Transform Recipes for Efficient Cloud Photo Enhancement

Project ID - 36

Team Members

- Kalyan Adithya M
- Abhishek Reddy Gangapuram
- Jaiwanth Mandava
- Sai Shashank Kalakonda

Main Goal

Main goal of the project is to implement a pipeline that reduces the time and energy cost of uploading an image and downloading the output images after applying certain transformations.

The major setback in image enhancement was the data that was needed to be uploaded/downloaded from the client side which now is cleverly taken care by figuring out the fact that input and output images have a lot in common.

Pipeline

On Client Side

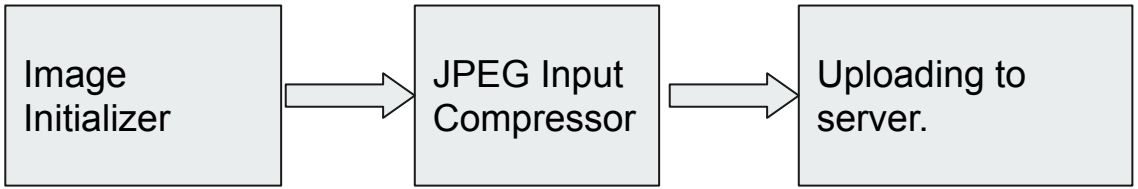
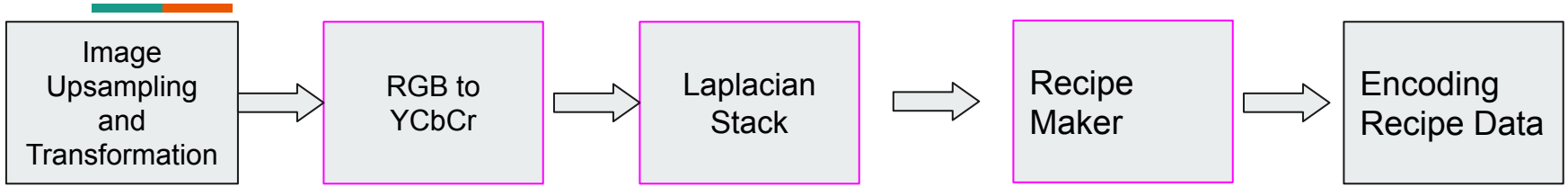


Image Initializer: Selection of an image from the dataset

JPEG Input Compressor: Downsampling the selected image and compressing it so as to decrease its resolution which helps in decreasing the upstream cost.

Uploading to server: Uploading the compressed image onto server along with histograms of each channel of the original image.

On Server Side



Upsampling the image: Upsampling the compressed image using the histograms and applying the required transformation.

RGB to YCbCr: Image is transformed from RGB Space to YCbCr space

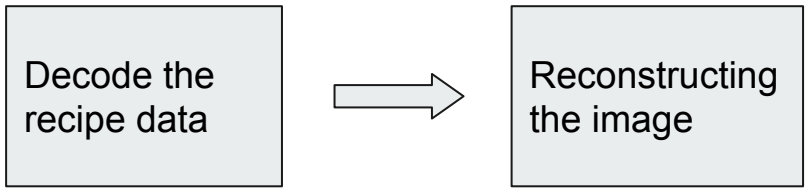
Creating Laplacian Stack: Laplacian Pyramid with (n+1) layers is developed on the image where the last level is known as "Residual layer" which is used in finding the coefficients of the recipe.

Recipe Maker : We extract the features which differentiate the input and output images after transformation.

Encoding Recipe Data: These features are then stored in clever way to reduce the data-transfer that needs to be done to the client side.

Pipeline

On Client Side



Decode the Recipe Data: From the data received from the server we decode the data to get the features back.

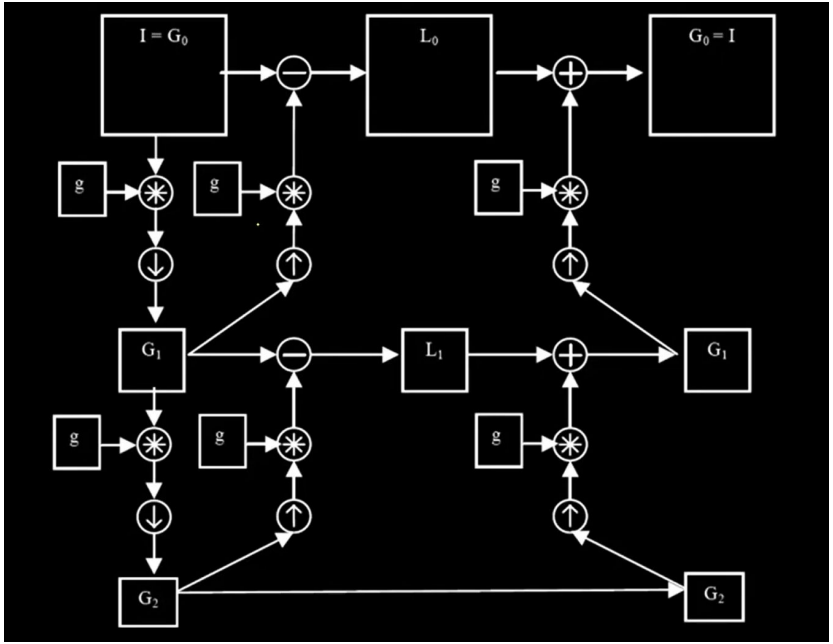
Reconstructing the image: Using the original image and the decoded data we reconstruct the image to get the final transformed image.

