Improving on Zero-Reference Deep Curve Estimation (Zero-DCE) for Image Enhancement

Charalampos Ioannou
Columbia University
ci2247@columbia.edu

Tanmay Jaiswal
Columbia University
tj2467@columbia.edu

Project Dataset folder
Project Google Colab page

Abstract

This paper improves upon the novel method of Zero-Reference Deep Curve Estimation, on images with a variety of lighting profiles, containing faces. Zero-DCE, formulates light enhancement of an image, as a task of image-specific curve estimation using a DNN. We found that the initial approach of the authors did not produce qualitatively good results on images containing faces. Our method uses the DCE-Net architecture but with different loss functions and improved data augmentation so that the results are improved specifically for our application. The lightweight deep network DCE-Net is trained to estimate pixel-wise and high-order curves for dynamic range adjustment of a given image. The model does not require labeled training data enabling us to use all the data at our disposal. Retraining for a specific condition is as simple as retraining with a set of sample images. The curve estimation is specially designed, considering pixel value range, monotonicity, and differentiability weighted differently from the initial approach. We improve the efficacy of the model in three ways, by augmenting the training dataset, introducing a bounded brightness loss function, and global contrast tuning. We illustrate the quantitative improvement across a wide range of images across different lighting conditions and skin tones of the subjects.

1. Introduction

The widespread adoption of smartphones with their high-quality cameras gives users the ability to snap a photo under a variety of environments. Most of these photos are often captured under sub-optimal lighting conditions due to environmental constraints. As such, there is a great need to post-capture enhance these instances to improve their aesthetic quality and transmission of information. In this paper, we focus our efforts to improve images taken in adverse light conditions, with extreme backlight, under-exposed subjects, and overexposed subjects that contain faces.

Our approach is focused on nuance. We follow the same principles that photographers use while editing their images. While we want to improve visibility in dark regions of the image, we do not want to increase the brightness such that it no longer represents what we see with our eyes. We want to brighten the shadows just enough so that the details are visible. Not any more than that. We also want to reduce the intensity in excessively bright regions for improved detail and clarity.
To achieve this, we make three improvements to the previous approach. First, we perform data augmentation on the initial dataset by varying the exposure of an image from its gamma value. Second, we compensate for the increase in exposure due to the model, as the default model tends to wash out colors and overexpose some areas. We do this by introducing a bounded loss on the brightness level. Third, we improve global contrast over an image. This spreads out the histogram of the image so that the intensities are mapped out over the entire range of colors.

2. System Design

We continue to use the same model architecture as the original paper. A 6 layer CNN with skip connections is used to estimate a set of best-fitting Light-Enhancement curves. It outputs a set of parameters generated on a pixel by pixel level. The LE curves generated from these parameters iteratively enhance a given input image.

The functions used to generate an image from the LE curves are unique and interesting.
$LE(I(x); \alpha) = I(x) + \alpha I(x)(1 - I(x))$
They have some special properties. If the input is in the range [0,1], and the input parameters are in the range [-1, 1], the output is always in the range of [0,1]. The curves are monotonous and differentiable. Applying the curve to the previous output can give us monotonic polynomial curves.

The LE curve is applied separately to three RGB channels instead of solely on the illumination channel. The three-channel adjustment better preserves the inherent color and reduces the risk of over-saturation.

The original paper uses a combination of four weighted losses to define the features that we look for in an image. The losses used originally are Spatial Consistency Loss, Exposure Control Loss, Color Constancy Loss, Illumination Smoothness Loss.
The spatial consistency loss ensures that the contrast between adjacent pixels remains consistent between the input and enhanced image. The exposure control loss tries to increase the exposure of a region to a given mean value (between 0.4 to 0.6). The color constancy loss tries to maintain the mean value of all color intensities around the mean intensity of the image. The illumination smoothness loss ensures that the curves applied to the image are continuous across the image. There shouldn't be very large variations in the corrections between adjacent pixels.

In addition to this, we introduce the over-exposure control loss and the global contrast loss. We supplement this with an additional data augmentation step to provide a wider range of inputs to the model.

Over-Exposure control loss

As admitted by the authors of the paper, the original model overexposes parts of the image that are already well exposed. To counteract this effect the over-exposure control loss penalizes the high luminance values beyond a specified limit.

The loss can be expressed as:

$$L_{ovex} = \frac{1}{M}\sum_{k=1}^{M}(max(Y(k) - C, 0))^2$$

Where M represents the number of nonoverlapping local regions of size 16×16, Y is the average intensity value of a local region in the enhanced image. C is the limit over which luminance is penalized. Setting C to 0.4 provides the best results. This number seems low, but the penalty is also quite small at lower intensities because the loss is squared. This means that higher values are penalized a lot more the further they are from C.

Global contrast loss

The global contrast loss tries to maximize the overall contrast in the image. This helps the image use the entire dynamic range available to it. The contrast of an entire image is proportional to the variance in the intensities of the pixels. To increase the contrast, we maximize the variance. In color images, other than pixel intensity, the difference between the intensities of the colors also creates contrast. Therefore the variance of each color channel was maximized independently. We penalize the model if the variance is below a certain limit.

The loss can be expressed as:

$$L_{gc} = \sum_{\forall P \in E} D - \sigma_P$$

Where σ_p denotes the variance in intensity values of the P channel in the enhanced image. The limit D was varied from 1/24 to 1/2. The variance of a uniform distribution is 1/12. This value was chosen to induce greater variance than a flat histogram. After experimentation, a variance of ⅛ (double the variance of uniform distribution) gave the best results.

