'metadata.json').exists():
    with open(SCENE_DIR / 'metadata.json') as f:
        metadata = json.load(f)
    print(f"\nMetadata:")
    print(f"  Version: {metadata.get('version', 'N/A')}")
    print(f"  Classes: {metadata.get('num_classes', 'N/A')}")
    print(f"  Train scenes: {metadata.get('num_train_scenes', 'N/A')}")
    print(f"  Val scenes: {metadata.get('num_val_scenes', 'N/A')}")

# Load normalization stats
if (SCENE_DIR / 'normalization_stats.json').exists():
    with open(SCENE_DIR / 'normalization_stats.json') as f:
        stats = json.load(f)
    print(f"\nNormalization stats:")
    for i, name in enumerate(stats['channel_names']):
        print(f"  {name}: mean={stats['mean'][i]:.4f}, std={stats['std'][i]:.
↪4f}")

# Load coordinate counts
if (SCENE_DIR / 'train_coords.npy').exists():
    train_coords = np.load(SCENE_DIR / 'train_coords.npy', allow_pickle=True).
↪item()
    print(f"\nTrain coords: {len(train_coords['coords'])} crops")
```

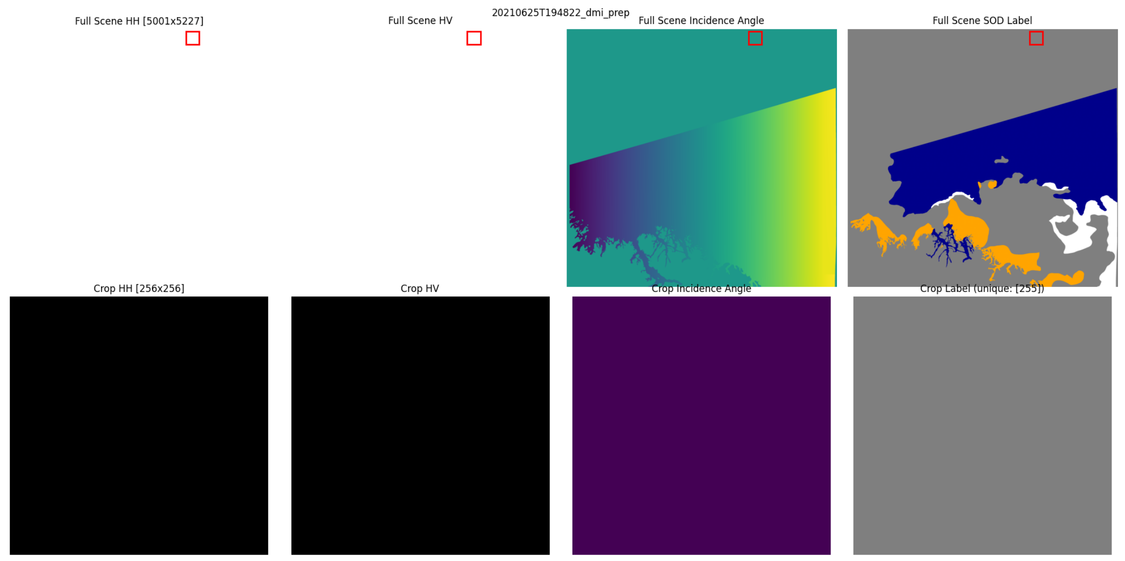
```python
if (SCENE_DIR / 'val_coords.npy').exists():
    val_coords = np.load(SCENE_DIR / 'val_coords.npy', allow_pickle=True).item()
    print(f"Val coords: {len(val_coords['coords'])} crops")
```

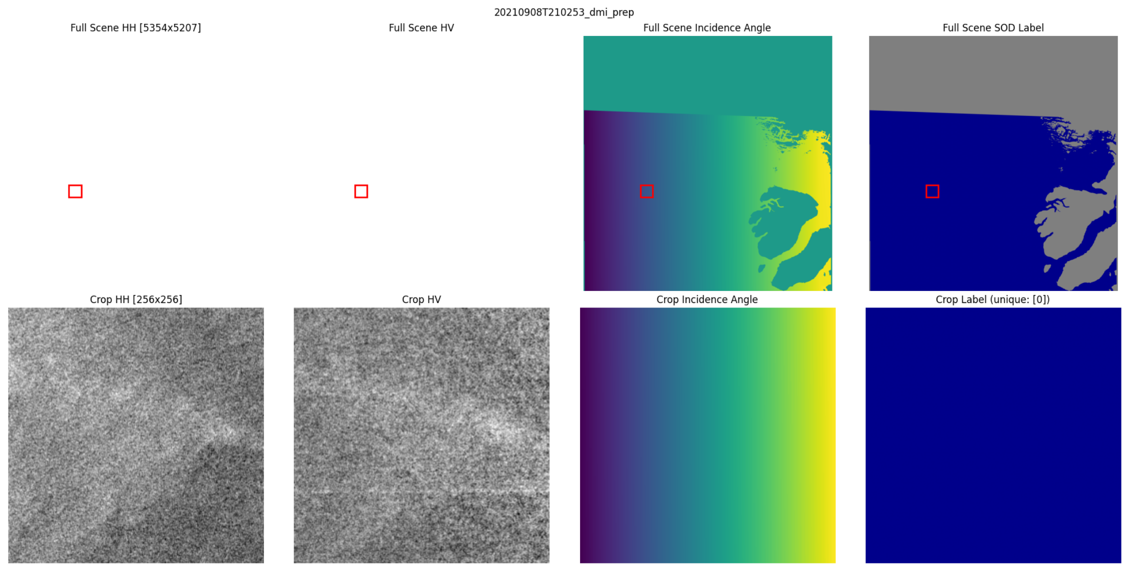
```
Train scenes: 435
Val scenes: 77

