# PROJECT REPORT
## EXTREME LOW LIGHT IMAGE DENOISING

Submitted By: Abhishek Baghel
DSAI 3rd Year, 22125004

## INTRODUCTION:

The rapid advancement of computer vision technologies and algorithms has significantly expanded the range of applications for image processing tasks such as object classification, object detection, image segmentation, security surveillance, medical imaging and other specialized tasks. For the efficient working of the model of these tasks, clear images are required, but in the real world, there are many factors which leads to some noise in the images obtained to test upon these models. Noise in images refers to unwanted random variations in brightness or colour information which leads to a decrease in the efficiency of the computer vision models unless these images are not correctly pre-processed. Some of the causes of the noise in images include sensor noise (occurring due to the sensor of the camera used to capture the image), high ISO settings (leads to grainier images), low light conditions, environmental factors (due to dust, fog or other conditions), etc. Although, in this project, we mainly discuss about the noise due to the low light conditions, i.e., the low light images. The preprocessing of these types of images increases the efficiency of the computer vision models for their respective tasks. In this project, we try to achieve this preprocessing or denoising of the low light images with the use of the MIRNet model. The aim of the model is to maintain spatially precise high-resolution representations throughout the entire network and to receive contextual information from the low-resolution representations. The model uses the Charbonnier loss for the optimization process.

## ARCHITECTURE:

The core of the architecture is the multi-scale residual block comprising of the following elements: (a) parallel multi-resolution convolution streams for extracting multi-scale features, (b) information exchange across those streams, (c) spatial and channel attention mechanisms for capturing the contextual information, (d) attention based multi-scale feature aggregation.

The different components in the architecture are:

(1) Selective Kernel Feature Fusion: - Receives inputs from three parallel convolution streams carrying different scales of information and combines them, then uses global average pooling to compute channel wise statistics. These statistics are used to generate attention weights (feature descriptors) for each scale, which are applied back to the input features before combining them again.

(2) Spatial Attention Block: - Exploits the inter-spatial dependencies of convolutional features. It applies attention in the spatial domain. It compresses the feature map using channel pooling, applies a convolution to produce a single-channel attention map and then multiplies this attention map with the input tensor to emphasize important spatial regions.

(3) Channel Attention Block: - Exploits the inter-channel relationships of the convolutional features maps. It applies attention in the channel domain. It computes the global average pooling to summarize each channel, passes the result through two convolutional layers to produce attention weights, and then multiplies these weights with the input tensor to emphasize important channels.

(4) Dual Attention Unit: - Share information along both the spatial and channel dimensions. The dual attention unit applies both channel and spatial attention to the input tensor. It first processes the input with convolutions, applies channel and spatial attention blocks, concatenates their outputs, and then reduces the channels back to the original number before adding the result to the input tensor.

(5) Down sampling module: - Reduces the spatial dimensions of the input tensor while increasing the number of channels, by using a combination of convolution and max pooling operations.

