CSCE 748 - Computational Photography - Project

Name: Hema Deva Sagar Potala
UIN: 133001118

The github repository for the project is found here.
Reference to the paper on which this implementation is based on is here
Title: Low Light Enhancement Using Zero-Reference Deep Curve Estimation

1) Introduction

Most photos, especially with the growth of mobile cameras, captured suffer compromised aesthetic quality because of incorrect or not optimal lighting conditions. These sub-optimal conditions are mostly because of the lack of awareness about lighting strategy, which is the case with non-professional or laymen photographer. To overcome this, we can process these low light images with low light enhancing techniques that will improve the quality of the original picture. In this project, I explored one such technique from [1]. The method explored in this project, formulates light enhancement as a task of image specific curve estimations using a light weight deep neural network. During the rest of the report I will go through, briefly, the model details, loss functions, challenges faced while training, results and finally some concluding thoughts.

2) Methodology

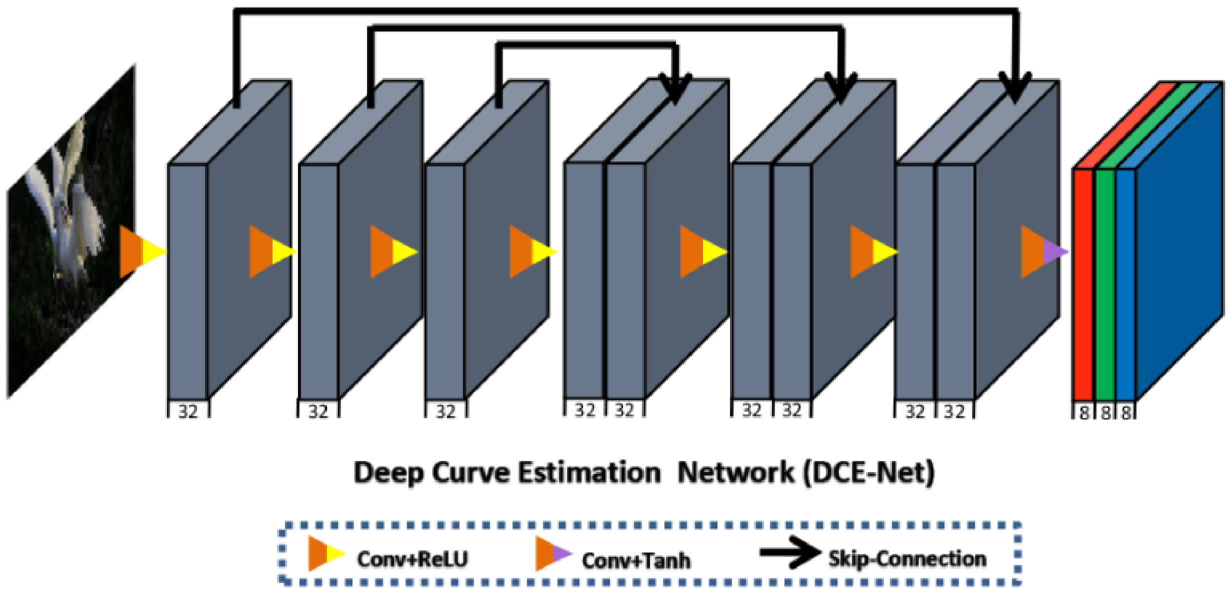
Like mentioned above, the method formulates light enhancement as a task of image specific curve estimation, known as light enhancement curves, using deep neural network. To be more precise this network is trained to estimate pixel-wise high order curves to adjust the dynamic range of the given image considering pixel range, monotonicity and differentiability. Unlike other neural based low light enhancement methods, like [2], [3], [4] and [5], this method doesn't require pair-wise training data. Rather, it uses carefully formulated non-reference loss functions, which implicitly measure the enhancement quality and drive the learning of the network.

