Computer Vision Project

Low Light Imagery

Team - Wanda Vision

TA Mentor – Pulkit Gera

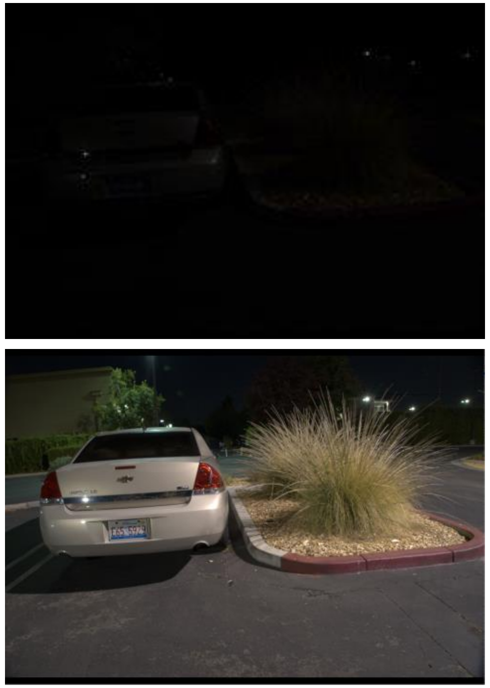
Team Members:
1. Dipanwita Guhathakurta - 2018112004
2. Manav Bhatia - 2018102009
3. Sanskar Tibrewal - 2018111034
4. Vedant Mundheda - 2018112006

Inspiration

Low Light Imaging

We derive inspiration from the CVPR 2018 paper "**Learning to See in the Dark**" which proposes a pipeline for processing low-light images, based on end-to-end training of a fully-convolutional network. The pipeline proposed can process raw sensor data as opposed to physical changes such as extending exposure time and using flash which can introduce blur and requires expensive hardware.

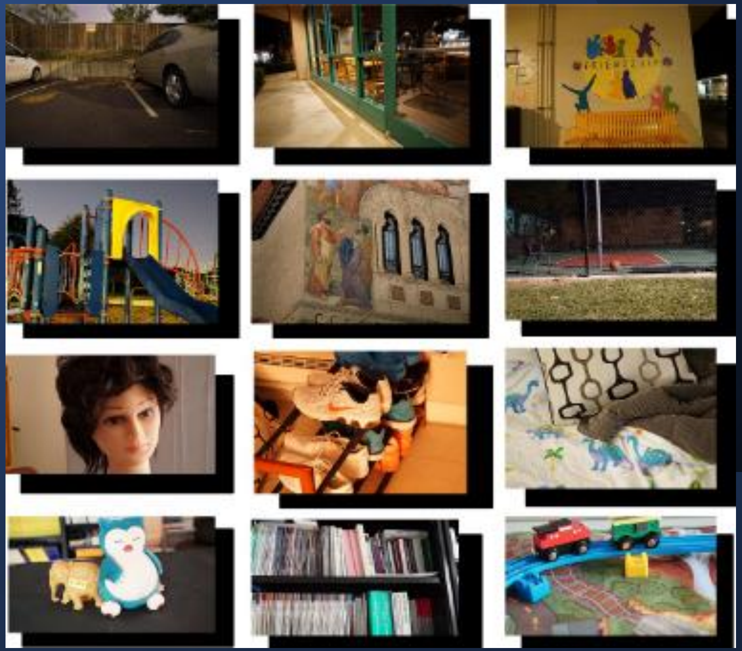
A) Raw sensor data (ISO 8000)
B) Image produced by camera using greater exposure (ISO 409,600)
C) Image produced from the network proposed (example from paper)



Top: Image from the Sony α7S II sensor with short exposure
Bottom: Output from the model as proposed in the paper

Implementation

DATASET: SID SONY



See-in-the-Dark dataset (SID)

Camera: Sony α7S II

Dataset Size: 5094 pairs of short-exposure and long-exposure images

Image shape: 4240×2832

Exposure for Input Images: 0.03 to 0.1 sec

Exposure for Ground Truth Images: 10 sec

Ground Truth (long exposure)	Input Image (short exposure)
	

Dataset link:
https://drive.google.com/file/d/1G6VruemZtpOyHjOC5N8Ww3ftVXOydSXx/view

INSTRUCTIONS TO RUN

The Instructions to Run this repository locally have been written in the README file, a screenshot of which can be seen here.