Metadata:
  Version: v5_memmap_optimized
  Classes: 6
  Train scenes: N/A
  Val scenes: N/A

Normalization stats:
  HH: mean=-0.1533, std=0.7334
  HV: mean=-0.2210, std=0.4487
  IncidenceAngle: mean=-0.0729, std=0.7055

Train coords: 91203 crops
Val coords: 13335 crops
```

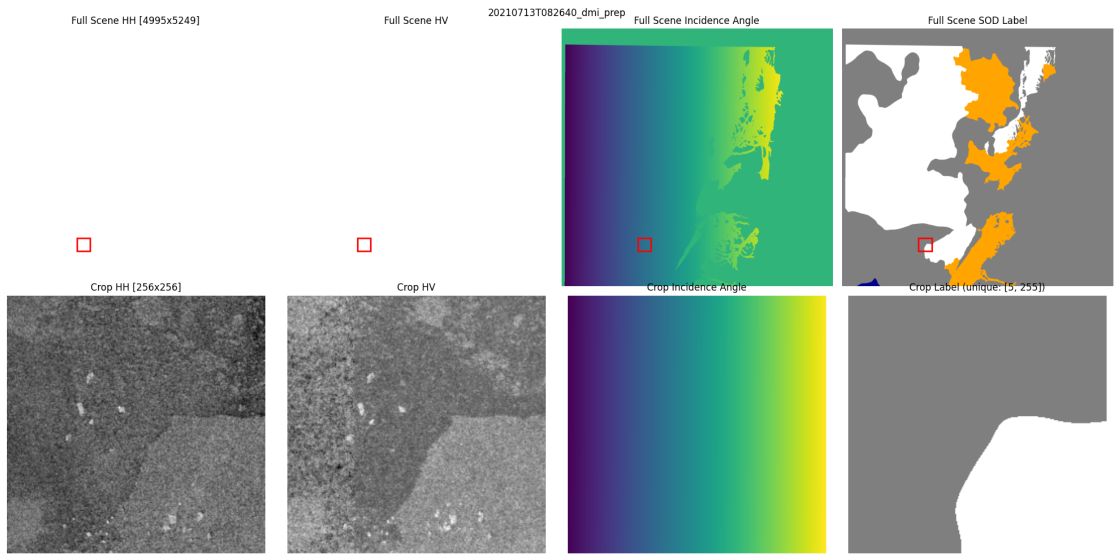
[3]:
```python
def load_scene(image_path):
    """Load a scene from memmap format (separate _image.npy and _label.npy␣
↪files)."""
    label_path = str(image_path).replace('_image.npy', '_label.npy')
    image = np.load(image_path, mmap_mode='r')  # [3, H, W] float16
    label = np.load(label_path, mmap_mode='r')  # [H, W] uint8
    return image, label

def label_to_rgb(label):
    """Convert label to RGB image for visualization."""
    H, W = label.shape
    rgb = np.zeros((H, W, 3))

    for cls_idx, color in enumerate(COLORS):
        mask = label == cls_idx
        rgb[mask] = color