3.1 Light Enhancement Curve: Before going into the details of the neural architecture and non-reference loss functions used here, I would like to take a step back and explain briefly the conceptual backdrop for designing the neural model for enhancement. Light enhancement curves are inspired from curves that are frequently used in photo editing to adjust the pictures, by operating at pixel level. They try to map low light image to its enhanced image automatically. The neural model, we will discuss, essentially tries to model these light enhancement curves, given the input image. How are these curves used to enhance the image though? Below equation mathematically answers this question

$$LE_n(x) = LE_{n-1}(x) + A_n(x)LE_{n-1}(x)(1 - LE_{n-1}(x))$$

Where A_n is curve map that does pixel wise curving.
It is very apparent that this equation is recursive. Meaning to obtain n^{th} level enhanced image it needs $n-1^{th}$ enhanced image and n^{th} pixel level curve map. Now the goal is to estimate the $A's$, for which a deep learning architecture, called zero-reference DCE, is used for.

3.2 DCE-Net: The models learns to map the input, typically a low light image, to its best fitting curve parameter map. The architecture is a plain CNN assemble of seven convolution layers with symmetrical concatenation. Each layer consists of 32 convolutional kernels of size 3×3 and stride 1 followed by ReLU activation. No down sampling or batch normalization is done since they break the relation among the neighbouring pixels. Below is the visual representation of the model architecture.



The model outputs 24 curve maps with which we treat the input through 8 iterations, higher order enhancement, to obtain the final version of the enhanced image. The advantage with this architecture is that it is very light weight with only 79, 416 trainable parameters and while training, we can resize the input to 256 x 256 or much lower if needed. And the resultant model can be used to enhance image of any size and not just 256 x 256.

3.3 Non-Reference Loss: As mentioned earlier, one of the advantage of this model is it doesn't require any labelled data. Instead non reference losses are used to drive the learning of the model. There are 4 such losses, which are combined during the back-propagation, that are used here. All of these 4, individually or combined, helps the models to learn in generating enhanced images that have most of the original characteristics of preserved in some way. Since it is redundant to explain these loss functions in detail, I would give a brief about them and write the math of those equations.

- Spatial Consistency Loss: This loss makes sure that the enhanced image have spatial coherence as it was in the original image. Below is the equation for calculating this loss

$$L_{spa} = \frac{1}{K} \sum_{i=1}^{K} \sum_{j \in \Omega(i)} (|Y_i - Y_j| - |I_i - I_j|)^2$$

  where Y and I are the average intensities of the enhanced and actual images and K is the number of non -overlapping local regions. Local regions are of size 4×4 here.

- Exposure Control Loss: This loss prevents the enhanced image from over exposing or under exposing. Below is the equation for the loss

$$L_{exp} = \frac{1}{M}\sum_{k=1}^{M} |Y_k - E|$$

where $E = 0.6$, refer to [6] and [7] for this, and Y is average intensity of local regions of the enhanced image. Here local region is of size 16×16

- Color Constancy Loss: This loss tries to maintain the balance of the color tonality in the enhanced version, and its intuition is based on the Gray-world color constancy hypothesis described in [8]. Below is the equation for the calculating this loss

$$L_{col} = \sum_{\forall (p,q) \in \epsilon} (J^p - J^q)^2$$

where $\epsilon = \{$(R, G),(R, B),(G, B)$\}$ and J^p is average intensity for channel p in the enhanced image.

- Illumination Smoothness Loss: This loss tries to preserve the smoothness of the color tonality with respect to the neighbouring pixels. Below is the loss function

$$L_{tv_A} = \frac{1}{N}\sum_{n=1}^{N}\sum_{c\in\epsilon}(\Delta_x A_n^c)^2 + (\Delta_y A_n^c)^2$$

Where $\epsilon = \{$R, G, B$\}$ and N is total number of pixel positions across all curve maps. Note: This equation seems to be a bit different than what was mentioned in the original paper, because the authors seems to have slightly updated it.

So the total loss is weighted combination of the above losses.

$$L_{total} = w_{spa}L_{spa} + w_{exp}L_{exp} + w_{col}L_{col} + w_{tv_A}L_{tv_A}$$

After looking at the math of individual losses, it is very apparent that the order of these losses will be different from each other. Hence the search for weights associated with each loss turned out be much harder than I imagined.

3) Modeling Details:

The dataset used for training the model is part1 of SICE dataset[9], which have 360 degree multi exposure sequence of images. There are 3022 in total. To impart the model with a wide range of dynamic range capability, this dataset is choosen, because it have both low-light and over-exposed images. The dataset is randomly divided into 2422 and 600 images for training and testing respectively.