Data Augmentation

The images in the face dataset were taken in good light and not all of them were challenging. To overcome this, we added a step to vary the brightness of the input image. The function was loosely based on the way the γfactor of an image is varied. Humans do not interpret the brightness of light linearly. We interpret it almost logarithmically. Professional cameras store the raw linear information captured from the sensor in specific formats. However, consumer formats like JPEG, PNG, and more convert the information such that it is suitable for direct viewing by

applying a gamma correction. We undo the gamma correction by assuming an approximate gamma of 2.2 and reapply a higher/lower gamma value within the range [1.7, 3.2]. This simulates the effect of longer/shorter exposure times resulting in higher/lower brightness. Gamma correction is highly complex and our method is only an approximation of the actual process.

3. Methodology and Experiments

We trained our baseline model (without emphasis on faces) on selected low-light images from Part 1 of the SICE dataset[9]. It consists of 2,300 images in low-light settings. The model does not require labeled data. The output is tuned by changing the weights of the losses and retraining on case-specific images. The images were preprocessed to a fixed size (256x256). Each image is normalized to the range [0,1]. So far, the data and preprocessing are identical to the baseline Zero-DCE model. A data augmentation step to vary the brightness is applied with a probability of 0.3. The model was trained using the Adam optimizer with a learning rate of 0.0001 and a batch size of 32. Overall, we conducted four experiments.

Ablation study
The ablation study tries to assess the impact of the new losses that we introduce compared to the baseline model. The baseline model was trained on the low light dataset using all the losses in the original paper. We retrained the model, adding one of our losses at a time. Our model was trained on the same data as the baseline model. The enhanced test images from the SICE dataset were analyzed visually and the loss weights were finetuned over multiple iterations.

Impact of Dataset
The baseline model was trained with the new losses on the low light dataset. The new model was trained on the LFW dataset with augmentation using the same losses. The VV dataset was found to contain challenging images with humans in them and was used to assess the performance.

Timing analysis
We performed timing analysis on the model to ensure that it can perform inference in real-time. In addition to the image size mentioned in the paper (1200x900), we also analyze the performance with varying file sizes. Unlike training, the inference is rarely performed in a batched setting. The original paper provides run times for a batch size of 32 but we evaluate it on a frame by frame basis.

Quantitative analysis
The original paper evaluates the PSNR, MAE, and SSIM of their model on Part 2 of the SICE dataset[9]. However, certain details about the evaluation methodology are not clear. We do not know the size at which the images were compared. We do not know if the SSIM was calculated

on color images or grayscale images. Intuitively, the value of MAE should lie between 0 and 1 since we are averaging over the absolute differences between pixel intensities and the maximum absolute difference per pixel can only be in the range [0,1]. However, their value for MAE is in the hundreds.
The dataset consists of 229 sets of images. Each set contains 9 unedited under/overexposed images of the same scene. 13 potential label images were created using different multi-exposure image fusion and HDR imaging algorithms. The best image is chosen as the label based on subjective experiments as the reference image of each scene. We evaluated the PSNR, SSIM, and MAE of the enhanced form of each under/overexposed image compared to the reference image. Their model and weights were picked directly from their GitHub repository but no evaluation implementation was found. We used an evaluation procedure based on what we could infer from the paper. The SSIM and PSNR values were calculated with the scikit image metrics package with default parameters of the functions. The SSIM was evaluated on color images by setting the multichannel parameter to True. The MAE was calculated as the difference between corresponding pixels in both images averaged over the number of pixels and color channels.

4. Experimental Results

Ablation study

1. Overexposure compensation loss

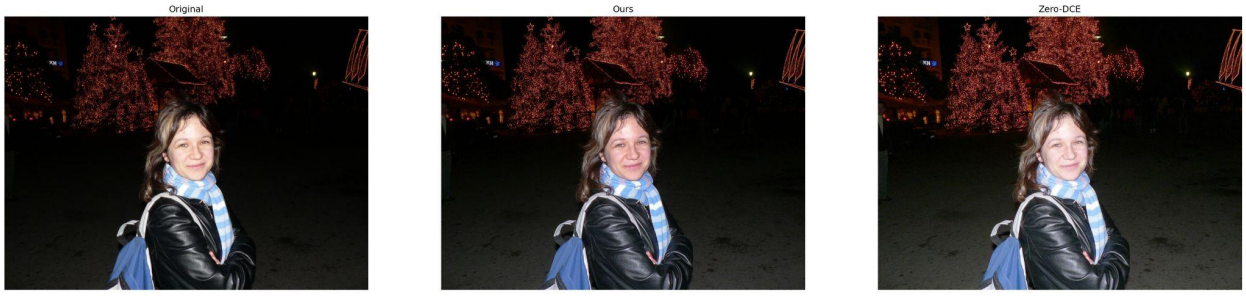

Fig. 1: Side-by-side comparison of a sample from the dataset(left), processed with an improved model trained with overexposure control loss DCE(center) and the original DCE output(right)

