Normalization Utilities (normalization_utils.py)

This file handles the **Preprocessing** (preparing images for the AI) and **Post-processing** (preparing AI outputs for visualization). It ensures the neural network receives data in a standard mathematical distribution.

1. Imports and Constants

```
import numpy as np
```

- **Line:** Imports the NumPy library.
- **Why:** We need this for efficient array operations (matrices), specifically for handling image data which are stored as 3D arrays (Height, Width, Channels).

```
# --- SECOND Dataset Constants ---
# Calculated from the training set of the SECOND dataset
SECOND_MEAN_A = np.array([113.40, 114.08, 116.45])
SECOND_STD_A  = np.array([48.30,  46.27,  48.14])

SECOND_MEAN_B = np.array([111.07, 114.04, 118.18])
SECOND_STD_B  = np.array([49.41,  47.01,  47.94])
```

- **Lines:** define specific statistics (Mean and Standard Deviation) for the **SECOND** dataset.
- **Why A vs B?** In change detection, "A" is the *Pre-change* image and "B" is the *Post-change* image. These images might be taken in different seasons (e.g., Winter vs Summer).

- **Mean:** The average brightness of the Red, Green, and Blue channels.
- **Std:** How much the contrast varies.
- **Concept:** We treat A and B differently so the network sees them "normalized" regardless of seasonal lighting differences.

```
# --- LandsatSCD Dataset Constants ---
LANDSAT_MEAN_A = np.array([141.53, 139.20, 137.73])
...
LANDSAT_STD_B  = np.array([85.97,  86.01,  86.81])
```

- **Lines:** Similar statistics but for the **LandsatSCD** dataset.
- **Observation:** Notice that Landsat values are generally higher (brighter) and have higher standard deviation (more contrast/variance) than SECOND. This is why we need dataset-specific normalization; using SECOND stats on Landsat data would result in incorrect inputs.

2. The Selector Function (`get_constants`)

```
def get_constants(dataset_name, time_step):
```

- **Line:** Defines a helper function to automatically pick the right numbers.
- **Inputs:**
  - `dataset_name`: e.g., "Second_dataset".
  - `time_step`: "A" (Before) or "B" (After).

```
    d_name = dataset_name.lower()
    time = time_step.upper()
```

- **Lines:** Standardizes input strings.
- **Why:** Programmers make typos. `lower()` ensures "Second", "SECOND", and "second" all work. `upper()` ensures "a" becomes "A".

```
    if 'second' in d_name:
        if time == 'A': return SECOND_MEAN_A, SECOND_STD_A
        if time == 'B': return SECOND_MEAN_B, SECOND_STD_B

    elif 'landsat' in d_name:
        if time == 'A': return LANDSAT_MEAN_A, LANDSAT_STD_A
        if time == 'B': return LANDSAT_MEAN_B, LANDSAT_STD_B
```

- **Lines:** Checks which dataset key is in the name and returns the corresponding global constants we defined earlier.

```
    print(f"Warning: Unknown dataset '{dataset_name}'. Using standard ImageNet normalization.")
    imagenet_mean = np.array([123.675, 116.28, 103.53])
    imagenet_std = np.array([58.395, 57.12, 57.375])
    return imagenet_mean, imagenet_std
```

- **Lines:** The **Fallback Mechanism**.

- **Logic:** If the user provides a dataset name we don't know (e.g., "GoogleEarth"), we default to **ImageNet** statistics.
- **Math:** Standard ImageNet mean is `[0.485, 0.456, 0.406]`. Since our images are 0-255 (not 0-1), we multiply by 255 to get `~123.675`.
- **Why:** Most Deep Learning backbones (ResNet, VGG) are pre-trained on ImageNet. Using these stats is a "safe bet" if specific dataset stats are unavailable.

3. The Normalizer (`normalize_image`)

```
def normalize_image(image, time_step, dataset_name):
```

- **Purpose:** Prepares an image for the Neural Network.

```
    mean, std = get_constants(dataset_name, time_step)
```

- **Line:** Retrieves the correct Mean and Std using the helper function above.

```
    # Ensure image is float for division
    image = image.astype(np.float32)
```

- **Line:** Converts the image data type to 32-bit Floating Point.

- **Why:** Images usually come as `uint8` (integers 0-255). If we divide integers in Python/NumPy, we might lose precision or get errors. We need decimals (e.g., 1.543) for the network.

```
    # Apply normalization (broadcasting handles shape differences)
    normalized_image = (image - mean) / std
```

- **Line:** The **Z-Score Normalization Formula**.
  1. `image - mean`: Centers the data. The average pixel value becomes 0. Darker pixels become negative, brighter pixels become positive.
  2. `/ std`: Scales the data. It ensures the values mostly fall between -1 and 1.
- **Broadcasting:** The image is `(Height, Width, 3)`. The Mean is `(3,)`. NumPy automatically subtracts Red mean from Red channel, Green from Green, etc.

```
    return normalized_image
```

- **Output:** Returns a tensor of floats, ready for the CNN.

4. The Denormalizer (`denormalize_image`)

```
def denormalize_image(normalized_image, time_step, dataset_name):
```

- **Purpose:** Takes the network's data and converts it back to a viewable picture (e.g., for saving a JPG or showing in a GUI).

```
mean, std = get_constants(dataset_name, time_step)
```

- **Line:** Gets the stats to reverse the math.

```
image = (normalized_image * std) + mean
```

- **Line:** Reverses the Z-Score formula.
  1. `* std` : Un-scales the data.
  2. `+ mean` : Un-centers the data (adds the brightness back).

```
# Clip values to valid image range and convert to integer
return np.clip(image, 0, 255).astype(np.uint8)
```

- **Line:** Final safety cleanup.
- `np.clip(image, 0, 255)` : Neural networks are approximate. The math might result in a pixel value of `-5`