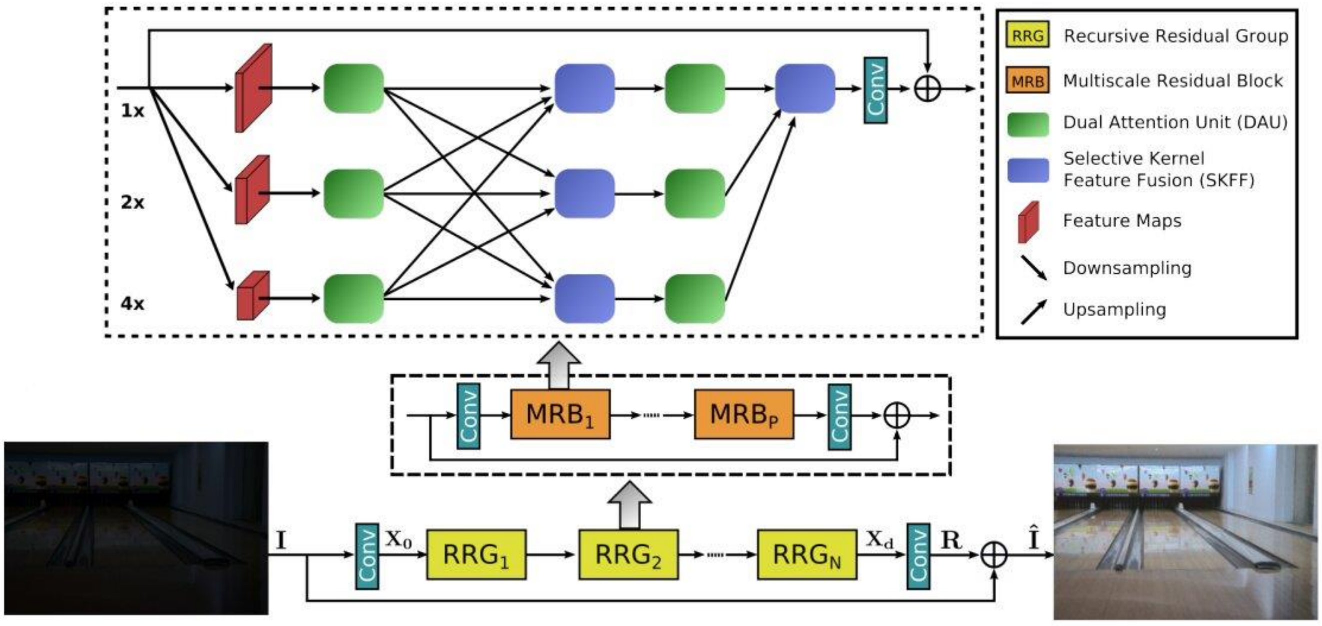
(6) Up sampling module: - Increases the spatial dimensions of the input tensor while reducing the number of channels by using a combination of convolution and up sampling operations.

(7) Multi-Scale Residual Block: - Core unit of the architecture, capable of generating spatially precise output by maintaining high-resolution representations, while receiving rich contextual information from the low-resolutions. The multi scale residual block combines features from three (in this project) scales using down sampling and up sampling modules. Each scale undergoes dual attention processing, and their outputs are fused using selective kernel feature fusion. The resulting features are further processed and added back to the input tensor.

(8) Recursive Residual Group: - Consists of a sequence of multi scale residual blocks. It processes the input tensor through a series of convolutions and finally adds the processed tensor back to the input.

In this project, we have taken the number of recursive residual groups to be 3, while the number of multi scale residual blocks in each recursive residual group is 2 and the number of scales in each multi scale residual block is 3.

The following figure clearly depicts the architecture of the MIRNet model: -



RRG Recursive Residual Group
MRB Multiscale Residual Block
Dual Attention Unit (DAU)
Selective Kernel Feature Fusion (SKFF)
Feature Maps
Downsampling
Upsampling
1x
2x
4x
Conv
MRB₁
MRBₚ
I
X₀
RRG₁
RRG₂
RRG_N
X_d
R
Î

Fig.1 – Architecture of the MIRNet model

## OVERALL PIPELINE:

The image data is taken and processed, and random crops are taken from the images (for both the ground truth and the low light images), this data is then converted into tensors and passed through the network. The network first applies a convolutional layer to extract low-level features, then features maps are passed on to the recursive residual groups yielding deep features. Next, a convolutional layer is applied to these deep features to obtain a residual image **R**. Finally, the denoised image is obtained as **Î = I + R**, where **I** is the original low light image and **Î** is the denoised image. We optimize the training process using the Charbonnier loss:

$$L(\hat{I}, I^*) = \sqrt{\left\|\hat{I} - I^*\right\|^2 + \varepsilon^2},$$

where $\mathbf{I}^*$ denotes the ground-truth image, and **ε** is a constant which we set here to $10^{-3}$.

The metrics used for the evaluation of the model is peak signal-to-noise ratio (PSNR). PSNR measures the ratio between the maximum possible power signal (the peak signal) and the power of corrupting noise that affects the fidelity of its representation (the noise). A higher PSNR value indicates a higher quality reconstruction or enhancement. The formula is as follows:

$$PSNR = 10 * \log_{10} \frac{MAX^2}{MSE}$$

where MAX denotes the maximum possible pixel value (255), and MSE denotes the mean squared error between the denoised image and the ground truth image calculated as the summation of the squared difference between the corresponding pixels of the two images.