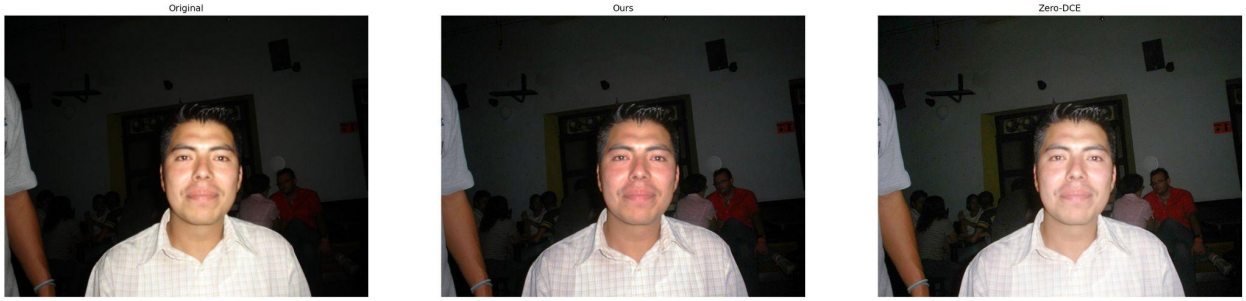

Fig. 2: Side-by-side comparison of a sample from the dataset(left), processed with an improved model trained with overexposure control loss DCE(center) and the original DCE output(right)

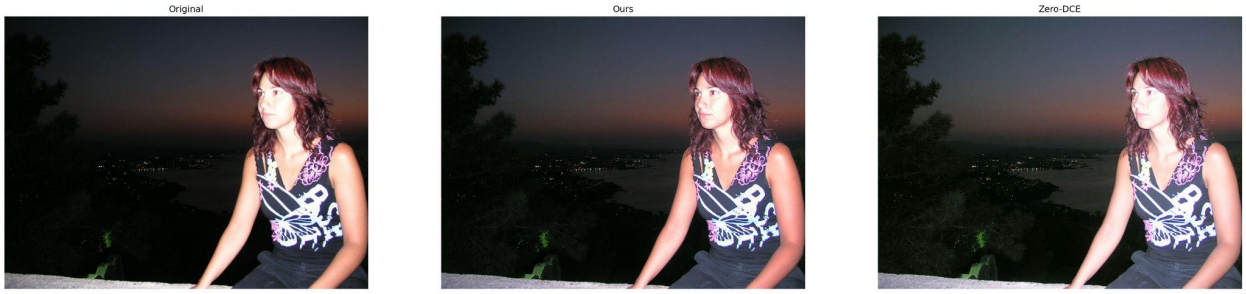

Fig. 3: Side-by-side comparison of a sample from the dataset(left), processed with an improved model trained with overexposure control loss DCE(center) and the original DCE output(right)

In Figures 1, 2, and 3, the human subject in the foreground is overexposed while the background is underexposed. The default model on the right further washes out the images, brightening even the already bright regions. Our model, on the other hand, compensates for the excess brightness. It is also to some extent, able to recover details and reduce the effect of the harsh flash used. The overexposure control loss prevents clipping and reduces the intensity in the excessively bright regions. In all three examples above, our model trained with overexposure control loss results in more natural skin tones and compensates for the effect of harsh flash that illuminates only the foreground.

2. Global contrast

Global contrast loss is used so that the image makes full use of the available dynamic range. In effect, global contrast is simply the variance of the pixel intensities. The global contrast loss improved the image by retaining high-quality shadows that still looked dark while improving visibility. This is visible in Fig 4 where the trees in the shadow region on the bottom left are still dark enough that we can see the shadow on them but bright enough to see the details. In Fig 5, showing the Eiffel tower, global contrast loss ensures that the building in the foreground is not too bright. Without global contrast, the building in the foreground is so bright that it takes attention away from the subject and the image ends up looking unreal. With global contrast, we can still tell that the building is in the shadow of the light cast by the tower. In Fig 6, the color of the sky remains dark and it retains its hues instead of getting washed out. The regions in shadow on the wall are bright enough to see detail but not too bright.

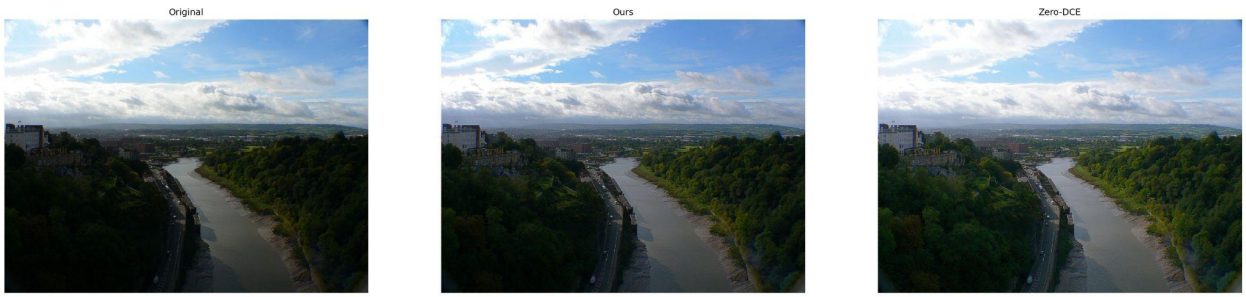

Fig. 4: Side-by-side comparison of a sample from the dataset(left), processed with an improved model that uses global contrast loss DCE(center) and the original DCE output(right)

Fig. 5: Side-by-side comparison of a sample from the dataset(left), processed with an improved model that uses global contrast loss DCE(center) and the original DCE output(right)

