Conv2d and Conv2dTranspose

Your task involves implementing Conv2d and Conv2dTranspose operations and comparing them with PyTorch's equivalents. You have to use only basic tensor operations from PyTorch. The solution when you're using F.conv2d or F.conv2_transpose operations is not valid. **Please, implement it in the Google Colaboratory and share results.** We estimate this task takes about 2-3 hours max. You can encapsulate solutions in specific classes or use a full functional approach depending on your choice.

Here's a detailed breakdown of each step:

1. Implement Conv2d Forward Operation:

In this step, you'll create the forward operation for the Conv2d layer. You can refer to the PyTorch documentation to understand the exact formula for the convolution operation. Use the same stride, groups, dilation parameters as in the PyTorch code.

- Write a function that takes input data, weights, and biases as inputs.
- Implement the convolution operation.
- Ensure that the output matches PyTorch's Conv2d forward output within a small tolerance (e.g., using the Mean Squared Error or a similar metric) with the same inputs, weights and biases

2. Implement Conv2d Backward Operation:

Here, you'll derive and implement the backward equations for the input data, weights, and biases of the Conv2d layer.

- Derive the gradients with respect to input data, weights, and biases using backpropagation formalism
- Write functions to compute gradients for input data, weights, and biases.
- Implement the functions to compute gradient of inputs (dL / dx), weights (dL / dw) and biases (dL / db) using the gradient (dL / dy) as input into backward function (**remember: use only basic tensor operations or your implemented code**)
- Verify that the computed gradients match those obtained from PyTorch's backward operator, and compare the results using an appropriate metric

3. Implement and Compare Conv2dTranspose:

In this step, you'll focus on the Conv2dTranspose (also known as deconvolution) operation, which is used for upsampling or "undoing" the effect of a convolution.

- Implement the forward operation for Conv2dTranspose, considering default stride, dilation, and groups parameters.
- Create a function to compute the gradients for input data, weights, and biases during the backward pass.
- Compare the output of your Conv2dTranspose implementation with PyTorch's Conv2dTranspose using a suitable metric.

Overall Evaluation:

Throughout the process, you'll need to compare your implementations with PyTorch's built-in functions. Metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), or gradient differences can be used for comparison. Ensure that your implementations produce similar outputs and gradients, with differences within a small tolerance (1e-6 or similar magnitude).

Remember to thoroughly test your implementations with various inputs, weights, and biases to ensure their correctness. Document your code, equations, and comparisons clearly to demonstrate your understanding and the effectiveness of your implementations.