Introduction

- Kashish Aggarwal

- 22113072

Zero-Reference Deep Curve Estimation (Zero-DCE) approaches low-light image enhancement by estimating an image-specific tonal curve using a deep neural network. In this method, a lightweight deep network, known as DCE-Net, is trained to predict pixel-wise and high-order tonal curves to adjust the dynamic range of a given image.

Zero-DCE accepts a low-light image as input and outputs high-order tonal curves. These curves are then applied for pixel-wise dynamic range adjustments, resulting in an enhanced image. The process ensures that the enhanced image's range is maintained and the contrast between neighbouring pixels is preserved. This approach is inspired by the curves adjustment feature found in photo editing software like Adobe Photoshop, where users can adjust the tonal range of an image.

What makes Zero-DCE particularly appealing is its lack of dependence on reference images. Unlike traditional methods, Zero-DCE does not require pairs of input and output images during training. Instead, it relies on a set of carefully designed non-reference loss functions that implicitly evaluate enhancement quality, guiding the network's training process.

The paper work implemented is : https://arxiv.org/abs/2001.06826
https://in.docworkspace.com/d/sIKf3qL1K7orHswY?sa=cl

Set up the environment and imports necessary libraries for building and training a deep learning model using Keras with TensorFlow as the backend.

Setting Backend: Ensuring Keras uses TensorFlow, which is essential if multiple backends are available or if specific TensorFlow features are needed.

Library Imports: These libraries are typically used for:

- Random operations: Data shuffling, random sampling.
- Numpy: Efficient data handling, especially for large datasets.
- Glob: Loading datasets by matching file patterns.
- PIL: Image loading and preprocessing.
- Matplotlib: Visualizing data, model performance, and results.
- Keras: Defining, compiling, and training neural networks.
- TensorFlow: Executing the deep learning models, leveraging GPU acceleration.

Download the LOL Dataset

The LoL Dataset, designed for low-light image enhancement, consists of 485 images for training and 15 images for testing. Each image pair in this dataset includes a low-light input image along with its corresponding well-exposed reference image. We use 300 low-light images from dataset for training and remaining 185 for validation. Then resize the images ti be used for both training and validation purpose(256*256). In DCEnet , we don't require corresponding enhanced images .

DCE-Net

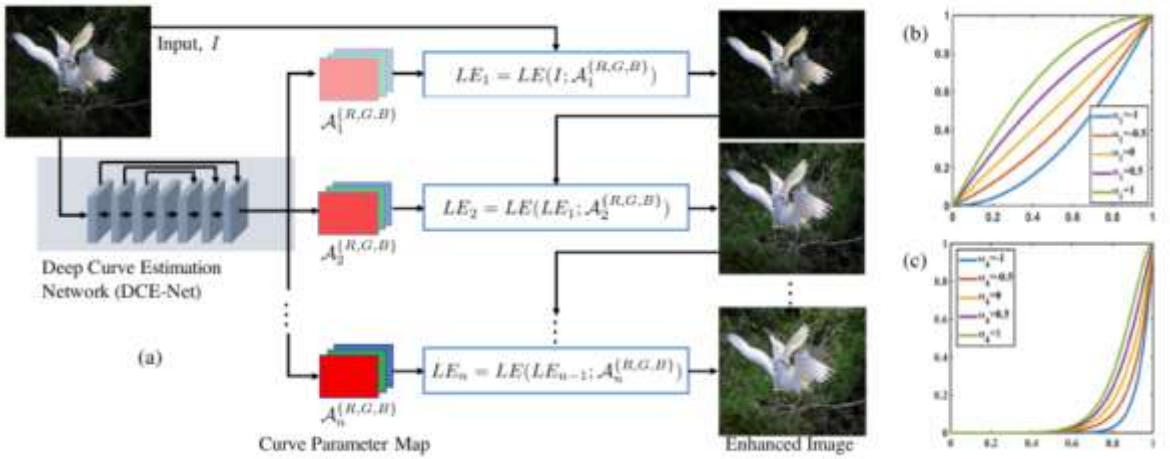
This network estimates set of best-fitting LE-curves for given image. The framework maps all pixels of input's RGB channels and iterate it to get final enhanced image. A light-enhancement curve is a type of curve that automatically maps a low-light image to its enhanced version, with self-adaptive parameters derived solely from the input image. When designing such a curve, three key objectives must be considered:

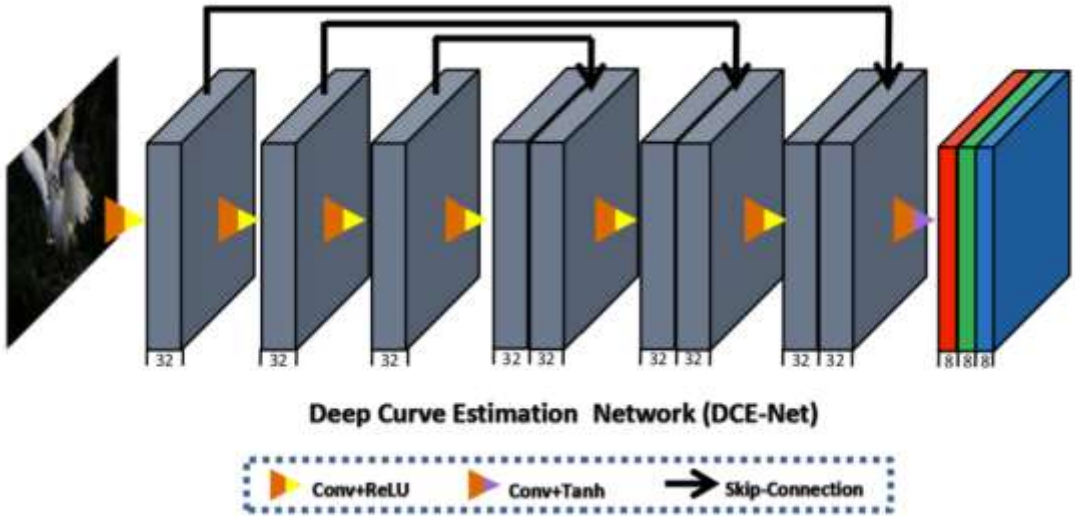
1. Ensure that each pixel value in the enhanced image remains within the normalized range [0,1] to prevent information loss from overflow truncation.
2. Maintain monotonicity to preserve the contrast between neighboring pixels.
3. Keep the curve shape simple and differentiable to facilitate backpropagation during training.

This light-enhancement curve is applied separately to the three RGB channels, rather than just the illumination channel. This approach helps better preserve the image's inherent colors and reduces the risk of over-saturation.

DCE-Net is a lightweight deep neural network designed to learn the mapping between a low-light input image and its optimal curve parameter maps. The network takes a low-light image as input and outputs a set of pixel-wise curve parameter maps corresponding to higher-order curves.

DCE-Net is constructed as a simple convolutional neural network (CNN) with seven convolutional layers and symmetrical concatenation. Each layer comprises 32 convolutional kernels with a 3×3 size and a stride of 1, followed by the ReLU activation function. The final convolutional layer uses the Tanh activation function, producing 24 parameter maps for 8 iterations, with each iteration requiring three curve parameter maps—one for each of the three RGB channels.





