Homework

- The current UNet implementation only processes 1D signals.
- Your homework is to take the UNet1D implementation and convert it into one that can process images.
- Your final architecture should accept batches of images: (batch x channel x height x width).
- No need to train it on anything yet.
- Turn in your implementation condensed into a single .py file.

Grading Rubric for 1D UNet Modifications and Testing

Overview

This rubric evaluates the successful modification of the provided 1D UNet code to handle 2D image inputs and the implementation of a test with a simulated image input. The code should process 2D inputs (e.g., images with batch size, channels, height, and width) instead of 1D inputs (e.g., time series or spectrograms). Students must also test the modified code using the provided example input.

The rubric is divided into two main sections: **Code Modifications** (90 points) and **Testing** (10 points). Each criterion specifies the expected changes or implementation details and the points awarded for correct completion.

Example Input for Testing

Students should use the following code to test their modified UNet model:

```
if __name__ == '__main__':
    batch_size = 32
    in_channels = 3
    img_height = 128
    img_width = 128

    model = UNet(in_channels)

    x = torch.randn(batch_size, in_channels, img_height,
       img_width)
    print(x.shape)

    y = model(x)
    print(x.shape, y.shape)
```

This code creates a simulated image input with a batch size of 32, 3 input channels (e.g., RGB), and a spatial resolution of 128×128 pixels. The model should process this input and produce an output with the same shape.

Rubric

0.1 Code Modifications (90 points)

The following components of the `1D_UNet.py` code must be modified to handle 2D image inputs. Each component is evaluated based on the correctness of the modification to support 2D convolutions, rearrangements, and other operations.

0.1.1 DownBlock (15 points)

- **Description**: Modify the `DownBlock` class to handle 2D inputs by updating the `Rearrange` operation to downsample the spatial dimensions (height and width) and adjust the `Conv1d` to `Conv2d`.

0.1.2 UpBlock (15 points)

- **Description**: Modify the `UpBlock` class to upsample 2D inputs and apply a 2D convolution.

0.1.3 Block (10 points)

- **Description**: Update the `Block` class to use 2D convolutions and normalization for 2D inputs.

0.1.4 ResBlock (10 points)

- **Description**: Modify the `ResBlock` class to use a 2D skip connection.

0.1.5 RMSNorm (10 points)

- **Description**: Update the `RMSNorm` class to normalize 2D inputs across the channel dimension.

0.1.6 LinearAttention and Attention (15 points)

- **Description**: Modify both `LinearAttention` and `Attention` classes to handle 2D inputs by updating the rearrangement and convolution operations.

0.1.7 UNet (15 points)

- **Description**: Update the `UNet` class to initialize and process 2D inputs.

0.1.8 Successful Execution and Output (10)

- **Description**: The test code must run without errors and produce the expected output shapes.

1 Total Points: 100

- **Code Modifications**: 90 points (DownBlock: 15, UpBlock: 15, Block: 10, ResBlock: 10, RMSNorm: 10, LinearAttention/Attention: 15, UNet: 15)

- **Testing**: 10 points

Notes for Students

- Ensure all modifications support 2D inputs by replacing `nn.Conv1d` with `nn.Conv2d` and updating `Rearrange` operations to handle both height and width dimensions.
- Verify that the model processes the input tensor `(32, 3, 128, 128)` and produces an output of the same shape.
- Test the code thoroughly to avoid runtime errors, such as shape mismatches or incorrect tensor operations.