## DATASET PREPARATION AND OTHER SPECIFICATIONS:

The dataset on which the training is done is the LoL dataset which contains 485 images for training and 15 for testing. Each image pair in LoL consists of a low-light input image and its corresponding well-exposed reference image. We have split the images available for training into train and validation sets with the train dataset containing 385 images and the validation dataset containing 100 images. The Adam optimizer is used for the optimization of the training process with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and the learning rate being $2 \times 10^{-4}$ . Patches of size (128,128) are extracted from the images and batch size of 4 is used. The number of recursive residual group used in the network architecture is 3 and the number of multi scale residual group in each recursive residual group is 2. The model was trained for 50 epochs.

(We also trained the model with train and validation set of 300 and 185 images respectively and keeping the learning rate as $10^{-4}$ (keeping all other parameters as same), but the results were not so significantly different than the ones obtained in the above configuration.)

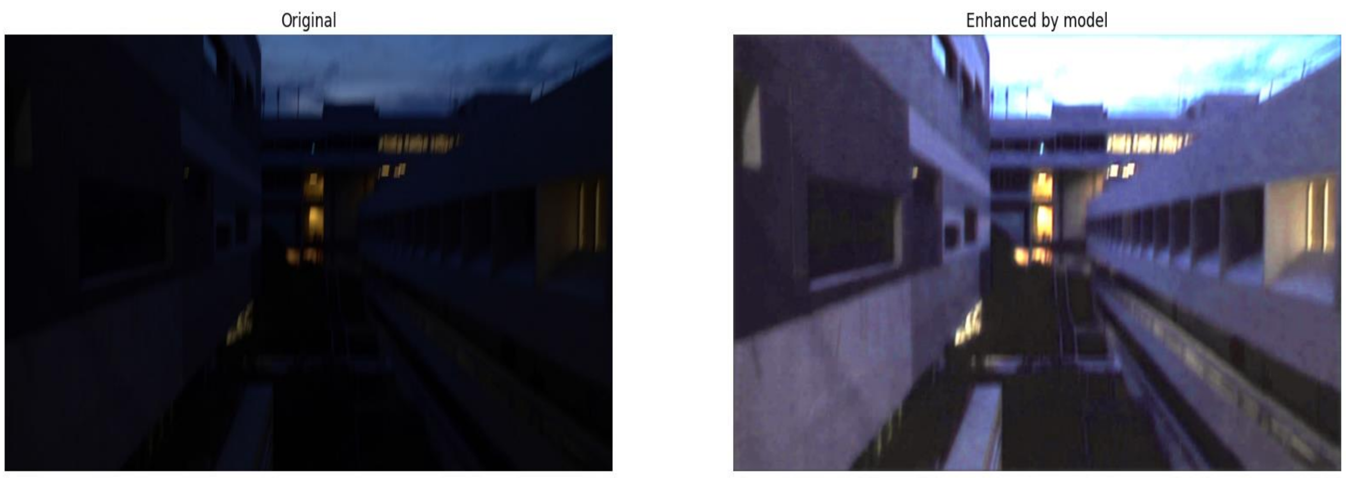
The weights obtained after training the model using the above configuration can be downloaded using this link.