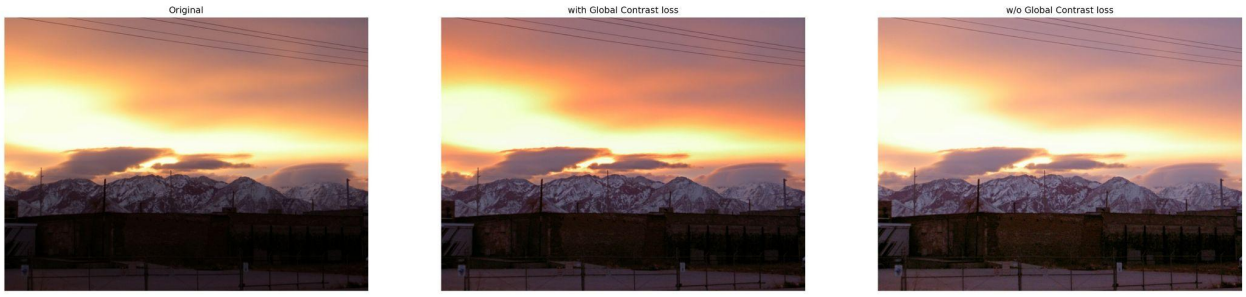

Fig. 6: Side-by-side comparison of a sample from the dataset(left), processed with an improved model that uses global contrast loss DCE(center) and the original DCE output(right)

The Impact of the dataset

Since our focus was to improve the performance on face images, we also study the importance of training the model specifically on faces to improve performance. The original model is trained on a set of low-light images from the SICE dataset. We further train the model on the LFW dataset [5] while applying data augmentation to reduce and increase the brightness(gamma) of the image. The results show that the original model washes out the skin tones. The retrained model, which is still not perfect, tends to deal with human skin tone better. Admittedly, the LFW dataset that we used consists mainly of Caucasian models and our model performs better on lighter skin tones. However, we believe that with a choice of better data, the model can work well across a variety of skin tones.

Timing Analysis

We tested the model on a Tesla T4 GPU. This is a low-cost, high-efficiency GPU for the data center primarily focused on light inference workloads. Its performance is comparable with a GTX 1650 that you find in gaming laptops. Additionally, the testing was performed on Google

Colab using an iPython kernel. This resulted in large variances in run time. No optimization techniques like converting to an ONNX model, low precision calculations were used.

We compare the performance of the model to YOLOv5 because it is considered a real-time model when run on GPU. However, it is not a fully convolutional network. Its output is not an image but a simple vector. Due to these differences, it looks like the time for inference time increases linearly for both models but the rate of increase of our model is 1.5x that of Yolov5. Given this constraint, our results are on par with the expected results from the Zero-DCE paper.

YOLOv5 boasts an inference time of 2ms on Nvidia V100 with a 640x640 image. For the sake of an estimate, if we extrapolate from this, our model should take about 6ms on a V100 at the same resolution. Obviously, there is more to it than that and this is only a vague estimate. For reference, the authors of the Zero-DCE 2.5ms on a GTX 2080Ti and we expect our run time to be very similar.

Image Size	**Time for inference**	**Frames per second**	**Reference faster YOLOv5 run times**
256x256	14.3ms	69.9	15.4
480p (852×480)	72.5ms	13.8	24ms
720p (1280x720)	152ms	6.6	51ms
1080p (1920x1080)	290ms	3.45	100ms

Quantitative results
As mentioned before, our evaluation methodology may be different from the Zero-DCE implementation and our values vary accordingly.
Below, we report our results for both models.

Model	**PSNR (dB) (higher is better)**	**SSIM (higher is better)**	**MAE (Lower is better)**
Zero-DCE default	14.18	0.6491	0.1932
Ours	14.47	0.6297	0.1718

The PSNR of our model is greater, suggesting that we add lesser noise compared to the default implementation. We hypothesize that this is the effect of penalizing very high pixel intensities and promoting global contrast. This counteracts the addition of excess brightness resulting in

noise in dark regions of the image. The MAE also appears to be better because we mostly try to improve extremely over/underexposed regions while minimizing the overall change to the image. The SSIM score compares images based on the similarity in luminance, contrast, and structure within each region of the image. The default model still performs better when it comes to SSIM scores.

5. Conclusion and Future Work

In this project, we designed and implemented an improvement on top of the originally proposed DCE model so that it can improve an image captured under low light conditions or overexposed one. We found that the system performed remarkably well *quantitatively* on our dataset and the inference time is less than 15ms for low-resolution images making it appropriate for real-time or video applications. For example, the live feed of a person over a video call is small in group meetings or when a screen is being shared. In such situations, it is possible to improve the quality in real-time.

It can also be used for photo and video editing by re-training the model for a few iterations while specifying the desired weightage value of the different losses. While the images may still require human intervention, the model can make a good first pass. Because we generate the curves and apply them to the model, the original image can remain unedited until the editor approves of the curves to be applied.

Future work can improve the model for video applications where the correction curves are applied to all the frames of the video. The difference is that the correction curves cannot change drastically from frame to frame and using a recurrent layer can help produce consistent correction curves.

Acknowledgments

We thank our Prof. Belhumeur Peter for his support throughout the duration of this course project.

Bibliography

1. Guo, Li et al. “Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement”
2. Cai, Gu, et al. “Learning a deep single image contrast enhancer from multi-exposure image”
3. Jiang, Gong et al. “Deep light enhancement without paired supervision”
4. Wang, Zhang et al. “Underexposed photo enhancement using deep illumination estimation”
5. Huang et al. “Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments”(LFW)

6. Lee et al. "Contrast enhancement based on layered difference representation" (DICM)
7. Vonikakis et al. "Multi-Scale Image Contrast Enhancement" (VV)
8. Wang et al. "Multiscale structural similarity for image quality assessment."
9. Cai et al. "Learning a Deep Single Image Contrast Enhancer from Multi-Exposure Images"

Appendix:

Loss functions that did not work

Other than the experiments covered above two other loss functions were tried. However, they did more harm than good.