    # Mark ignore pixels (255) as gray
    ignore_mask = label == 255
    rgb[ignore_mask] = [0.5, 0.5, 0.5]

    return rgb

def visualize_scene(image_path, crop_size=256):
    """Visualize a full scene with a random crop highlighted."""
    image, label = load_scene(image_path)
    H, W = label.shape

    # Pick a random crop location
    r = random.randint(0, max(0, H - crop_size))
    c = random.randint(0, max(0, W - crop_size))

    fig, axes = plt.subplots(2, 4, figsize=(20, 10))

    # Top row: Full scene
    axes[0, 0].imshow(image[0], cmap='gray', vmin=np.nanpercentile(image[0],␣
↪1), vmax=np.nanpercentile(image[0], 99))
    axes[0, 0].set_title(f'Full Scene HH [{image.shape[1]}x{image.shape[2]}]')
    axes[0, 0].add_patch(plt.Rectangle((c, r), crop_size, crop_size,␣
↪fill=False, edgecolor='red', linewidth=2))

    axes[0, 1].imshow(image[1], cmap='gray', vmin=np.nanpercentile(image[1],␣
↪1), vmax=np.nanpercentile(image[1], 99))
    axes[0, 1].set_title('Full Scene HV')
    axes[0, 1].add_patch(plt.Rectangle((c, r), crop_size, crop_size,␣
↪fill=False, edgecolor='red', linewidth=2))

    axes[0, 2].imshow(image[2], cmap='viridis')
    axes[0, 2].set_title('Full Scene Incidence Angle')
    axes[0, 2].add_patch(plt.Rectangle((c, r), crop_size, crop_size,␣
↪fill=False, edgecolor='red', linewidth=2))

    label_rgb = label_to_rgb(label)
    axes[0, 3].imshow(label_rgb)
    axes[0, 3].set_title('Full Scene SOD Label')
    axes[0, 3].add_patch(plt.Rectangle((c, r), crop_size, crop_size,␣
↪fill=False, edgecolor='red', linewidth=2))

    # Bottom row: Crop (what training sees)
    crop_img = image[:, r:r+crop_size, c:c+crop_size]
    crop_lbl = label[r:r+crop_size, c:c+crop_size]

    axes[1, 0].imshow(crop_img[0], cmap='gray')
    axes[1, 0].set_title(f'Crop HH [{crop_size}x{crop_size}]')

    axes[1, 1].imshow(crop_img[1], cmap='gray')
    axes[1, 1].set_title('Crop HV')

    axes[1, 2].imshow(crop_img[2], cmap='viridis')
    axes[1, 2].set_title('Crop Incidence Angle')

    crop_rgb = label_to_rgb(crop_lbl)
    axes[1, 3].imshow(crop_rgb)
    axes[1, 3].set_title(f'Crop Label (unique: {np.unique(crop_lbl).tolist()})')

    for ax in axes.flat:
        ax.axis('off')

    plt.suptitle(image_path.name.replace('_image.npy', ''), fontsize=12)
    plt.tight_layout()
    plt.show()
```

[4]:
```
# Visualize 3 random train scenes
print('=== TRAIN SCENES ===')
for scene_path in random.sample(train_images, min(3, len(train_images))):
    visualize_scene(scene_path)
```