3.1) Implementation Issues:

There are few implementation issues I faced based on my GPU limitation and data-loader design. I tried to have higher batch size (like 16 images per batch) to make the model robust, but the gpu ran into out-of-memory error multiple times and the maximum batch size that I am able to use without facing any memory issues is 8 images per batch (258 x 256 image size for training). Although the trainable parameters for the model is less, spatial dimensions of the the activation maps across different layers says the same as the input image (256 x 256, in this case) while the number of channels/depth stay at 32 (and 64 before the last layer). Which is unlikely in a normal CNN models. I think this could be the primary reason why the run time memory occupied on the GPU during the feed forward shoot up for higher batch sizes.
Another issue that I faced was the model training taking too much time. When I traced what the bottleneck is for this shoot up time, I found that doing resize with anti-aliasing in torch is taking too much than pillow library's resize. So I had to edit the data loader and treat the images with non-torch based image libraries for processing. After this change, the training time halved.

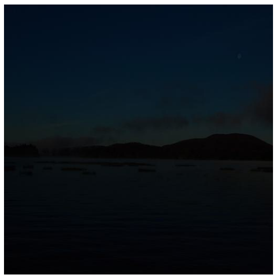
3.2) Issues with loss scaling:

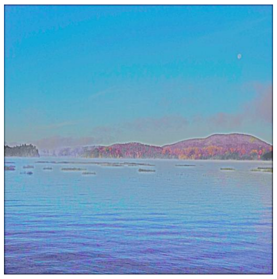
As mentioned above, and looking at the loss equation written, we can easily guess that the scales of the losses are very different for different losses. So the individual losses have to be weighted before back-propagation.
The issue I faced was the recommended weights in the paper didn't seem to give good results for me. So I had to try out different weight combinations for different losses until I started getting decent quality output. I had to visually inspect the output to understand the model status (for a particular loss weights combination), because just looking at total loss don't seem to give complete understanding about the quality of the model. Like for example, the total loss keeps decreasing with every iteration but the model keeps giving not so good results qualitatively. This was the case with most of the bad combinations I tried with. Figure 1 shows some examples of the outputs I got from the models which were trained for 300 iterations with different loss weights combination. From figure (1), We can see that (b), (c), (d) seems to be very bad quality enhancements of the input (a). For all of these models the training conditions are the same except for loss weights. Finally the weights used for image (e) seems to give a very stable and good quality output. I think that is evident from the image (e) it-self. Note: I am judiciously ignoring the outputs from several other bad loss weights combinations I tried with.

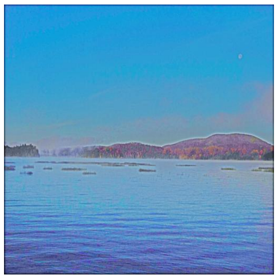
3.3) Model training:

Once I got an initial set of loss weights, I trained the model (with those weights) for nearly 10000 iterations and logged their outputs. Figure 2 shows the progress of total loss and different individual raw losses as the training progressed. Image (a) in figure 2, is the total loss and rest all are individual raw losses. We can see most of the loss (including total loss) reduces very quickly to a stable point and then hovers around similar values for the later iterations of the training, except for Spatial Consistency loss and illumination smootheness loss. While Spatial consistency loss raises to a value and stabilizes there, illumination smoothness loss seems to be gradually increasing as training goes on. Like mentioned in the earlier sections, illumination smoothness loss preserve the monotonicity relations between neighboring pixels and the fact that this loss is increasing gradually with training may explain some of the anamolous results I observed.