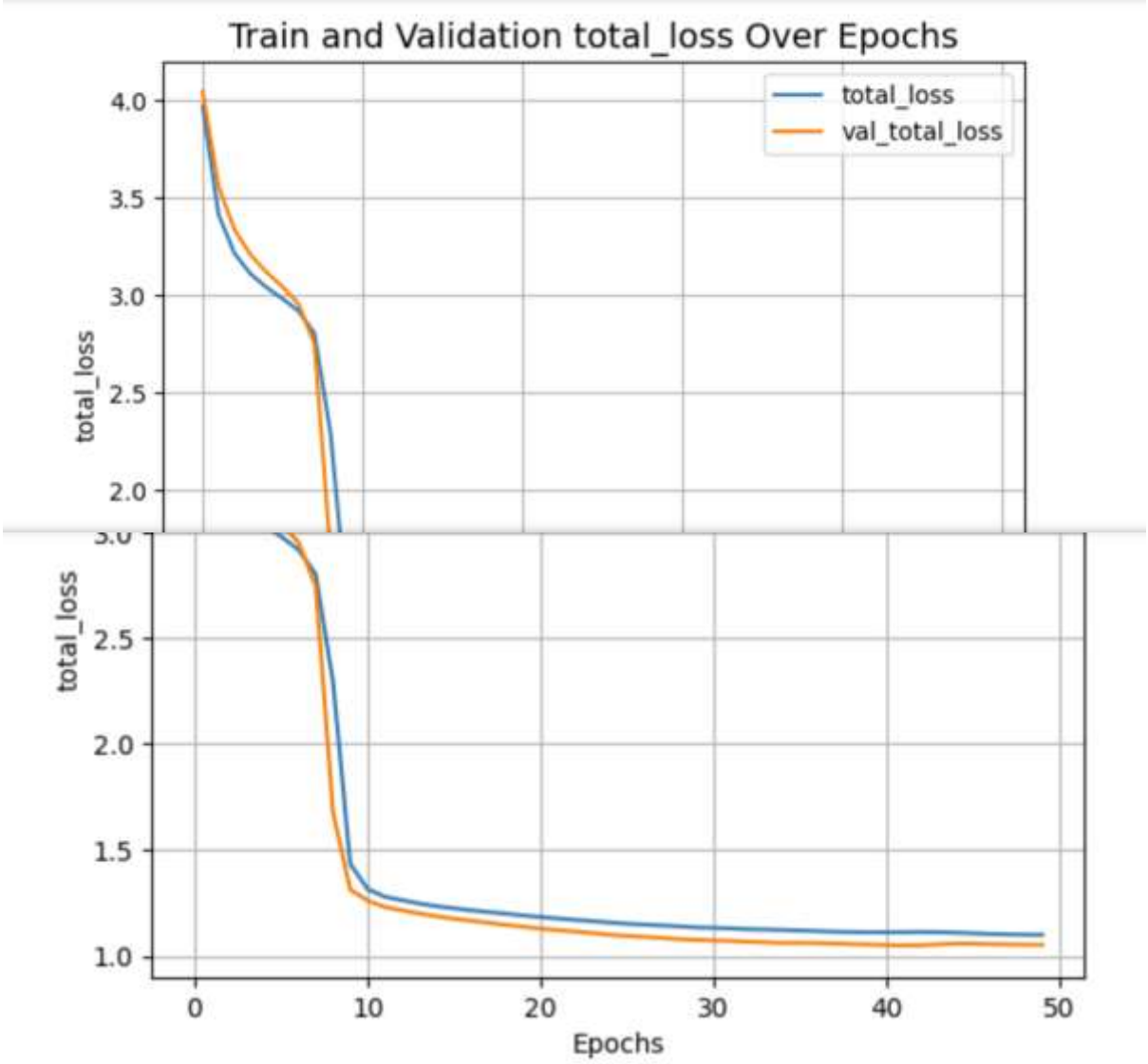
Loss functions

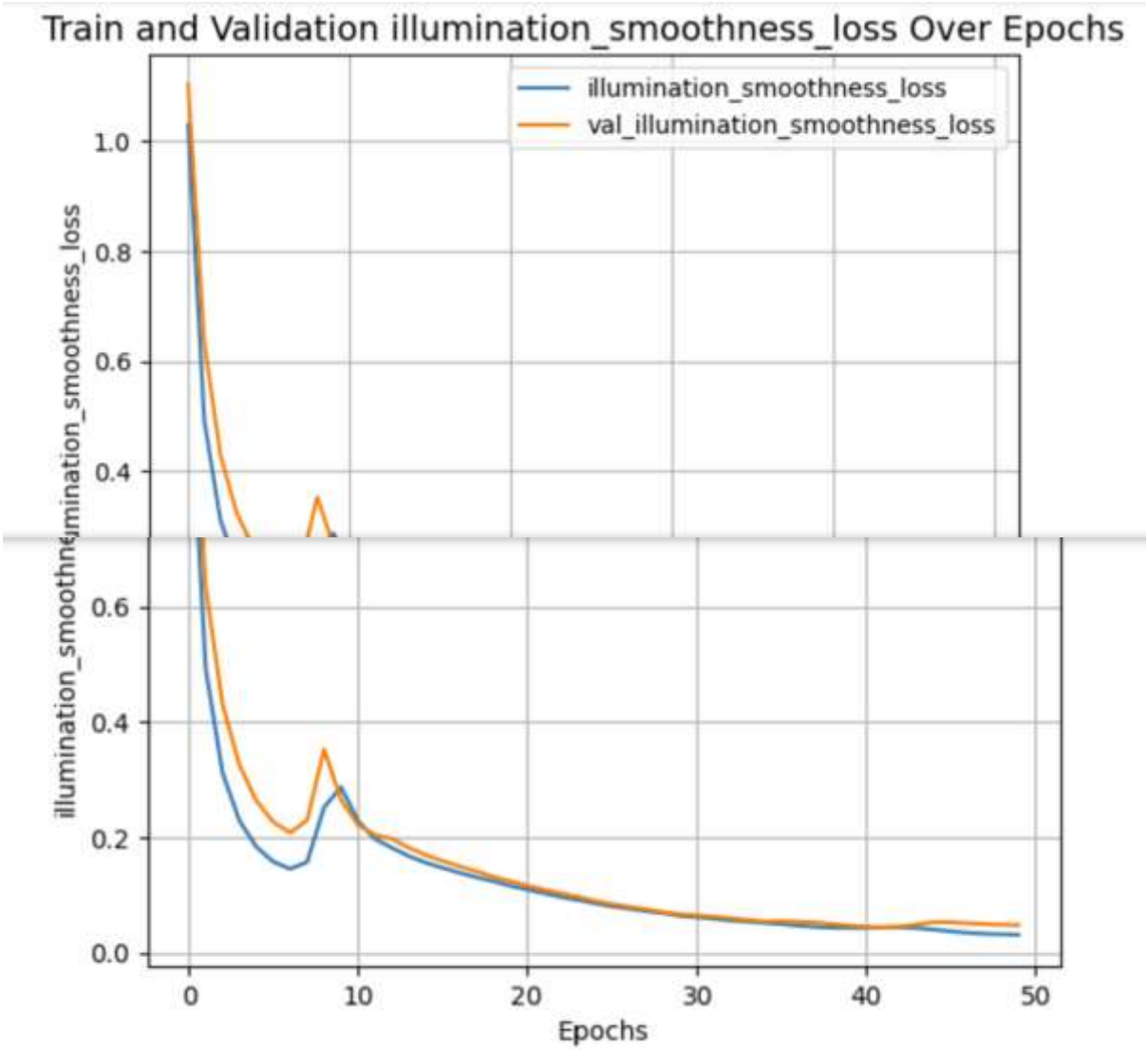
To enable zero-reference learning in DCE-Net, we employ a set of differentiable zero-reference losses to assess the quality of the enhanced images:

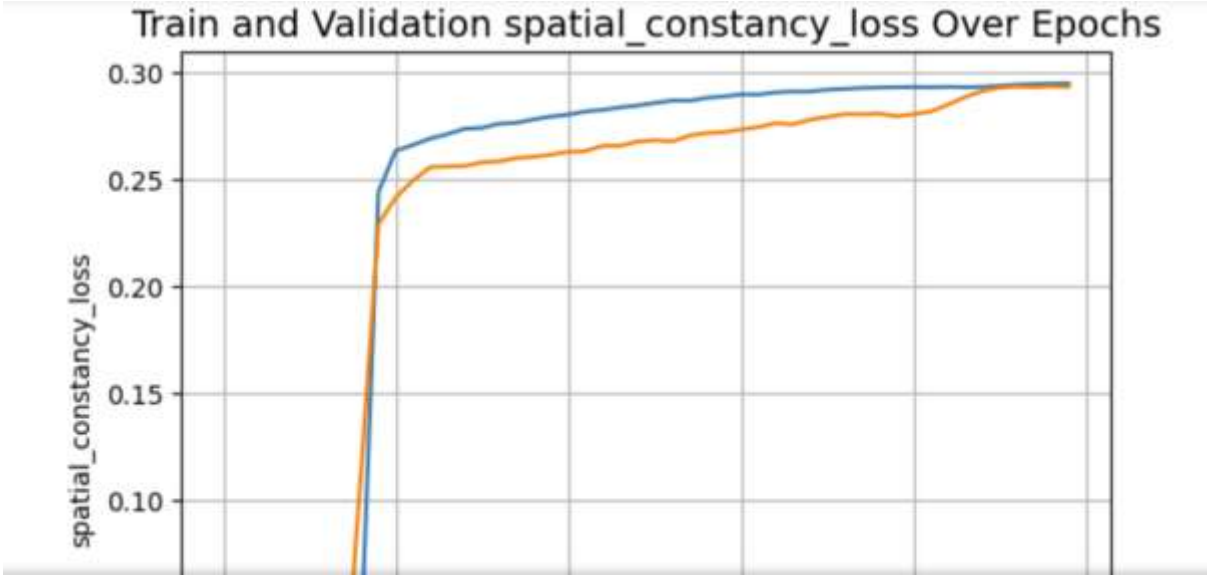
1. **Color Constancy Loss**: This loss corrects potential color deviations in the enhanced image, ensuring consistent color representation.
2. **Exposure Loss**: To manage underexposed or overexposed regions, this loss measures the distance between the average intensity of a local region and a predefined well-exposed level, set to 0.6.
3. **Illumination Smoothness Loss**: This loss preserves the monotonicity between neighbouring pixels by applying it to each curve parameter map, maintaining smooth transitions in illumination.
4. **Spatial Consistency Loss**: This loss promotes spatial coherence in the enhanced image by preserving the contrast between neighbouring regions in both the input

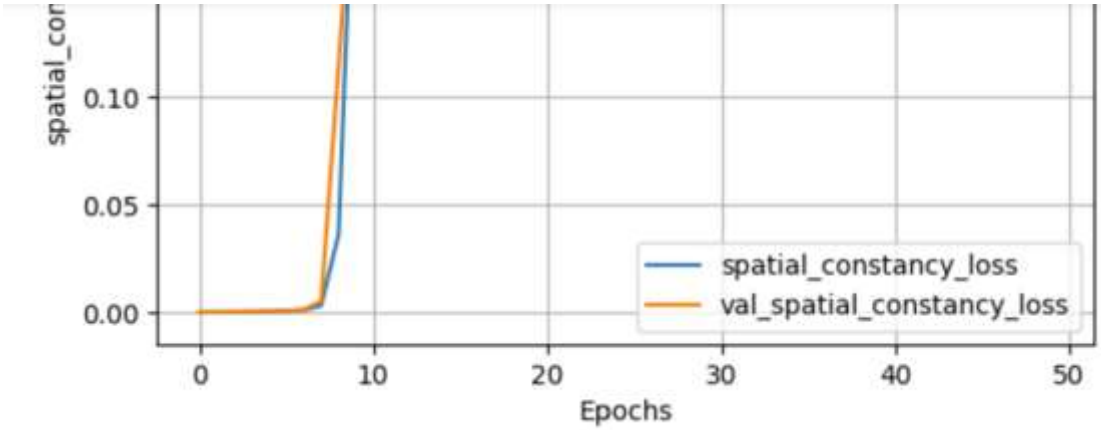
image and its enhanced version.