```
=== TRAIN SCENES ===

/home/mike1/Arcticv2/.venv/lib/python3.13/site-
packages/numpy/lib/_function_base_impl.py:132: RuntimeWarning: overflow
encountered in cast
  'get_virtual_index': lambda n, quantiles: (n - 1) * quantiles,
/home/mike1/Arcticv2/.venv/lib/python3.13/site-
packages/numpy/lib/_function_base_impl.py:4687: RuntimeWarning: overflow
encountered in cast
  indexes_above_bounds = virtual_indexes >= valid_values_count - 1
/home/mike1/Arcticv2/.venv/lib/python3.13/site-
packages/numpy/lib/_function_base_impl.py:4593: RuntimeWarning: invalid value
encountered in multiply
  lerp_interpolation = add(a, diff_b_a * t, out=… if out is None else out)
/home/mike1/Arcticv2/.venv/lib/python3.13/site-
packages/numpy/lib/_function_base_impl.py:4594: RuntimeWarning: invalid value
encountered in scalar multiply
  subtract(b, diff_b_a * (1 - t), out=lerp_interpolation, where=t >= 0.5,
```

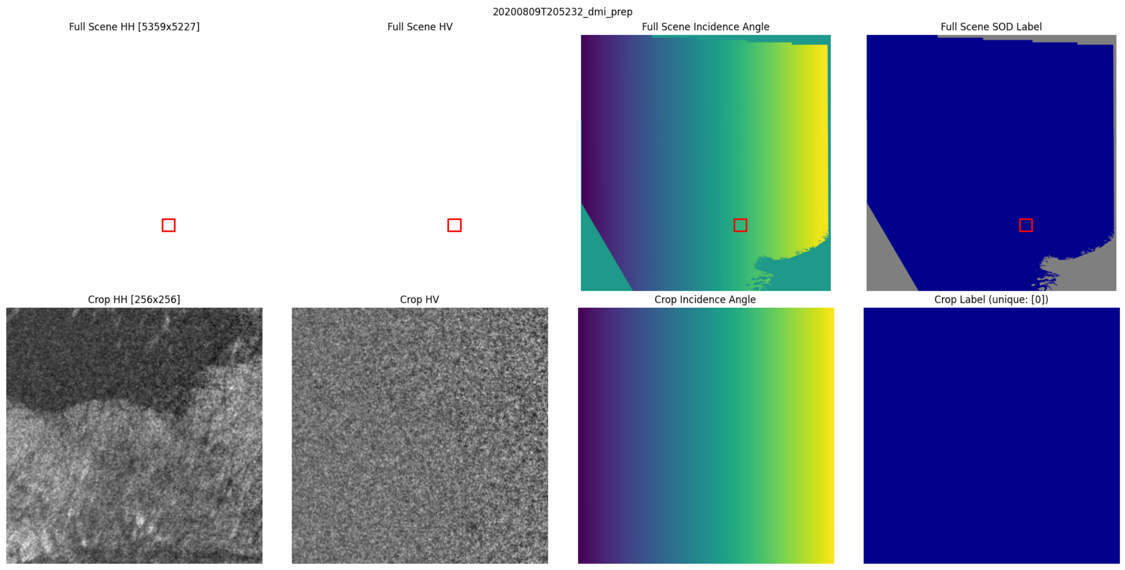

20210625T194822_dmi_prep
Full Scene HH [5001x5227]
Full Scene HV
Full Scene Incidence Angle
Full Scene SOD Label
Crop HH [256x256]
Crop HV
Crop Incidence Angle
Crop Label (unique: [255])

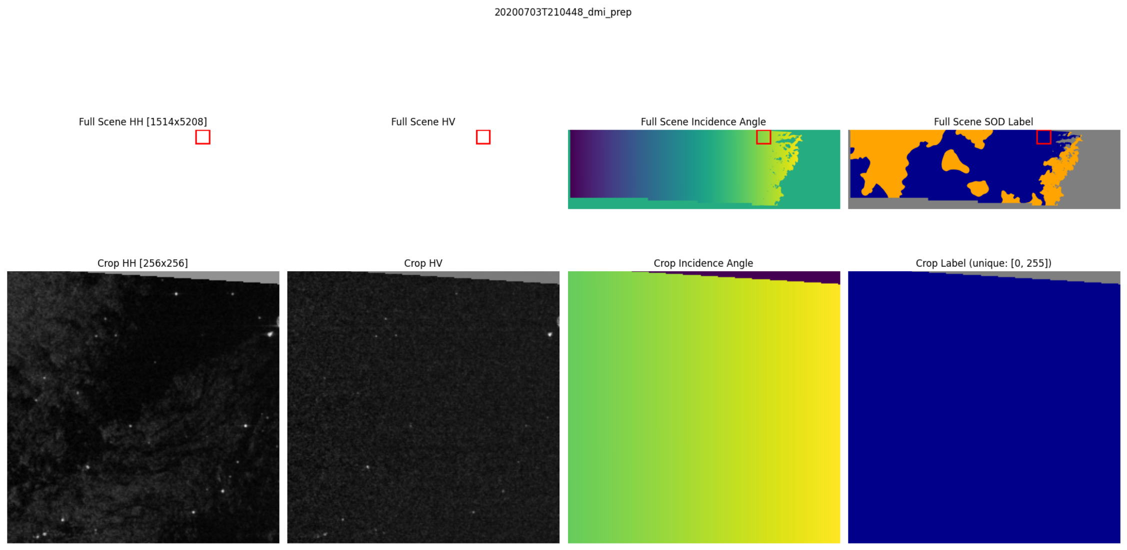

20210908T210253_dmi_prep
Full Scene HH [5354x5207]
Full Scene HV
Full Scene Incidence Angle
Full Scene SOD Label
Crop HH [256x256]
Crop HV
Crop Incidence Angle
Crop Label (unique: [0])



