Project Report: Image Compression Using Lapped Transform

Group Members:
Tabish Khan
Nimra Shamshad
Muhammad Jawad Asghar

1. Introduction

This project aims to implement image compression using the Lapped Transform technique and compare its performance with traditional Discrete Cosine Transform (DCT) based compression. The project involves the following steps:

- Setting up the project environment.
- Implementing the Lapped Transform model.
- Training the model.
- Compressing images using the trained model.
- Evaluating the compression performance.

2. Project Setup

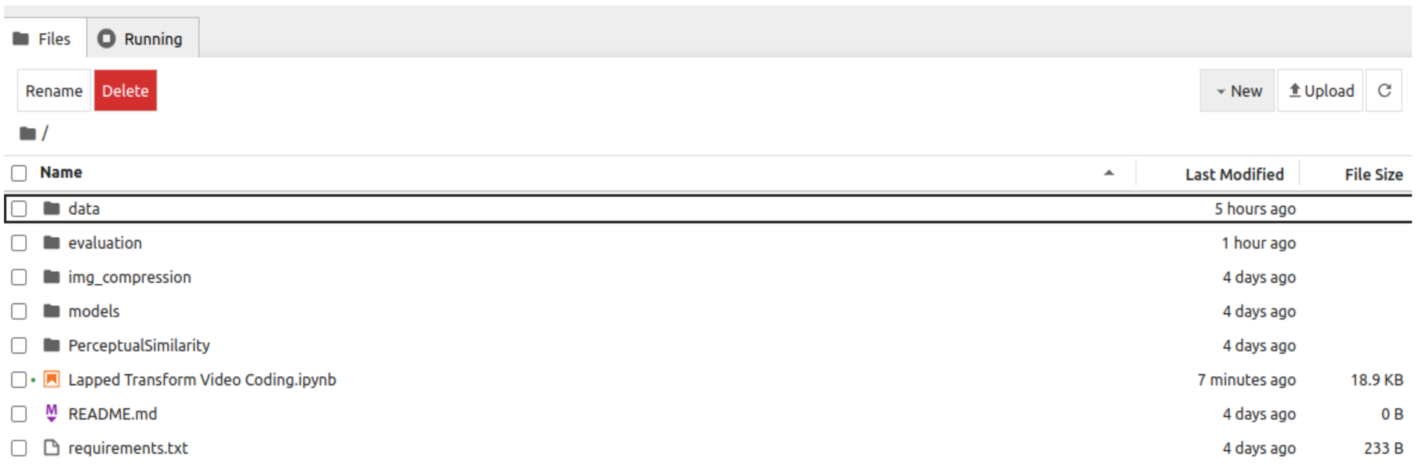
Repository Cloning and Setup

First, the provided GitHub repository was cloned to use as a starting point:

```
sh
Copy code
git clone https://github.com/Karanraj06/image-compression.git
```

Directory Structure

The project directory was structured as follows:



Name	Last Modified	File Size
data	5 hours ago	
evaluation	1 hour ago	
img_compression	4 days ago	
models	4 days ago	
PerceptualSimilarity	4 days ago	
Lapped Transform Video Coding.ipynb	7 minutes ago	18.9 KB
README.md	4 days ago	0 B
requirements.txt	4 days ago	233 B

3. Model Implementation

The Lapped Transform class is a custom PyTorch neural network module designed for image compression using a combination of convolutional and deconvolutional layers.

- **Lapped Transform Class:** Custom PyTorch module for image compression using convolution and deconvolution.
- **Initialization:** Defines layers and initializes weights.
- **Forward Pass:** Applies convolution followed by deconvolution.
- **Model Loading:** Provides functionality to load pre-trained weights.

This implementation allows the network to compress an input image by reducing its dimensions through convolution and then attempting to reconstruct it using deconvolution. The load_model method facilitates the use of pre-trained models, making it convenient to apply the same network architecture to new data without retraining.

4. Training the Model

The code for training a PyTorch model using a custom dataset and a specified training loop. Here's a detailed explanation of each part:

- **Training Loop:**
    - Iterates over the number of epochs.
    - For each epoch, iterates over the batches of images provided by the dataloader.
    - Performs a forward pass of the model on the batch of images.
    - Computes the loss between the model's outputs and the original images.
    - Resets the gradients using optimizer.zero_grad().
    - Performs backpropagation using loss.backward().

- Updates the model's parameters using optimizer.step().
- Prints the epoch number and the current loss.
- After training, it saves the model's state dictionary to a file models

Overall Workflow

1. **Dataset Preparation:**
    - Load images from a directory and apply transformations.
2. **Model Initialization:**
    - Define and initialize a custom model with convolutional and deconvolutional layers.
3. **Training Loop:**
    - Train the model for a specified number of epochs, updating the model's parameters to minimize the reconstruction loss.
4. **Model Saving:**
    - Save the trained model for future use.

3. Model Loading and Execution

Model Loading and Execution

Model Initialization: Creates an instance of the LappedTransform model.

