IBGate: The Information Bottleneck Gate

This module uses a "Lambda Map" (a probability map of where change occurs) to decide which features are important.

- `self.gate_encoder`: A small network that takes the features + the lambda map and produces a "Gate" (values between 0 and 1 via Sigmoid).
- `if lambda_map.shape[-2:] != x.shape[-2:]`: Checks if the guidance map matches the resolution of the current features. If not, it uses bilinear interpolation to resize it.
- `torch.cat([x, gate_map], dim=1)`: Combines the original features and the resized lambda map. This allows the gate to see both the "what" (features) and the "where" (probability map).
- `return x * (1 + gate)`: This is a **Residual Gate**. Instead of just multiplying by the gate (which could zero out features), it adds 1. This means the gate acts as a "boost" for regions where the model is confident change is happening.

FeatureDifferenceModule: Learnable Change Detection

In standard change detection, you just subtract two images: `abs(T1 - T2)`. This module makes that process "smart."

- `diff = torch.abs(f1 - f2)`: Calculates the standard pixel-wise difference.
- `torch.cat([f1, f2, diff], dim=1)`: Concatenates the original T1 features, T2 features, and the difference.
- `self.fusion`: A 1×1 Convolution that learns how to weigh these three components.
- **Why?** Pure subtraction is noisy. By looking at `f1` and `f2` directly, the model can distinguish between "real change" (a new building) and "noise" (a shadow or different lighting).

MambaFusionBlock: The Core Building Block

This block merges "Up-sampled Deep Features" with "Skip Connection Features" (coming from the encoder).

- `self.skip_proj`: A 1×1 Convolution that ensures the skip connection features have the same number of channels as the deep features.
- `x_up = F.interpolate(...)`: Upsamples the deep features to match the higher resolution of the skip features.
- `self.fusion_conv`: Concatenates the two and uses a 3×3 convolution to blend them.
- `x_hwc = x_fused.permute(0, 2, 3, 1)`: Changes the shape from $[B, C, H, W]$ to $[B, H, W, C]$.
- **Why?** The **Mamba (SS2D)** backbone expects data in "channel-last" format (like a list of pixels) rather than "channel-first" (like a stack of images).
- `self.mamba(x_norm, x_norm)`: Runs the Mamba state-space model. This allows every pixel in the fused map to scan its neighbors in four directions to refine the context globally.
- `return x_fused + out_spatial`: Adds the refined Mamba features back to the original fused features (Residual connection).

MultiScaleMambaDecoder: The U-Net Style Decoder

This class implements a hierarchical decoder that reconstructs the image resolution from the deepest bottleneck.

- `self.encoder_dims`: Stores the list of channels from the encoder (e.g., `[96, 192, 384, 768]`).
- `self.stage1, stage2, stage3`: These are `MambaFusionBlock` instances. Note the order: `stage1` fuses the deepest layer (`encoder_dims[3]`) with the next layer up (`encoder_dims[2]`).
- `self.ib_gate`: An Information Bottleneck Gate applied only to the deepest features (`c4`).

- **Why?** The deepest features contain the most abstract semantic info. Filtering noise here via the `lambda_map` (Uncertainty/Change awareness) prevents errors from propagating upward through the whole decoder.
- `c1, c2, c3, c4 = features`: Unpacks the multi-scale features from the encoder (from highest resolution to lowest resolution).
- `x = self.stage1(x, c3) ... stage3(x, c1)`: The "Decoding Upward" phase. It progressively increases the spatial resolution while incorporating "Skip Connections" (`c3, c2, c1`) to recover fine details lost during downsampling.
- `logits = self.head(x)`: The final 1×1 convolution that maps the processed features to the required number of output classes.

BCDDecoder: Binary Change Detection Decoder

This is the top-level module that produces the final "Change vs. No-Change" map.

1. Multi-Scale Difference

- `for f1, f2 in zip(feats_t1, feats_t2): diffs.append(...)`: It calculates the "Learnable Difference" at every single resolution (1/4, 1/8, 1/16, 1/32).

2. Multi-Scale Lambda Injection

- `self.lambda_inj_d4, d3, d2`: These are 1×1 Convolutions that project the 1-channel lambda map into high-dimensional space.
- `inject_lambda(feat, l_inj)`: A helper function that resizes the lambda map and **adds** it to the features.
- **Why?** Instead of just using the lambda map at the very beginning, this "injects" it at every stage of the upsampling process. This ensures the model stays focused on the regions of interest as it builds the final high-resolution map.

3. The Decoding Flow

1. **Bottleneck:** Start with deepest difference `d4`. Inject Lambda.
2. **Stage 1:** Upsample and fuse with `d3`. Inject Lambda.
3. **Stage 2:** Upsample and fuse with `d2`. Inject Lambda.
4. **Stage 3:** Upsample and fuse with `d1`.
5. **Head:** Use a final 1×1 Conv to collapse all features into a single prediction map.