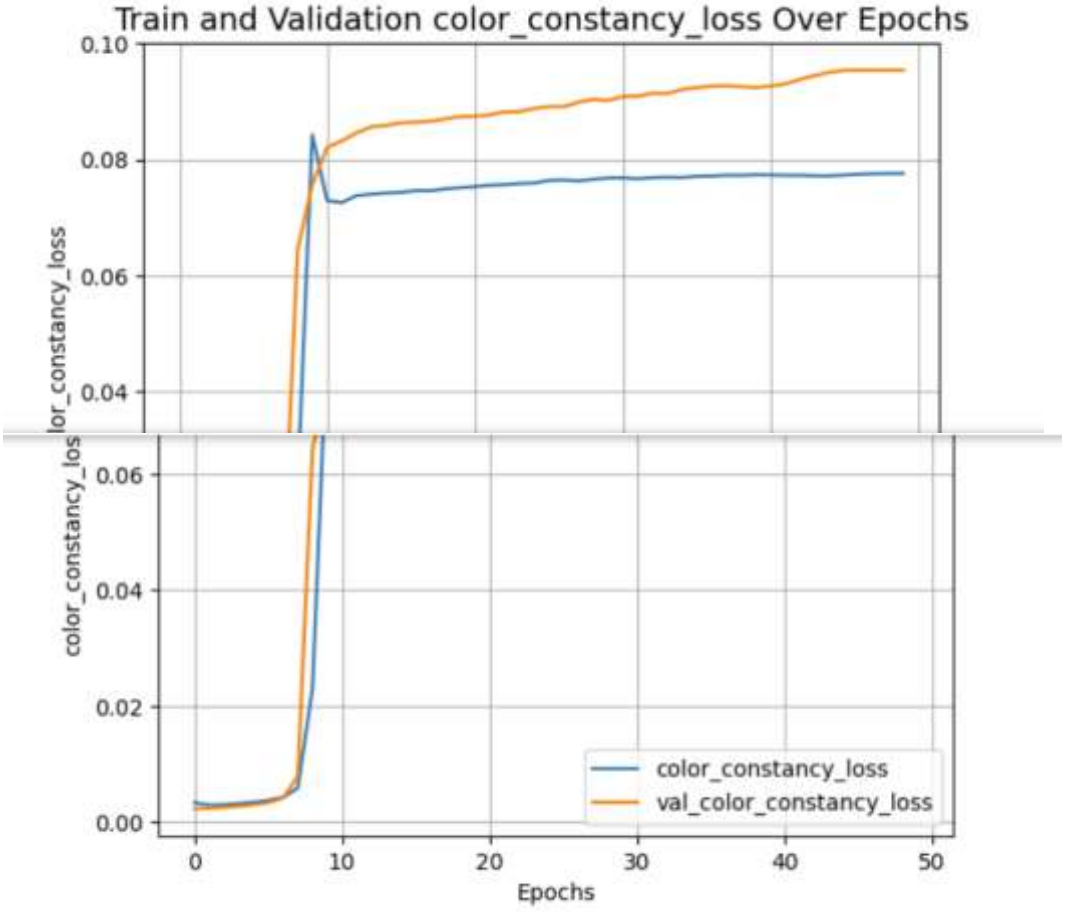
We implement the Zero-DCE framework as a Keras subclassed model. And start model training .