Color temperature consistency loss

A loss function was created to maintain the consistency of color temperature. Color temperature is a measure of what looks white in your image. Red-yellow lights have lower temperatures while blue-purple lights have higher temperatures. If the things that look white in real life don't look white in the image, the temperature has changed. The images have either an orange or blue hue to them. In some cases, we found that the color temperature of the enhanced image was different from the input. The loss function we created penalized the model if the normalized difference between any two color channels in the input was different from the normalized difference between any two color channels in the output. It turns out that only some colors need to be boosted to improve the overall brightness in most cases. For example, in a picture of trees, only the intensity of green color rises when brightened. This loss did not make sense and resulted in a color jitter effect being added to the enhanced image.

A loss function was created to improve the local contrast in addition to the previously mentioned global contrast. The loss function tried to maximize the variance within a 16x16 region of the image. This resulted in artifacts and noise in the simpler parts of the enhanced image. The intuition behind this is that some areas of the image like a sky or a simple plain background have inherently low contrast. Adding contrast only added noise to that patch of the image.

Code

Our code in red. Reference code in green

```
#Part 0 - Load Dataset and Libraries
"""
```

```
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)
datadir = "/content/gdrive/MyDrive/DL_harris/data/"

"""## Right click on the shared directory and "Add shortcut"

Try to find this directory and change the datadir below.
"""

import torch
import torch.nn as nn
import torch.nn.functional as F
import math
from torchvision.models.vgg import vgg16
import numpy as np
import os
import sys
import torch.utils.data as data
from PIL import Image
import glob
import random
import cv2
import torchvision
import torch.backends.cudnn as cudnn
import torch.optim
import argparse
import time
from torchvision import transforms
import time
import matplotlib.pyplot as plt

"""# Model Definition"""

class enhance_net_nopool(nn.Module):
        def __init__(self):
                super(enhance_net_nopool, self).__init__()
                self.relu = nn.ReLU(inplace=True)

                number_f = 32
                self.e_conv1 = nn.Conv2d(3,number_f,3,1,1,bias=True)
```

```
        self.e_conv2 = nn.Conv2d(number_f,number_f,3,1,1,bias=True)
        self.e_conv3 = nn.Conv2d(number_f,number_f,3,1,1,bias=True)
        self.e_conv4 = nn.Conv2d(number_f,number_f,3,1,1,bias=True)
        self.e_conv5 = nn.Conv2d(number_f*2,number_f,3,1,1,bias=True)
        self.e_conv6 = nn.Conv2d(number_f*2,number_f,3,1,1,bias=True)
        self.e_conv7 = nn.Conv2d(number_f*2,24,3,1,1,bias=True)

    def forward(self, x):
        x1 = self.relu(self.e_conv1(x))
        x2 = self.relu(self.e_conv2(x1))
        x3 = self.relu(self.e_conv3(x2))
        x4 = self.relu(self.e_conv4(x3))
        x5 = self.relu(self.e_conv5(torch.cat([x3,x4],1)))
        x6 = self.relu(self.e_conv6(torch.cat([x2,x5],1)))

        x_r = torch.tanh(self.e_conv7(torch.cat([x1,x6],1)))
        r1,r2,r3,r4,r5,r6,r7,r8 = torch.split(x_r, 3, dim=1)

        x = x + r1*(torch.pow(x,2)-x)
        x = x + r2*(torch.pow(x,2)-x)
        x = x + r3*(torch.pow(x,2)-x)
        enhance_image_1 = x + r4*(torch.pow(x,2)-x)
        x = enhance_image_1 + r5*(torch.pow(enhance_image_1,2)-enhance_image_1)

        x = x + r6*(torch.pow(x,2)-x)
        x = x + r7*(torch.pow(x,2)-x)
        enhance_image = x + r8*(torch.pow(x,2)-x)
        r = torch.cat([r1,r2,r3,r4,r5,r6,r7,r8],1)
        return enhance_image_1,enhance_image,r

"""# Preparing Data

## Data Augmentation
"""

def vary_gamma(img, gamma_range = (2,2.6), probability = 0.3):
    x = np.random.rand()
    if x < probability:
        gamma = np.random.rand()*(gamma_range[1] - gamma_range[0]) + gamma_range[0]
        gamma_inc = (img**(1/gamma) ) ** 2.2
```

```
        return gamma_inc

    return img

"""## PyTorch Dataset"""

random.seed(1143)
def populate_train_list(lowlight_images_path):
        image_list_lowlight = glob.glob(lowlight_images_path + "/**/*.jpg", recursive=True)
        image_list_lowlight = image_list_lowlight + glob.glob(lowlight_images_path +
"/**/*.JPG", recursive=True)
        train_list = image_list_lowlight
        random.shuffle(train_list)
        return train_list

class lowlight_loader(data.Dataset):
        def __init__(self, lowlight_images_path):
                self.train_list = populate_train_list(lowlight_images_path)
                self.size = 256
                self.data_list = self.train_list
                print("Total training examples:", len(self.train_list))

        def __getitem__(self, index):
                data_lowlight_path = self.data_list[index]
                data_lowlight = Image.open(data_lowlight_path)