Run locally:

- Clone the repository:

  `git clone https://github.com/Computer-Vision-IIITH-2021/project-wandavision.git`

- Navigate to the code folder:

  `cd see_in_dark`

- Download and extract the dataset folders `/long` and `/short` in the `Sony_test/` folder.

- Create two new folders: `mkdir saved_model/`

  `mkdir test_result_new/`

Training from scratch:

- Change the directory locations on the files as per your directory structure.
- Train the model from scratch: `python train_Sony.py`
- Test the model: `python test_Sony.py`

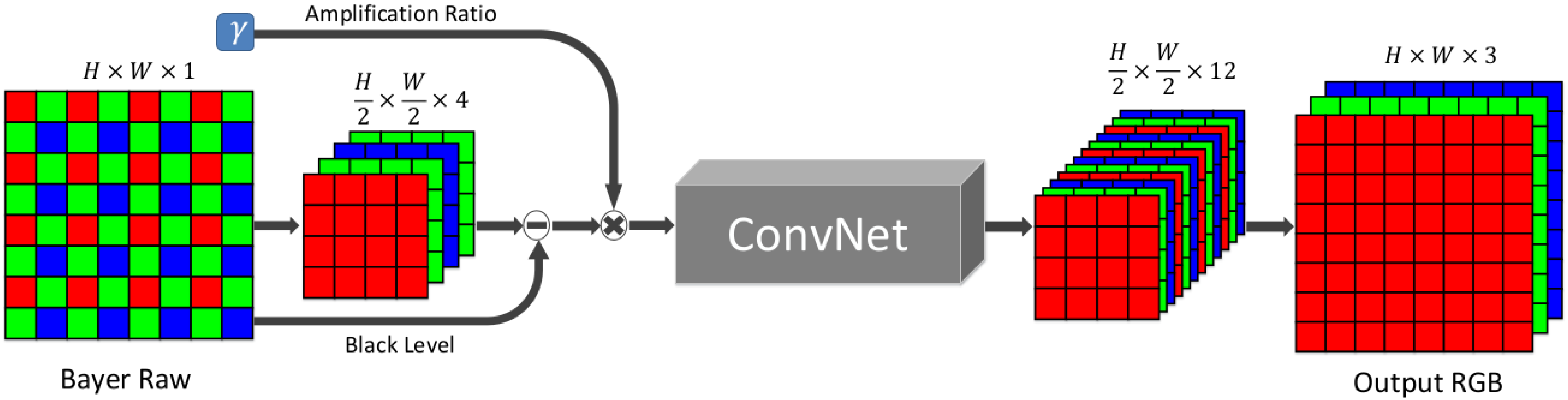
Using the pre-trained model:

- Pretrained model is saved as `checkpoint_sony_e4000.pth` under `Sony_test/saved_model`
- To test the pretrained model: `python test_Sony.py`
- Test images are stored in `test_results_Sony/` directory.

Get quantitative results:

- Run all cells of the notebook `Quantitative_results.ipynb` changing the test directory path to where the test results are stored.
- The notebook generates PSNR and SSIM values between pairs of Ground Truth and output images.

IMAGE PROCESSING PIPELINE



CHOICE OF ARCHITECTURE & LOSS FUNCTION

Why FCN?