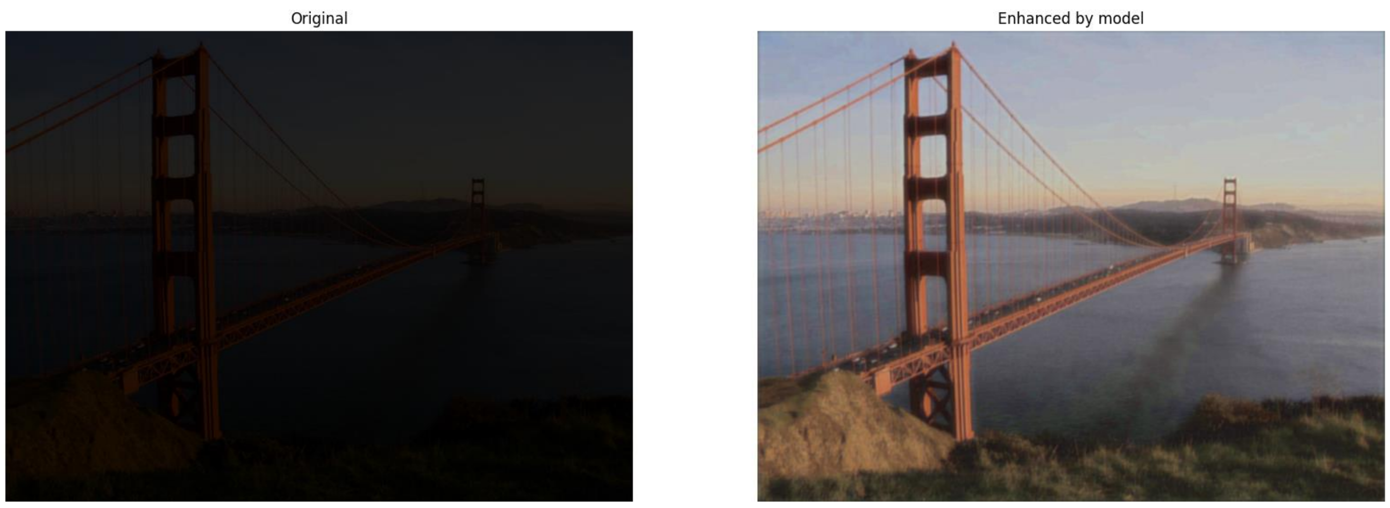
## RESULTS:

The model is evaluated mainly on the metrics PSNR (peak signal-to-noise ratio) and we have also calculated the MAE (mean absolute error) and MSE (mean squared error).

| *Metrics* | *Train Dataset (LoL)* | *Valid Dataset (LoL)* | *Test Dataset (LoL)* |
|---|---|---|---|
| *MAE* | 143.30 | 102.54 | 134.57 |
| *MSE* | 98.10 | 100.60 | 98.75 |
| *PSNR* | 28.30 | 28.15 | 28.25 |

Some of the examples of the images enhanced by the model:





## PROJECT IMPLEMENTATION:

The implementation of the project and the code files can be found in the following GitHub repository: Image-enhancement-using-MIRNet

The GitHub files contain the following files: -

- **MIRNET_model.py**: - This file contains the architecture and the full code for the MIRNet model
- **Utils.py**: - This file contains the additional utility functions required by the other code files
- **Train.py**: - This file helps to run the training script and trains the model
- **Test.py**: - This file is used to calculate the psnr for a given set of low light images and their corresponding ground truth images
- **Main.py**: - This file is used to run the inference on a set of images and returns the denoised images
- **Requirements.txt**: - This file contains the libraries that need to be installed for the proper working of the model
- **Image_enhancement_tutorial.ipynb**: - This file helps you to learn about the usage of the other files and also tells how to download the LoL dataset as well as the pre-trained weights.

Additional details on the working of each of the file can be found in the readme section of the GitHub repository of this project.

## WORKING OF THE FILES:

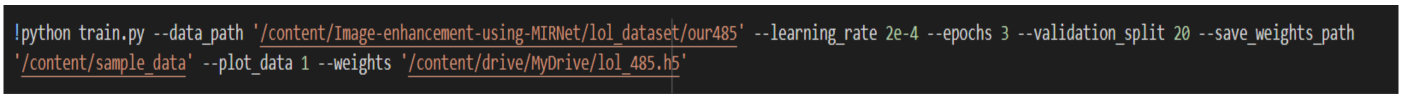
### *train.py*

You must use the train.py file to train the model, the various parameters that you need to pass while running the file are: -

- data_path: - Path to the training data (must include folders named *low* and *high* which contain the low light images and the ground truth (the well-lit) images respectively). It is a required parameter.
- learning_rate: - Set the learning rate (default = 2e-4)