```
[5]: # Visualize 2 random val scenes
     print('=== VAL SCENES ===')
     for scene_path in random.sample(val_images, min(2, len(val_images))):
         visualize_scene(scene_path)
```

```
=== VAL SCENES ===
```





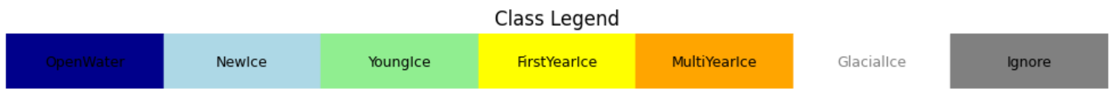
[6]:
```
# Check scene sizes and class distribution
print('\n=== SCENE STATISTICS ===')

heights, widths = [], []
class_pixels = {i: 0 for i in range(6)}
ignore_pixels = 0
total_pixels = 0

# Sample scenes for distribution
sample_scenes = random.sample(train_images, min(50, len(train_images)))

for path in sample_scenes:
    _, label = load_scene(path)
    heights.append(label.shape[0])
    widths.append(label.shape[1])

    for cls_idx in range(6):
        class_pixels[cls_idx] += (label == cls_idx).sum()
    ignore_pixels += (label == 255).sum()
    total_pixels += label.size

print(f'\nScene dimensions (from {len(sample_scenes)} scenes):')
print(f'  Height: min={min(heights)}, max={max(heights)}, mean={np.
 ↪mean(heights):.0f}')
print(f'  Width:  min={min(widths)}, max={max(widths)}, mean={np.mean(widths):.
 ↪0f}')

print(f'\nPixel distribution:')
```

```
for cls_idx, count in class_pixels.items():
    pct = count / total_pixels * 100
    print(f'  {CLASS_NAMES[cls_idx]}: {pct:.2f}%')
print(f'  Ignore (255): {ignore_pixels / total_pixels * 100:.2f}%')
```

```
=== SCENE STATISTICS ===

Scene dimensions (from 50 scenes):
  Height: min=1262, max=5360, mean=4922
  Width:  min=5171, max=5361, mean=5249