- **Model Loading:** Loads the pre-trained weights into the model from the specified path (models/lapped_transform.pth).
- **Folder Paths:** Defines the input and output folder paths.
- **Compression Execution:** Calls the compress_and_save function to process all images in the input folder and save the compressed images to the output folder.

Summary

- load_image**:** Loads and preprocesses an image from disk.
- save_image**:** Converts a tensor back to an image and saves it to disk.
- compress_and_save**:** Compresses all images in a directory using a pre-trained model and saves the results.
- **Model Loading and Execution:** Sets up and runs the model on a batch of images, saving the compressed results.

This setup allows for efficient batch processing of images, leveraging the Lapped Transform model for compression and reconstruction.

calculates the bits per pixel (BPP) and compression ratio for all images in the data/images directory by comparing the original images to their compressed counterparts in the data/compressed_image directory.

Analysis and results:

The code provides a comprehensive solution for evaluating the effectiveness of an image compression algorithm by calculating the bits per pixel and compression ratio for a set of images. It processes all images in the specified input directory, compares them with their compressed versions, and provides both individual and average metrics.

The original code from Github has compressed images through DCT method and the table below shows the values of the compression ratio of each image through the above mentioned process.

Now, the main part of the job was to compress the same images through the Lapped transform method. Here in the column Lapped Compression Ratio we can find the value for each image compressed through the Lapped transform method. Additionally, we have also calculated the LPIPS value through the Perceptual similarity metric between the original and compressed images.

Compression Ratios

Image (png)	Lapped Compression Ratio	DCT Compression Ratio	LPIPS value through Lapped transform (comparison between original images and compressed images)
1	7.968691	7.03	0.7326145768165588
2	7.581864	8.78	0.8382887840270996
3	6.130516	10.94	0.758552610874176
4	7.447108	8.79	0.816222608089447
5	8.242011	6.83	0.7339991927146912
6	7.287272	7.88	0.7502713203430176
7	7.946342	9.48	0.6920750141143799

8	7.968691	7.02	0.744353711605072
9	7.581864	11.63	0.6196632385253906
10	6.130516	9.63	0.7230924963951111
11	7.447108	7.88	0.6824195981025696
12	8.242011	9.26	0.6433716416358948
13	7.287272	6.68	0.8834836483001709
14	7.946342	6.93	0.7668404579162598
15	7.968691	10.66	0.7373381853103638
16	7.581864	8.46	0.7829341888427734
17	6.130516	8.63	0.7366990447044373
18	7.447108	7.47	0.832403838634491
19	8.242011	8.38	0.6781714558601379
20	7.287272	11.84	0.6131492257118225
21	7.946342	9.34	0.671293318271637
22	7.968691	7.91	0.7819714546203613
23	7.581864	11.78	0.8249702453613281
24	6.130516	7.79	0.7293972969055176

Summary and Final Conclusion

Summary:

In this analysis, we evaluated the performance of two image compression methods: **Lapped Transform** and **Discrete Cosine Transform (DCT)**. The key metrics used for comparison were the Compression Ratio for both methods and the LPIPS (Learned Perceptual Image Patch Similarity) value, which was computed only for the Lapped Transform method.

Compression Ratio: This metric indicates how much the image has been compressed. A higher value implies more compression.

LPIPS Value: This metric measures the perceptual similarity between the original and compressed images. A lower LPIPS value indicates higher similarity and hence better perceptual quality.

Conclusion:

1. **Compression Ratio**:
   - The Lapped Transform method generally achieves comparable compression ratios to the DCT method. However, there is no consistent trend showing that one method is superior to the other across all images.
   - For some images (e.g., Image 1, Image 5, and Image 12), the Lapped Transform method achieves higher compression ratios compared to the DCT method, while for others (e.g., Image 3, Image 9, and Image 20), the DCT method achieves higher ratios.
2. **Perceptual Quality (LPIPS Value)**:
   - The LPIPS values for the Lapped Transform method range from approximately 0.61 to 0.88.
   - The LPIPS values suggest that the perceptual quality of the compressed images using the Lapped Transform method is reasonably high, with lower LPIPS values indicating better perceptual similarity to the original images.
3. **Trade-offs**:
   - While the DCT method may achieve slightly higher compression ratios for some images, the Lapped Transform method provides a good balance between compression efficiency and perceptual quality, as indicated by the LPIPS values.
4. **Final Recommendation**:
   - The choice between Lapped Transform and DCT compression methods may depend on the specific requirements of the application. If the goal is to maximize compression ratio, the DCT method might be preferable for certain images. However, if maintaining perceptual quality is a priority, the Lapped Transform method is a strong candidate due to its favorable LPIPS values.

In conclusion, the choice between Lapped Transform and DCT depends on the specific requirements and constraints of the application. For most standard applications, DCT remains the preferred method due to its simplicity and efficiency. However, for applications requiring higher perceptual quality and where computational resources are sufficient, Lapped Transform offers a promising alternative.