1. FlowSmoothnessLoss

Purpose: Regularizes optical flow. It ensures that the "movement" vectors predicted by the model aren't chaotic but flow smoothly across the image.

- `dy = torch.abs(flow[:, :, 1:, :] - flow[:, :, :-1, :])`: Calculates the difference between a pixel and the pixel directly below it.
- `dx = torch.abs(flow[:, :, :, 1:] - flow[:, :, :, :-1])`: Calculates the difference between a pixel and the pixel to its right.
- `return (torch.mean(dx) + torch.mean(dy))`: Returns the average "jitter."
- **Why?** In nature, motion is usually continuous. If one pixel "moves" left, its neighbor likely moves left too. This loss penalizes sudden, jagged changes in flow.

2. SemanticConsistencyLoss (SCL)

Purpose: This is a "Logic Reinforcer." It enforces that if the model predicts **"No Change"** at a pixel, then the semantic category (e.g., Forest) must be identical in both T1 and T2.

- `unchanged_weight = 1.0 - change_mask.float()`: Inverts the change mask. Now, 1 means "Unchanged" and 0 means "Changed."
- `if unchanged_weight.sum() < 1e-5`: A safety check. If the entire image changed, we can't calculate consistency, so we return 0 to avoid errors.
- `if self.metric == 'cosine'`:
    - `sim = F.cosine_similarity(...)`: Measures the angle between the T1 and T2 feature vectors.

    - `loss = (1.0 - sim) * unchanged_weight`: If similarity is 1 (perfect), loss is 0. We multiply by the weight so we **only** penalize differences in the "Unchanged" areas.
- `return loss.sum() / unchanged_weight.sum()`: Averages the loss based only on the number of unchanged pixels.
- **Why?** This forces the network's internal representation to be "Seasonal Invariant." It helps the model ignore color changes caused by light or seasons while focusing on real semantic category changes.

3. DiceLoss

Purpose: Handles class imbalance in the Change Detection map.

- `preds = torch.sigmoid(logits).view(-1)`: Converts raw scores to probabilities (0 to 1) and flattens them into a long list.
- `intersection = (preds * targets).sum()`: Calculates where the model and the ground truth both agree there is a "Change."
- `dice = (2. * intersection + self.smooth) / (preds.sum() + targets.sum() + self.smooth)`: Calculates the overlap ratio.
- `return 1 - dice`: Since we want to *minimize* loss, we subtract the overlap from 1.
- **Why?** In change detection, "Change" pixels are usually very rare compared to "No-Change" pixels. Standard loss (BCE) would get a 99% score by just guessing "No Change" everywhere. Dice Loss forces the model to actually find the small changed regions.

4. LambdaSCDLoss (The Master Loss)

Purpose: This is the conductor. It coordinates all other losses into one final "Total Loss" value.

Initialization

- `self.sem_ce = nn.CrossEntropyLoss(...)`: Standard loss for identifying land types (Forest, Water, etc.).
- `self.bcd_bce = nn.BCEWithLogitsLoss()`: Standard loss for the "Change vs. No Change" binary map.
- `self.bcd_dice = DiceLoss()`: The overlap-based loss described above.

The Forward Pass

- `loss_sem = (loss_sem1 + loss_sem2) / 2`: Calculates how well the model classified the land type in both T1 and T2 images.
- `target_bcd = targets['bcd'].unsqueeze(1).float()`: Prepares the change mask by adding a "Channel" dimension so it can be multiplied with feature maps.
- `loss_bcd = loss_bcd_bce + loss_bcd_dice`: Combines the "strength" of BCE (pixel accuracy) with the "precision" of Dice (overlap).
- `loss_scl = self.scl_loss(...)`: Applies the Semantic Consistency logic mentioned in section 2.
- `loss_kld = outputs.get('kld_loss', ...)`: Measures "Uncertainty." This comes from the Information Bottleneck gate. It penalizes the model if it carries too much "noisy" information from the input to the output.
- `loss_flow = self.flow_loss(outputs['flow'])`: Applies the smoothness regularization to the optical flow vectors.

The Weighted Total

- `total_loss = loss_sem + loss_bcd + (self.lambda_scl * loss_scl) + ...`:
- **Why the multipliers (`lambda`)?** Not all losses are equally important. For example, `lambda_flow` is very small (0.01) because flow smoothness is just a "hint" to help the model, whereas `loss_sem` is a primary goal.
- `return { ... }`: Instead of just returning one number, it returns a dictionary. This allows the user to see exactly which part of the model is struggling during training (e.g., "The segmentation is good, but the flow is

messy").