Pixel distribution:
  OpenWater: 30.32%
  NewIce: 0.28%
  YoungIce: 0.54%
  FirstYearIce: 1.65%
  MultiYearIce: 11.38%
  GlacialIce: 8.30%
  Ignore (255): 47.52%
```

[7]:
```
# Legend
fig, ax = plt.subplots(figsize=(10, 1))
for i, (name, color) in enumerate(zip(CLASS_NAMES, COLORS)):
    ax.add_patch(plt.Rectangle((i, 0), 1, 1, color=color, edgecolor='black'))
    ax.text(i + 0.5, 0.5, name, ha='center', va='center', fontsize=9,
            color='black' if i != 5 else 'gray')  # GlacialIce is white, use␣
 ↪gray text
ax.add_patch(plt.Rectangle((6, 0), 1, 1, color=[0.5, 0.5, 0.5],␣
 ↪edgecolor='black'))
ax.text(6.5, 0.5, 'Ignore', ha='center', va='center', fontsize=9)
ax.set_xlim(0, 7)
ax.set_ylim(0, 1)
ax.axis('off')
ax.set_title('Class Legend')
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_306868/3045762767.py:4: UserWarning: Setting the 'color' property
will override the edgecolor or facecolor properties.
  ax.add_patch(plt.Rectangle((i, 0), 1, 1, color=color, edgecolor='black'))
/tmp/ipykernel_306868/3045762767.py:7: UserWarning: Setting the 'color' property
will override the edgecolor or facecolor properties.
  ax.add_patch(plt.Rectangle((6, 0), 1, 1, color=[0.5, 0.5, 0.5],
edgecolor='black'))
```

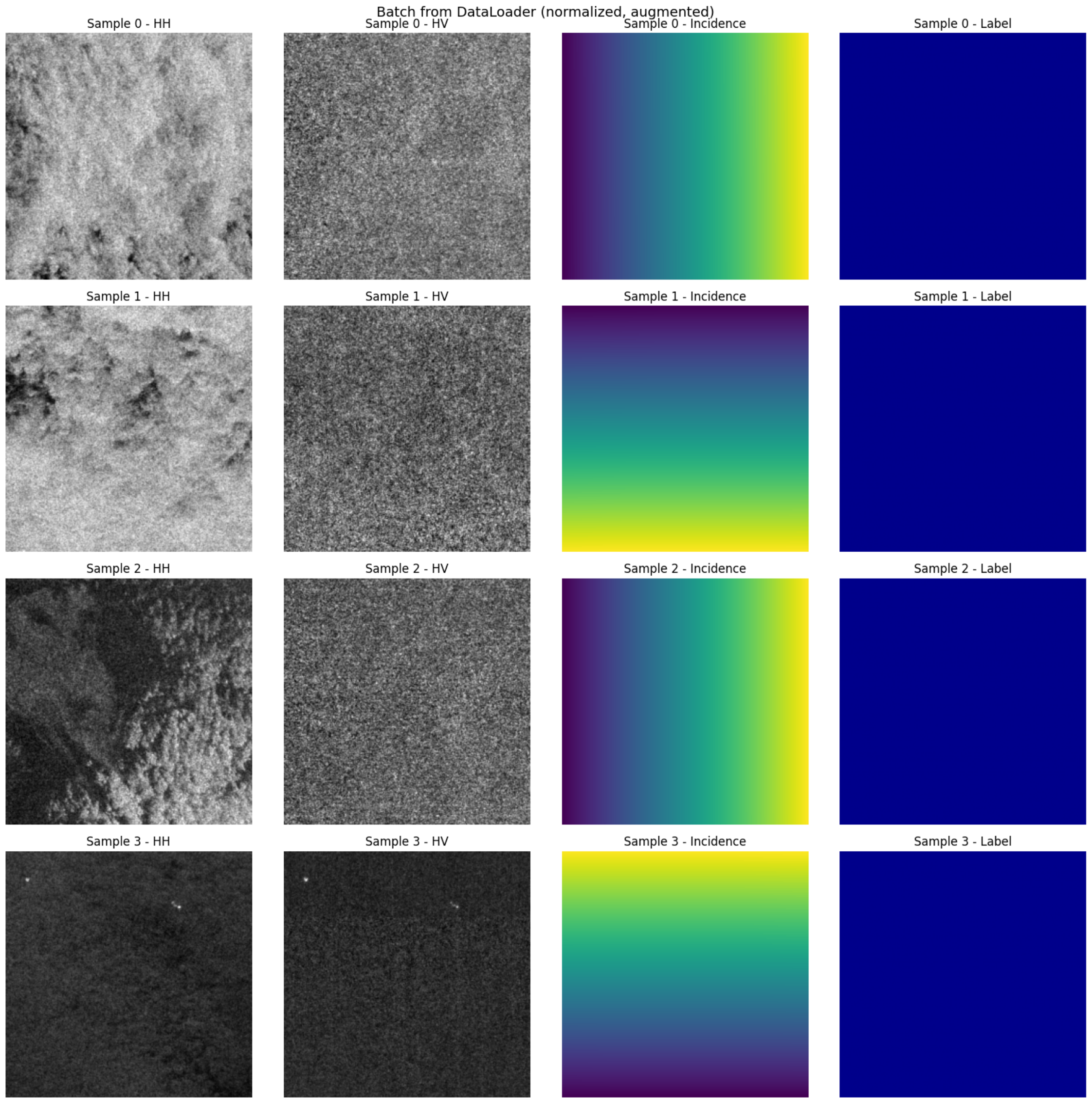


[8]:
```python
# Test the dataloader
import sys
import platform