                data_lowlight = data_lowlight.resize((self.size,self.size), Image.ANTIALIAS)
                data_lowlight = (np.asarray(data_lowlight)/255.0)
                data_lowlight = vary_gamma(data_lowlight, probability=0.2)
                data_lowlight = torch.from_numpy(data_lowlight).float()
                return data_lowlight.permute(2,0,1)

        def __len__(self):
                return len(self.data_list)

"""# *Defining* Loss Functions

###Color Loss Function
"""

class L_color(nn.Module):
```

```
    def __init__(self):
        super(L_color, self).__init__()

    def forward(self, x ):
        b,c,h,w = x.shape
        mean_rgb = torch.mean(x,[2,3],keepdim=True)
        mr,mg, mb = torch.split(mean_rgb, 1, dim=1)
        Drg = torch.pow(mr-mg,2)
        Drb = torch.pow(mr-mb,2)
        Dgb = torch.pow(mb-mg,2)
        k = torch.pow(torch.pow(Drg,2) + torch.pow(Drb,2) + torch.pow(Dgb,2),0.5)
        return k

"""###Spatial Loss Function"""

class L_spa(nn.Module):
    def __init__(self):
        super(L_spa, self).__init__()
        kernel_left = torch.FloatTensor( [[0,0,0],[-1,1,0],[0,0,0]]).cuda().unsqueeze(0).unsqueeze(0)
        kernel_right = torch.FloatTensor(
[[0,0,0],[0,1,-1],[0,0,0]]).cuda().unsqueeze(0).unsqueeze(0)
        kernel_up = torch.FloatTensor( [[0,-1,0],[0,1, 0
],[0,0,0]]).cuda().unsqueeze(0).unsqueeze(0)
        kernel_down = torch.FloatTensor( [[0,0,0],[0,1,
0],[0,-1,0]]).cuda().unsqueeze(0).unsqueeze(0)
        self.weight_left = nn.Parameter(data=kernel_left, requires_grad=False)
        self.weight_right = nn.Parameter(data=kernel_right, requires_grad=False)
        self.weight_up = nn.Parameter(data=kernel_up, requires_grad=False)
        self.weight_down = nn.Parameter(data=kernel_down, requires_grad=False)
        self.pool = nn.AvgPool2d(4)

    def forward(self, org , enhance ):
        b,c,h,w = org.shape
        org_mean = torch.mean(org,1,keepdim=True)
        enhance_mean = torch.mean(enhance,1,keepdim=True)
        org_pool =  self.pool(org_mean)
        enhance_pool = self.pool(enhance_mean)
        weight_diff = torch.max(
            torch.FloatTensor([1]).cuda() + 10000*torch.min(org_pool -
torch.FloatTensor([0.3]).cuda(),torch.FloatTensor([0]).cuda()),
```

```
            torch.FloatTensor([0.5]).cuda())

        E_1 = torch.mul(torch.sign(enhance_pool - torch.FloatTensor([0.5]).cuda())
,enhance_pool-org_pool)
        D_org_letf = F.conv2d(org_pool , self.weight_left, padding=1)
        D_org_right = F.conv2d(org_pool , self.weight_right, padding=1)
        D_org_up = F.conv2d(org_pool , self.weight_up, padding=1)
        D_org_down = F.conv2d(org_pool , self.weight_down, padding=1)

        D_enhance_letf = F.conv2d(enhance_pool , self.weight_left, padding=1)
        D_enhance_right = F.conv2d(enhance_pool , self.weight_right, padding=1)
        D_enhance_up = F.conv2d(enhance_pool , self.weight_up, padding=1)
        D_enhance_down = F.conv2d(enhance_pool , self.weight_down, padding=1)

        D_left = torch.pow(D_org_letf - D_enhance_letf,2)
        D_right = torch.pow(D_org_right - D_enhance_right,2)
        D_up = torch.pow(D_org_up - D_enhance_up,2)
        D_down = torch.pow(D_org_down - D_enhance_down,2)
        return D_left + D_right + D_up + D_down

"""###Exposure Loss Function"""

class L_exp(nn.Module):
    def __init__(self,patch_size,mean_val):
        super(L_exp, self).__init__()
        self.pool = nn.AvgPool2d(patch_size)
        self.mean_val = mean_val

    def forward(self, x ):
        x = torch.mean(x,1,keepdim=True)
        mean = self.pool(x)
        d = torch.mean(torch.pow(mean- torch.FloatTensor([self.mean_val] ).cuda(),2))
        return d

class L_over_exp(nn.Module):
    def __init__(self,patch_size,mean_val):
        super(L_over_exp, self).__init__()
        self.pool = nn.AvgPool2d(patch_size)
        self.mean_val = mean_val
```

```
    def forward(self, x ):
        x = torch.mean(x,1,keepdim=True)
        mean = self.pool(x)
        d = torch.mean(torch.pow(torch.maximum(mean -
torch.FloatTensor([self.mean_val]).cuda(), torch.FloatTensor([0]).cuda()),2))
        return d

"""### Contrast loss (SSIM)"""

class L_global_cont(nn.Module):
    def __init__(self, var):
        super(L_global_cont, self).__init__()
        self.var = var

    def forward(self, enhance ):
        var = torch.var(enhance, dim=[2,3])
        d = torch.mean(torch.FloatTensor([self.var]).cuda() - var)
        return d

"""###TV Loss Function"""

class L_TV(nn.Module):
    def __init__(self,TVLoss_weight=1):
        super(L_TV,self).__init__()
        self.TVLoss_weight = TVLoss_weight

    def forward(self, x):
        batch_size = x.size()[0]
        h_x = x.size()[2]
        w_x = x.size()[3]
        count_h =  (x.size()[2]-1) * x.size()[3]
        count_w = x.size()[2] * (x.size()[3] - 1)
        h_tv = torch.pow((x[:,:,1:,:] - x[:,:,:h_x-1,:]),2).sum()
        w_tv = torch.pow((x[:,:,:,1:] - x[:,:,:,:w_x-1]),2).sum()
        return self.TVLoss_weight*2*(h_tv/count_h+w_tv/count_w)/batch_size

"""# Training
## Implement Training Function
"""
```