3.4) Results


(a) original input


(b) $w_{tv_A} = 200, w_{col} = 5$
$w_{exp} = 2.5, w_{spa} = 1$


(c) $w_{tv_A} = 400, w_{col} = 5$
$w_{exp} = 3, w_{spa} = 1$

(d) $w_{tv_A} = 200, w_{col} = 5$
$w_{exp} = 1, w_{spa} = 1$

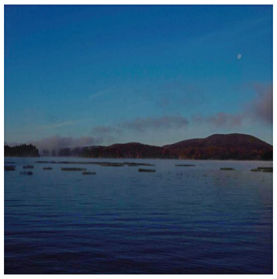

(e) $w_{tv_A} = 200, w_{col} = 5$
$w_{exp} = 10, w_{spa} = 1$

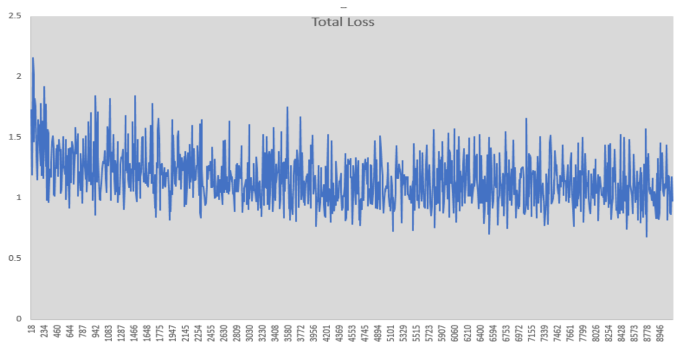
Figure 1: outputs from models trained with different loss weights

Since the model we trained uses non-reference loss and no pair data, there isn't any quantitative metric that can be calculated over the output to quantify the performance of the model. Hence we rely on the qualitative perception of the enhanced images from the model(s). Figure 3 shows the input images and the corresponding output enhanced image from the stable model (checkpoint at 3000 iterations). As we can see the model does a good job enhancing the images very realistically. These are some good example of the model performance.

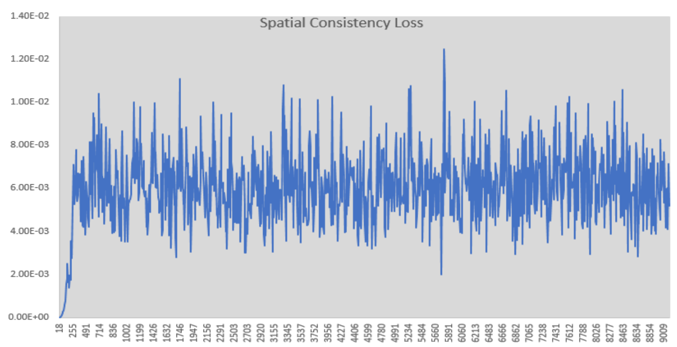
Like every model have its limitations, this model also seem to have limitations. Figure 4 shows the models performance on some bad examples, and it is very evident that the model either does a bad job or no job at all. For Images (a) and (b) in figure 4, the model did enhanced it but at the cost of adding too much noise. I wanted to see, given that the model is low light enhancer, how it behaves if an overexposed image is given as input. Image c (figure 4) is an example of over exposed image and the model further deteriorates the image by enhancing it's exposure to a much higher value.

Another observation I found is, the quality of the enhanced image improves with the number of training iterations up to a certain point, after that the enhanced image seem to decline in the quality subtly. This might be traced back to the increasing of illumination smoothness Loss, we saw in Figure 2, image (e). Figure 5 shows some example that I output during different iterations of training. We can see from images (a) & (b), the enhanced version at 9000 iteration seems to be a bit washed out (less saturation) compared to one from 3000 iterations. Also color tonality deviation from original image seems to become significant as the training iterations increase. From my observations, I considered the model after 3000 iteration as stable. Results from that model is what presented in Figures 3 & 4.

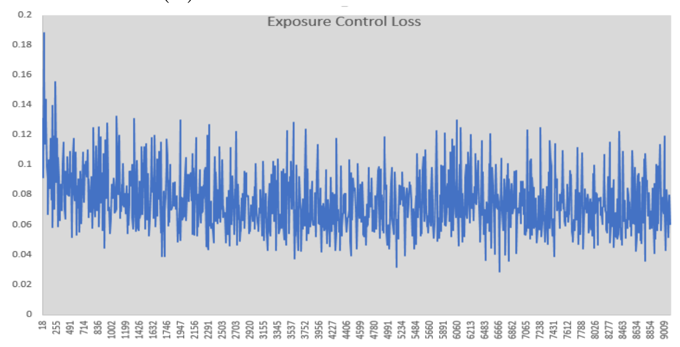
4) Conclusion:



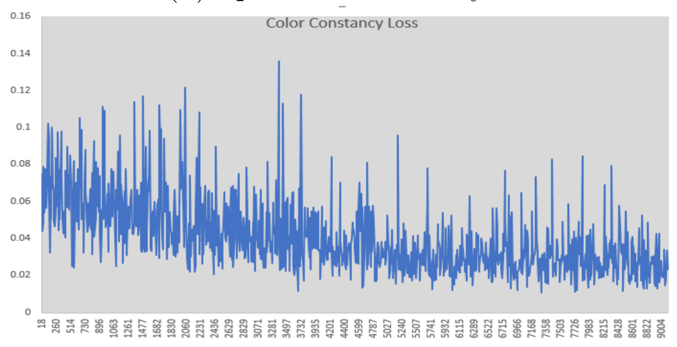
(a) Total Loss



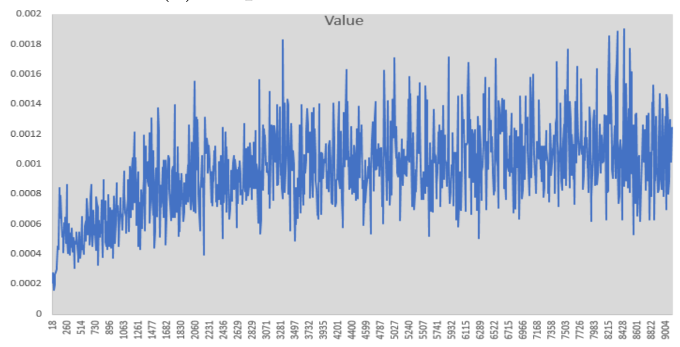
(b) Spatial Consistency loss



(c) Exposure Control Loss

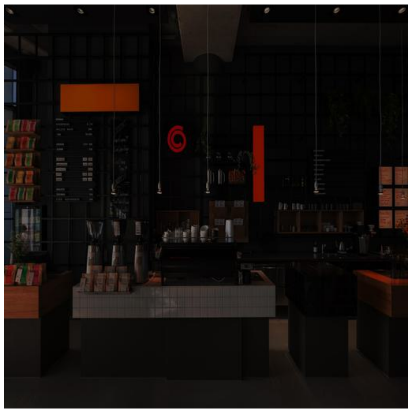


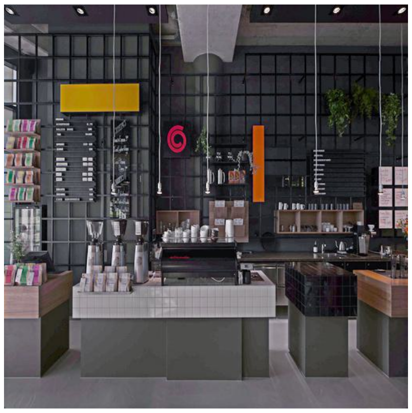
(d) Color Constancy Loss

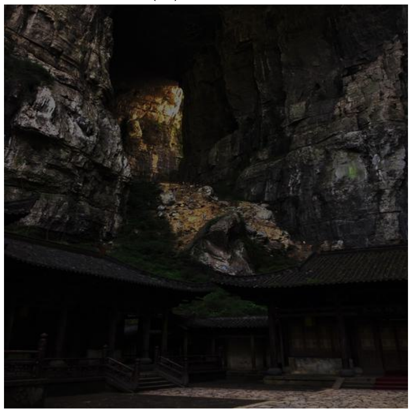


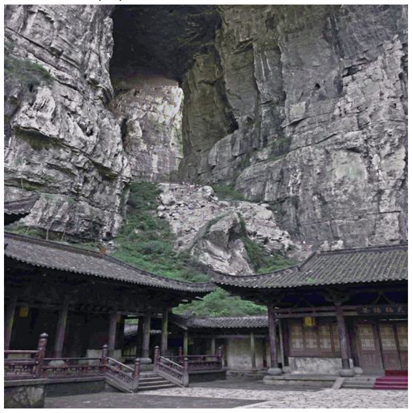
(e) Illumination Smootheness Loss

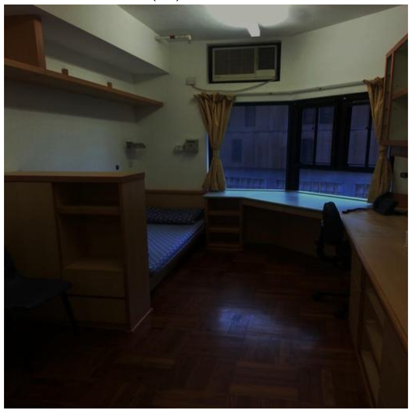
Figure 2: Progression of various losses during 10000 iterations of training