- weights: - Path to the pretrained weights, if no path is given, the model starts to train from scratch
- epochs: - Set the number of epochs (default = 1)
- validation_split: - Percentage of data to be divided into validation set. Value should be set between 0-100 and if no value is provided, then no validation set is made
- validation_data: - Path to the validation set if you don't want to split the training dataset (it will override the validation_split if used)
- save_weights_path: - Path to the folder where the weights and the plots are to be saved after training the model (If no path provided, will save the weights in the running directory)
- plot_data: - Whether to save the plots or not. Accepts the value 0 (false) and 1 (true). By default, the value is 0
- batch_size: - Set the batch size (default = 4)

Example: -



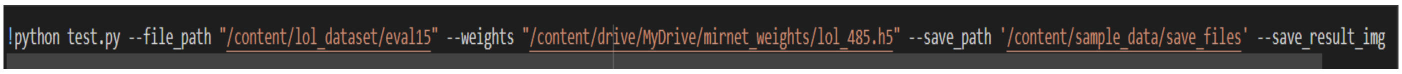
```
!python train.py --data_path '/content/Image-enhancement-using-MIRNet/lol_dataset/our485' --learning_rate 2e-4 --epochs 3 --validation_split 20 --save_weights_path
'/content/sample_data' --plot_data 1 --weights '/content/drive/MyDrive/lol_485.h5'
```

## *test.py*

You must use the test.py file in order to evaluate the model on a given set of images and returns the psnr for the set of images, the various parameters that you need to pass while running the file are: -

- file_path: - Path to the testing data (must include folders named *low* and *high* which contain the low light images and the ground truth (the well-lit) images respectively). It is a required parameter.
- weights: - Path to the pretrained weights, if no path is given, the model performs poor inference (since it acts as a model which has not been trained)
- save_result_img: - Whether to save the images or not. Accepts the value 0 (false) and 1 (true). By default, the value is 0 and it will not save the images even if the save_path is provided
- save_path: - Path where the resulting denoised images would be stored if the save_result_img value is set to 1. Becomes a required parameter if the value of save_result_img is set to 1

Example: -



```
!python test.py --file_path "/content/lol_dataset/eval15" --weights "/content/drive/MyDrive/mirnet_weights/lol_485.h5" --save_path '/content/sample_data/save_files' --save_result_img
```

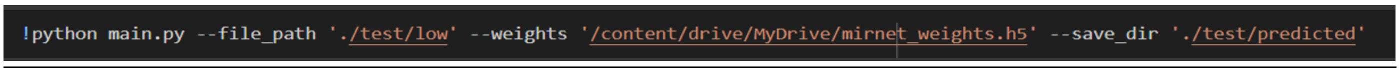
## *main.py*

The main.py accepts parameter values through argparse and the following set of parameters can be provided to it: -

- file_path: - this is a required parameter, and you need to pass the path to the test images in this parameter
- weights: - this is a not a required parameter, has a default value of None and you need to pass the path of the pretrained weights in this parameter

- save_dir: - this is a required parameter, and you need to pass the path where you want to save the denoised images in this parameter

Example: -



```
!python main.py --file_path './test/low' --weights '/content/drive/MyDrive/mirnet_weights.h5' --save_dir './test/predicted'
```

## CONCLUSION:

Visual inspection of the enhanced images obtained by the model shows great improvements in the brightness and visibility compared to the low light images. The MIRNet enhanced images can maintain their high resolution compared to the other baseline models. The examples of the result images demonstrate the ability of the MIRNet model to recover details in extremely low-light images, that were not that distinguishable previously.

Limitations that are observed in the model are the occasional over smoothing of the texture of the input image and hence some modifications are required to produce high resolution sharp images. Further modifications that can be done in the model are that it does not over brighten the input images, this might be achieved by doing some alterations in the images of the training set or by adding images of variable lightning conditions to make the model more robust. Another improvement that can be done is the optimization of the model architecture or techniques to improve the inference time to meet the prospects of the real time applications. Some techniques that can be used to achieve lower inference time are model pruning, teacher-student training, etc.

## RESOURCES:

- MIRNet paper: - https://arxiv.org/pdf/2003.06792v2
- https://github.com/swz30/MIRNet
- https://towardsdatascience.com/enhancing-low-light-images-using-mirnet-a149d07528a0