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)

    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)

def train(
    lowlight_images_path,
    lr=0.0001,
    weight_decay=0.0001,
    grad_clip_norm=0.2,
    num_epochs=5,
    train_batch_size=32,
    val_batch_size=4,
    num_workers=4,
    display_iter=30,
    snapshot_iter=60,
    snapshots_folder=datadir+"snapshots/",
    load_pretrain=False,
    pretrain_dir=datadir+"Epoch99.pth"):

    if not os.path.exists(snapshots_folder):
        os.mkdir(snapshots_folder)

    os.environ['CUDA_VISIBLE_DEVICES']='0'
    DCE_net = enhance_net_nopool().cuda()
    DCE_net.apply(weights_init)
    if load_pretrain == True:
        DCE_net.load_state_dict(torch.load(pretrain_dir))
    train_dataset = lowlight_loader(lowlight_images_path)

    train_loader = torch.utils.data.DataLoader(train_dataset,
                                               batch_size=train_batch_size,
                                               shuffle=True, num_workers=num_workers,
                                               pin_memory=True)
    l_color = L_color()
    l_ct = L_color_temp()
```

```
    l_spa = L_spa()
    l_o_exp = L_over_exp(16,0.3)

    l_exp = L_exp(64,0.5)
    l_TV = L_TV()
    l_gc = L_global_cont(0.15)

    optimizer = torch.optim.Adam(DCE_net.parameters(), lr=lr, weight_decay=weight_decay)
    DCE_net.train()
    try:
        for epoch in range(num_epochs):
            epoch_loss = np.zeros(6)
            for iteration, img_lowlight in enumerate(train_loader):
                img_lowlight = img_lowlight.cuda()
                enhanced_image_1,enhanced_image,A  = DCE_net(img_lowlight)
                loss_TV = 200*l_TV(A)
                loss_spa = 1*torch.mean(l_spa(enhanced_image, img_lowlight))
                loss_col = 5*torch.mean(l_color(enhanced_image))
                loss_o_exp = 3*torch.mean(l_o_exp(enhanced_image))
                loss_exp = 9*torch.mean(l_exp(enhanced_image))
                loss_gc = torch.mean(l_gc(enhanced_image))
                # loss_sim = F.mse_loss(enhanced_image, img_lowlight)
                epoch_loss += np.array([x.data.cpu().numpy() for x in [loss_TV, loss_spa, loss_col,
loss_exp, loss_gc, loss_o_exp]])

                loss =  loss_TV + loss_spa + loss_col + loss_exp + loss_gc + loss_o_exp

                optimizer.zero_grad()
                loss.backward()
                torch.nn.utils.clip_grad_norm_(DCE_net.parameters(),grad_clip_norm)
                optimizer.step()

                if ((iteration+1) % display_iter) == 0:
                    print("Cumulative Loss at iter", iteration+1, ":", epoch_loss/iteration)

            print("Epoch ", epoch, "\tLoss= ", epoch_loss)
            torch.save(DCE_net.state_dict(), snapshots_folder + "Epoch" + str(epoch) + '.pth')
    except:
        print("Epoch ", epoch, "\tLoss= ", epoch_loss)
        torch.save(DCE_net.state_dict(), snapshots_folder + "Epoch" + str(epoch) + '.pth')
```

"""## Start training Training"""

```
train(lowlight_images_path=datadir + "train_data/low_light",
      load_pretrain=True,
      pretrain_dir=datadir + "Epoch99.pth",
      snapshots_folder=datadir + "snapshots3/",
      num_epochs=10)
```

"""# Inference and Visualization"""

```
def lowlight_orig(image_path):
        os.environ['CUDA_VISIBLE_DEVICES']='0'
        data_lowlight = Image.open(image_path).convert("RGB")
        data_lowlight = (np.asarray(data_lowlight)/255.0)
        data_lowlight = torch.from_numpy(data_lowlight).float()
        data_lowlight = data_lowlight.permute(2,0,1)
        data_lowlight = data_lowlight.cuda().unsqueeze(0)

        DCE_net = enhance_net_nopool().cuda()
        DCE_net.load_state_dict(torch.load(datadir+'Epoch99.pth'))
        enhanced_image1, enhanced_image,_ = DCE_net(data_lowlight)

        end_time = (time.time() - start)
        image_path = image_path.replace('test_data','result')
        result_path = image_path
        return enhanced_image, enhanced_image1

def lowlight_mod(image_path):
        os.environ['CUDA_VISIBLE_DEVICES']='0'
        data_lowlight = Image.open(image_path).convert("RGB")
        data_lowlight = (np.asarray(data_lowlight)/255.0)
        data_lowlight = torch.from_numpy(data_lowlight).float()
        data_lowlight = data_lowlight.permute(2,0,1)
        data_lowlight = data_lowlight.cuda().unsqueeze(0)