(a) Input


(a) Model output


(b) Input


(b) Model output


(c) Input

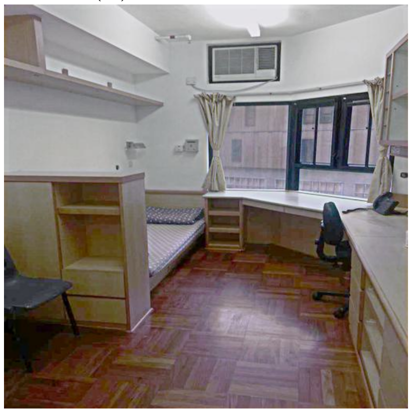
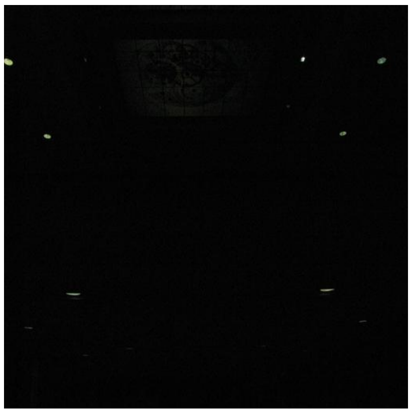
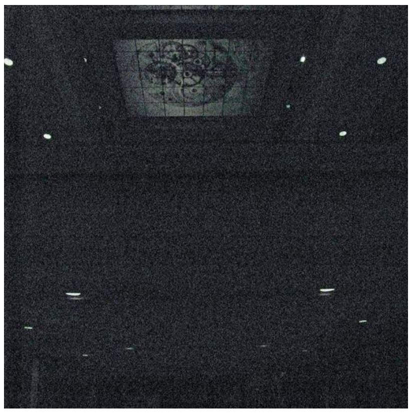
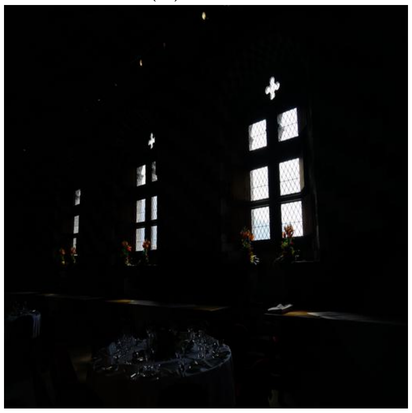
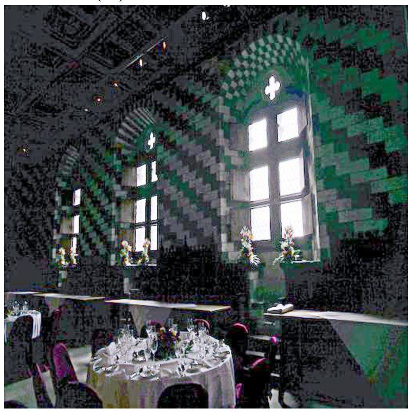

(c) Model output

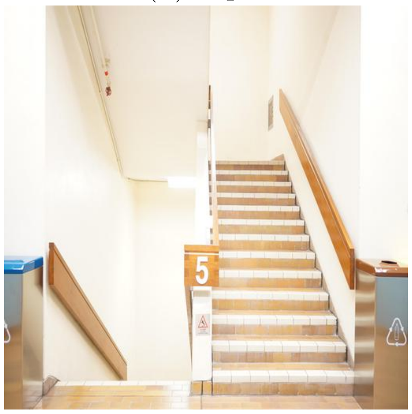
Figure 3: Model outputs for some good examples