if platform.system() == 'Windows':
    sys.path.insert(0, r'\\wsl.localhost\Ubuntu-24.04\home\mike1\Arcticv2\src')
    DATA_DIR = r'\\wsl.localhost\Ubuntu-24.
↪04\home\mike1\Arcticv2\data\ai4arctic_hugging face'
else:
    sys.path.insert(0, '../src')
    DATA_DIR = '../data/ai4arctic_hugging face'

from dataset import get_dataloaders

train_loader, val_loader = get_dataloaders(
    data_dir=DATA_DIR,
    batch_size=4,
    num_workers=0,
    crop_size=256,
    seed=42,
)

print(f'Train batches: {len(train_loader)}')
print(f'Val batches: {len(val_loader)}')
```

```
Using memmap data: ../data/ai4arctic_hugging face/npy_memmap
  Train: 91,203 crops
  Val:   13,335 crops
[train] 91,203 crops, 435 scenes
[val] 13,335 crops, 77 scenes
Train batches: 22800
Val batches: 3334
```

[9]:
```python
# Visualize a batch from dataloader
batch = next(iter(train_loader))
images = batch['image']  # [B, 3, 256, 256]
labels = batch['label']  # [B, 256, 256]

print(f'Batch image shape: {images.shape}')
print(f'Batch label shape: {labels.shape}')
print(f'Image range: [{images.min():.2f}, {images.max():.2f}]')
print(f'Label unique values: {torch.unique(labels).tolist()}')

fig, axes = plt.subplots(4, 4, figsize=(16, 16))
for i in range(4):
    # HH
    axes[i, 0].imshow(images[i, 0].numpy(), cmap='gray')
    axes[i, 0].set_title(f'Sample {i} - HH')
    axes[i, 0].axis('off')

    # HV
    axes[i, 1].imshow(images[i, 1].numpy(), cmap='gray')
    axes[i, 1].set_title(f'Sample {i} - HV')
    axes[i, 1].axis('off')

    # Incidence
    axes[i, 2].imshow(images[i, 2].numpy(), cmap='viridis')
    axes[i, 2].set_title(f'Sample {i} - Incidence')
    axes[i, 2].axis('off')

    # Label (convert -100 back to 255 for visualization)
    lbl = labels[i].numpy().copy()
    lbl[lbl == -100] = 255
    axes[i, 3].imshow(label_to_rgb(lbl.astype(np.uint8)))
    axes[i, 3].set_title(f'Sample {i} - Label')
    axes[i, 3].axis('off')

plt.suptitle('Batch from DataLoader (normalized, augmented)', fontsize=14)
plt.tight_layout()
plt.show()
```

```
Batch image shape: torch.Size([4, 3, 256, 256])
Batch label shape: torch.Size([4, 256, 256])
Image range: [-3.67, 4.44]
Label unique values: [0]
```


Batch from DataLoader (normalized, augmented)
Sample 0 - HH
Sample 0 - HV
Sample 0 - Incidence
Sample 0 - Label
Sample 1 - HH
Sample 1 - HV
Sample 1 - Incidence
Sample 1 - Label
Sample 2 - HH
Sample 2 - HV
Sample 2 - Incidence
Sample 2 - Label
Sample 3 - HH
Sample 3 - HV
Sample 3 - Incidence
Sample 3 - Label