Recipe Construction

- Quantizing image into set of blocks in order to compute recipe specific to local image blocks.
- This allows us to capture complex effects even though our image representation and transforms are both relatively simple.
- We work on the images in YCbCr space, and separate Luminance Y from Chrominance (Cb, Cr) while capturing the transformation.
- To capture multi-scale effects, we decompose an image into a Laplacian stack which is a multi-scale image decomposition similar to a Laplacian pyramid, but with all levels stored at the original resolution instead of subsampling which makes it easier to fit recipes. We do this for both input and output images.

Laplacian Stack

- The Gaussian pyramid is a technique in image processing that breaks down an image into successively smaller groups of pixels to blur it.
- Laplacian Pyramid is formed by the difference of consecutive gaussians(after up-sampling the smaller image).
- Laplacian Stack is formed by up-sampling all the levels of the pyramid to the original resolution.



Key Equations

Residual layer

$$R_c(p) = \frac{L_n[O_c](p) + 1}{L_n[I_c](p) + 1}$$

Chrominance Channels

$$\sum_{p \in \mathcal{B}} \| H[O_{CC}](p) - \mathbf{A}_c(\mathcal{B}) H[I](p) - \mathbf{b}_c(\mathcal{B}) \|^2$$

$$H[I] = \sum_{\ell=0}^{n-1} L_\ell[I]$$

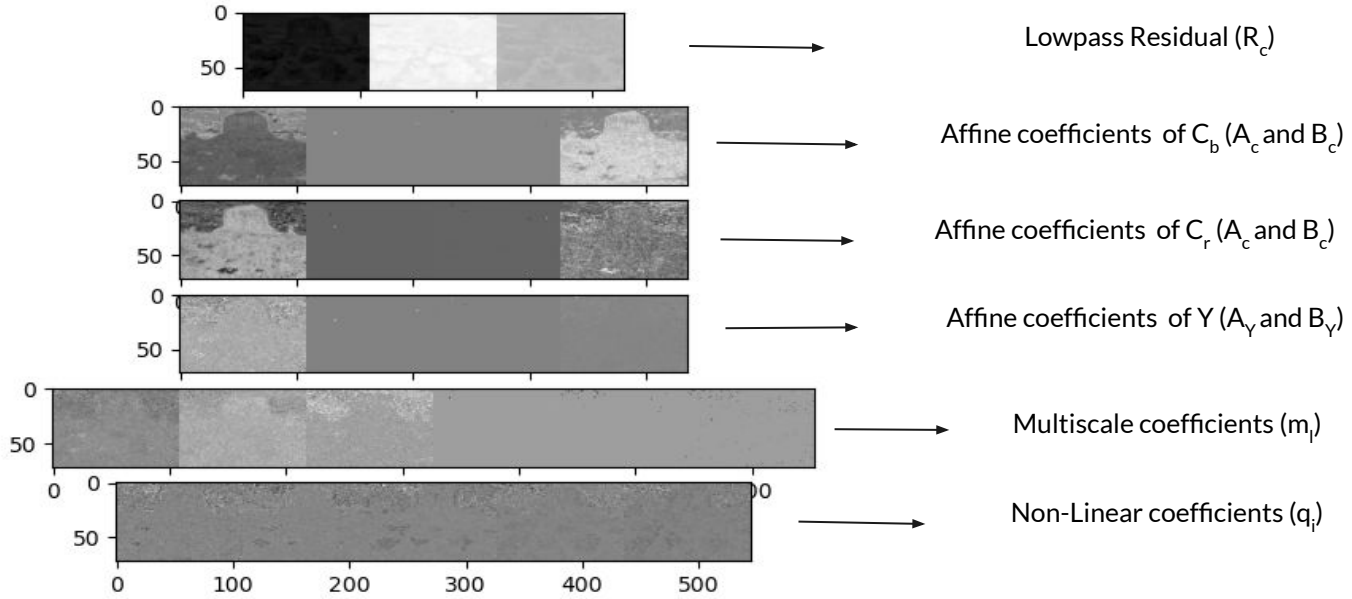
Luminance Channels

$$\sum_{p \in \mathcal{B}} \Big\| H[O_Y](p) - \mathbf{A}_Y(\mathcal{B}) H[I](p) - b_Y(\mathcal{B}) - \sum_{\ell=0}^{n-1} m_\ell(\mathcal{B}) L_\ell[I_Y](p) - \sum_{i=1}^{k-1} q_i(\mathcal{B}) s_i(H[I_Y](p)) \Big\|^2$$

Our Results

Input Image

Output Image



Our Encoded Recipe

To Do

1. Compressing the input image at client side, computing it’s histogram and transfer it over the network.
2. Compressing the recipe for efficient transformation over the network to the client side.
3. Reconstructing the required output image at client-side utilising the compressed recipe.
4. Test the pipeline over different types of transformations and images and obtain numbers to prove the efficiency of the system.