(a) Input


(a) Model output


(b) Input


(b) Model output


(c) Input

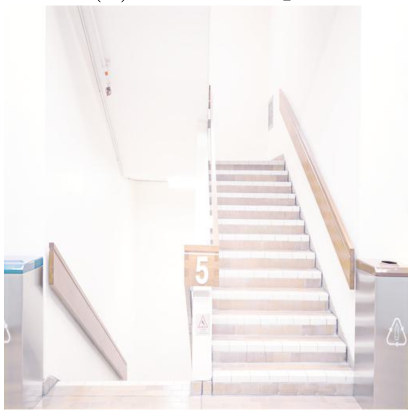
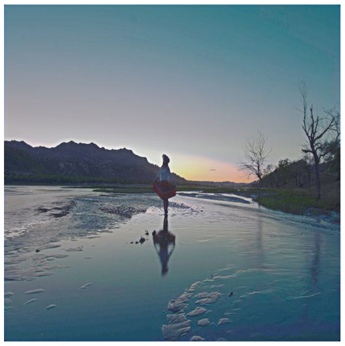
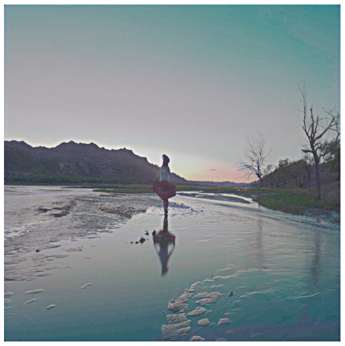

(c) Model output

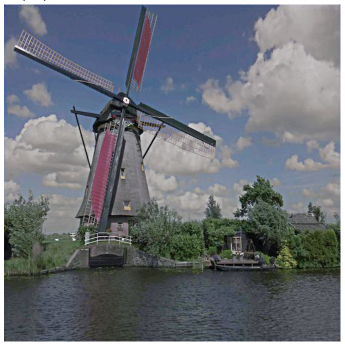
Figure 4: Model outputs for some bad examples

(a) Input


(a) After 3000 iterations


(a) after 9000 iterations

(b) Input


(b) After 3000 iterations

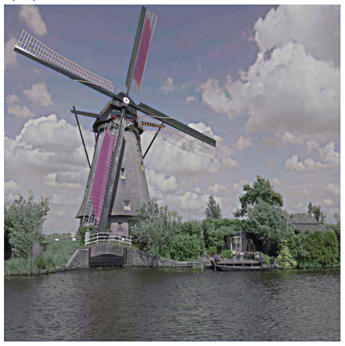

(b) After 9000 iterations

Figure 5: Model outputs at different iterations

Overall the model and this approach does good job in terms of enhancing low light images. However, there are some cases where it outputs deteriorated quality output by adding unnecessary noise (Figure 4). I also found that, low light images in day light (for example, photos clicked facing sun, hence the foreground became dark) seems to be handled well by the model compared to the low light images clicked at night. Probably because the model is able to intuitively leverage the radiance well in the former type than the rather type.
Although the model seemed to be light, since we won't do any spatial reduction through out the network, we can't train the model on the original images but only on the resized images (without running into memory errors). I firmly believe that this constrains the robustness of the model. Even during the inference, enhancing a single original image could be slow or leads to memory error. Thereby making this light weight model not so edge or mobile friendly in practice. I believe this model could act as a good baselines for low light image enhancement methods research, as it is really easy to build and requires no pair wise data.

Reference

1 Guo, Chunle, et al. "Zero-reference deep curve estimation for low-light image enhancement." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

2 Ruixing Wang, Qing Zhang, Chi-Wing Fu, Xiaoyong Shen, Wei-Shi Zheng, and Jiaya Jia. Underexposed photo enhancement using deep illumination estimation. In CVPR, 2019.

3 Chen Wei, Wenjing Wang, Wenhan Yang, and Jiaying Liu. Deep retinex decomposition for low-light enhancement. In BMVC, 2018.

4 Yifan Jiang, Xinyu Gong, Ding Liu, Yu Cheng, Chen Fang, Xiaohui Shen, Jianchao Yang, Pan Zhou, and Zhangyang Wang. EnlightenGAN: Deep light enhancement without paired supervision. In CVPR, 2019.

5 Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycleconsistent adversarial networks. In ICCV, 2017.

6 Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion. In PCCGA, 2007.

7 Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion: A simple and practrical alterrnative to high dynamic range photography. Computer Graphics Forum, 28(1):161– 171, 2009.

8 Gershon Buchsbaum. A spatial processor model for object colour perception. J. Franklin Institute, 310(1):1–26, 1980.

9 Jianrui Cai, Shuhang Gu, and Lei Zhang. Learning a deep single image contrast enhancer from multi-exposure image. IEEE Transactions on Image Processing, 27(4):2049–2026, 2018.

10 https://keras.io/examples/vision/zero_dce/