# Project 2 Report: Image-Compression Using Lapped Transform

**Group Members:**
Azmat Sultan Awan (65821)
Syed Mir Izhar Ali Shah

## 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:

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

## 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 model

## 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. Here's a detailed explanation of the code:

## Summary:

This 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 provided code imported from Github evaluates the perceptual similarity between original and compressed images using the LPIPS (Learned Perceptual Image Patch Similarity) metric.

## Compression Ratios

| Image (png) | DCT Compression Ratio | Lapped Compression Ratio | Lapped_Perceptual Similarity (LPIPS) |
|---|---|---|---|
| 1 | 7.3 | 32 | 0.47 |
| 2 | 9.34 | 32 | 0.53 |

| | | | |
|---|---|---|---|
| 3 | 10.94 | 32 | 0.28 |
| 4 | 7.97 | 32 | 0.38 |
| 5 | 8.78 | 32 | 0.42 |
| 6 | 9.48 | 32 | 0.47 |
| 7 | 7.79 | 32 | 0.26 |
| 8 | 9.26 | 32 | 0.42 |
| 9 | 8.46 | 32 | 0.33 |
| 10 | 7.02 | 32 | 0.35 |
| 11 | 6.68 | 32 | 0.44 |
| 12 | 6.93 | 32 | 0.40 |
| 13 | 6.83 | 32 | 0.60 |
| 14 | 8.79 | 32 | 0.47 |
| 15 | 7.47 | 32 | 0.36 |
| 16 | 11.63 | 32 | 0.40 |
| 17 | 8.38 | 32 | 0.29 |
| 18 | 7.91 | 32 | 0.41 |
| 19 | 11.84 | 32 | 0.41 |
| 20 | 10.66 | 32 | 0.33 |
| 21 | 9.63 | 32 | 0.44 |
| 22 | 11.78 | 32 | 0.42 |
| 23 | 8.63 | 32 | 0.23 |
| 24 | 7.88 | 32 | 0.43 |

```
Image 2
Original image size: 4718592 bytes
Compressed image size: 147456 bytes
Compression ratio: 32.0
Bits per pixel (bpp): 3.0
Perceptual similarity (LPIPS): 0.534422755241394
```


```
Image 7
Original image size: 4718592 bytes
Compressed image size: 147456 bytes
Compression ratio: 32.0
Bits per pixel (bpp): 3.0
Perceptual similarity (LPIPS): 0.26106807589530945
```


## Summary:

- load_image**:** Loads and preprocesses an image from disk.
- evaluate_similarity**:** Computes the perceptual similarity between two images using the LPIPS model.
- **Main Script:** Iterates over all images in the reference and modified image directories, computes the perceptual similarity for each pair, and prints both individual and average similarity metrics.