        DCE_net = enhance_net_nopool().cuda()
        DCE_net.load_state_dict(torch.load(datadir+'snapshots3/Epoch9.pth'))
        enhanced_image1,enhanced_image,_ = DCE_net(data_lowlight)
```

```
            end_time = (time.time() - start)
            image_path = image_path.replace('test_data','result')
            result_path = image_path
            return enhanced_image, enhanced_image1

visualization = False
with torch.no_grad():
    test_list = populate_train_list(datadir+"test_data")
    test_list.sort()

    for j, image in enumerate(test_list):
        # image = image
        plt.figure(figsize=(30,10), dpi=150)
        print(image, j)
        orig_image = cv2.imread(image)
        orig_image = cv2.cvtColor(orig_image, cv2.COLOR_RGB2BGR)
        plt.subplot(1,3,1)
        plt.axis('off')
        plt.title("Original")
        plt.imshow(orig_image)

        if visualization:
            color = ('b','g','r')
            for i,col in enumerate(color):
                histr = cv2.calcHist([(orig_image).astype(np.uint8)],[i],None,[256],[0,256])
                plt.subplot(2,3,4)
                plt.plot(histr,color = col)
                plt.xlim([0,256])
            print(np.median(orig_image))

        _,enhanced_new = lowlight_mod(image)
        enhanced_new = enhanced_new.data.permute(0,2,3,1).cpu().numpy().squeeze()
        enhanced_new = (enhanced_new*255).astype(np.uint8)
        plt.subplot(1,3,2)
        plt.axis('off')
        plt.title("with Global Contrast loss")
        plt.imshow(enhanced_new)

        if visualization:
            color = ('b','g','r')
```

```
        for i,col in enumerate(color):
            histr = cv2.calcHist([enhanced_new],[i],None,[256],[0,256])
            plt.subplot(2,3,5)
            plt.plot(histr,color = col)
            plt.xlim([0,256])
        print(np.median(enhanced_new))

    _,enhanced_orig = lowlight_orig(image)
    enhanced_orig = enhanced_orig.data.permute(0,2,3,1).cpu().numpy().squeeze()
    enhanced_orig = (enhanced_orig*255).astype(np.uint8)
    plt.subplot(1,3,3)
    plt.axis('off')
    plt.title("w/o Global Contrast loss")
    plt.imshow(enhanced_orig)

    if visualization:
        color = ('b','g','r')
        for i,col in enumerate(color):
            histr = cv2.calcHist([enhanced_orig],[i],None,[256],[0,256])
            plt.subplot(2,3,6)
            plt.plot(histr,color = col)
            plt.xlim([0,256])
        print(np.median(enhanced_orig))

    if visualization:
        plt.show()
    else:
        plt.savefig(datadir+f"test_set_outputs/test{j}.jpg", bbox_inches='tight')

    if j>20:
        break

"""# Evaluation"""

from skimage import metrics
from tqdm.notebook import tqdm

device = torch.device('cuda')
DCE_net = enhance_net_nopool().to(device)
DCE_net.load_state_dict(torch.load(datadir+'snapshots3/Epoch9.pth',map_location=device))
```

```
DCE_net.eval()
test_list = os.listdir(datadir+"test_data/Dataset_Part2/")
test_list.sort()
labels_list = os.listdir(datadir+"test_data/Dataset_Part2/Label")
labels_list.sort()
test_data_dir  =   datadir+"test_data/Dataset_Part2/"
len(test_list), len(labels_list)

psnr = []
mae = []
ssim  = []
with torch.no_grad():
    for folder, label in tqdm(zip(test_list, labels_list)):
        if folder not in label:
            print("folder and label mismatch")
            break
        for image_name in os.listdir(test_data_dir+folder):
            if not ("PNG" in image_name or \
                    "JPEG" in image_name or \
                    "JPG" in image_name or \
                    "png" in image_name or \
                    "PNG" in image_name):
                continue
            image_path =  f"{test_data_dir}/{folder}/{image_name}"
            label_path =  f"{test_data_dir}/Label/{label}"
            data_lowlight = Image.open(image_path).convert("RGB")
            data_lowlight = data_lowlight.resize((1200,900), Image.ANTIALIAS)
            data_lowlight = (np.asarray(data_lowlight)/255.0)

            data_label = Image.open(label_path).convert("RGB")
            data_label = data_label.resize((1200,900), Image.ANTIALIAS)
            data_label = (np.asarray(data_label)/255.0).astype(np.float32)

            data_lowlight = torch.from_numpy(data_lowlight).float()
            data_lowlight = data_lowlight.permute(2,0,1)
            data_lowlight = data_lowlight.unsqueeze(0).to(device)
            _,enhanced_image,_ = DCE_net(data_lowlight)
            enhanced_image  =  enhanced_image.data.cpu().permute(0,2,3,1).numpy().squeeze()
```

```
        mae.append(np.mean(np.abs(enhanced_image-data_label)))
        psnr.append(metrics.peak_signal_noise_ratio(data_label, enhanced_image))
        ssim.append(metrics.structural_similarity(data_label, enhanced_image,
multichannel=True))
print(np.mean(np.array(psnr)))
print(np.mean(np.array(ssim)))
print(np.mean(np.array(mae)))

np.save(datadir+"psnr_ours.npy",  np.array(psnr))
np.save(datadir+"ssim_ours.npy",  np.array(ssim))
np.save(datadir+"mae_ours.npy",  np.array(mae))
```