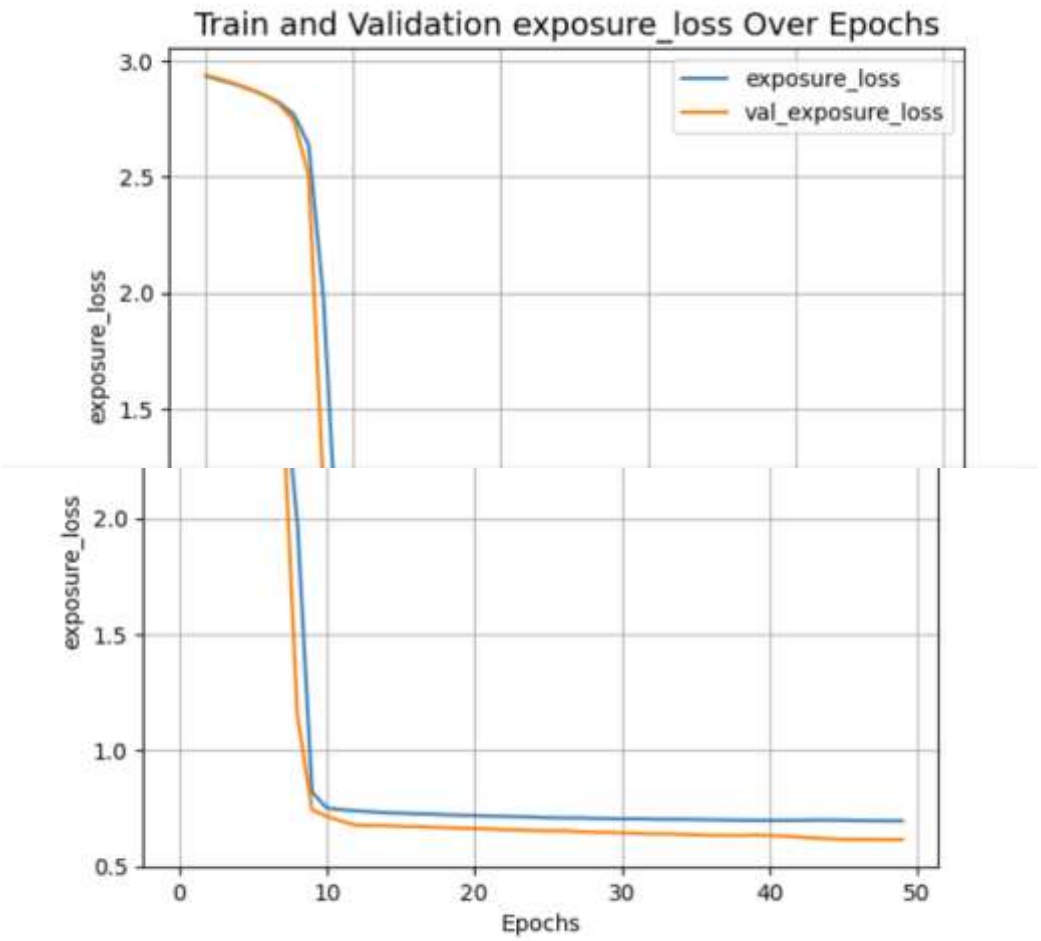



Train and Validation illumination_smoothness_loss Over Epochs
illumination_smoothness_loss
val_illumination_smoothness_loss
illumination_smoothness_loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50


Train and Validation spatial_constancy_loss Over Epochs
spatial_constancy_loss
0.10
0.15
0.20
0.25
0.30


spatial_con
0.10
0.05
0.00
0
10
20
30
40
50
Epochs
spatial_constancy_loss
val_spatial_constancy_loss


Train and Validation color_constancy_loss Over Epochs
lor_constancy_loss
0.10
0.08
0.06
0.04
color_constancy_los
0.06
0.04
0.02
0.00
0
10
20
30
40
50
Epochs
color_constancy_loss
val_color_constancy_loss



Summary-

We can improve this project further by adjusting the number of convolutional layers , number of epochs , learning rate .

We need to check the validation loss with number of epochs . The optimal situation will be the decrease in validation error with number of epochs. Observe the variation of epochs with losses and iterate the dataset with changing values and try to achieve the desired result . I initially started with 32 convolutional layers and 100 epoches . And observe a rough/steep change in values at 20 epoches . Then tried with 10 epoches and got a nearly linear variation of epochs with losses . After interpretation, I tried with 32 layers and 50 epoches and also with 64 layers 50 epoches to check the dependence of dataset on number of convolutional layers . I used a batch size of 16 in this training code . A batch size of 32 can optimise out result more . Also trying learning rate of 1e-3 instead can work out . Trying these possibilities or changing the training dataset , can help us achieve optimisation level of minimum 23 PSNR.

PSNR is widely used to assess the quality of compressed or reconstructed images or videos by comparing them to the original, providing a numerical measure of fidelity. With 64 layers and 50 epoches , it was observed that validation total loss first decreases significantly but after 42 epoches , it started to increase and decrease(almost constant).Getting mse zero , so psnr is inf db. By this observation , we can test training dataset for less than 40 epoches . And observe the pattern . The implemented paper showed a psnr value of nearly 16 with DCEnet on lol dataset.