End-to-end image processing pipeline that runs fast even on large raw images, does not require a fixed size input output like Fully Connected Layers

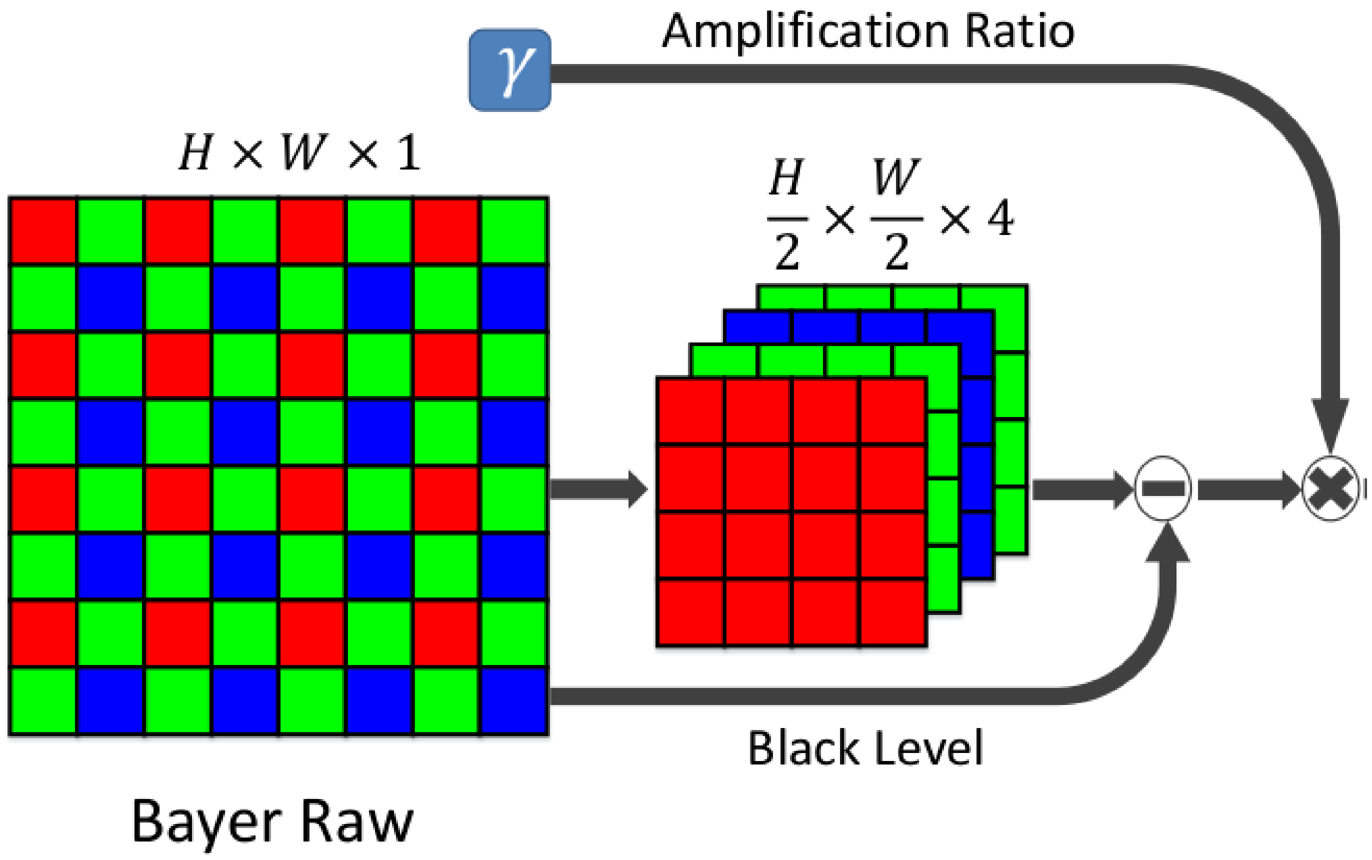
Why UNet?

- less memory consumption
- doesn't use fully connected layers to work on small image patches

For other model and training parameters, we performed quantitative experiments and decided on the best choice of parameters, results for which will be shown in subsequent slides.

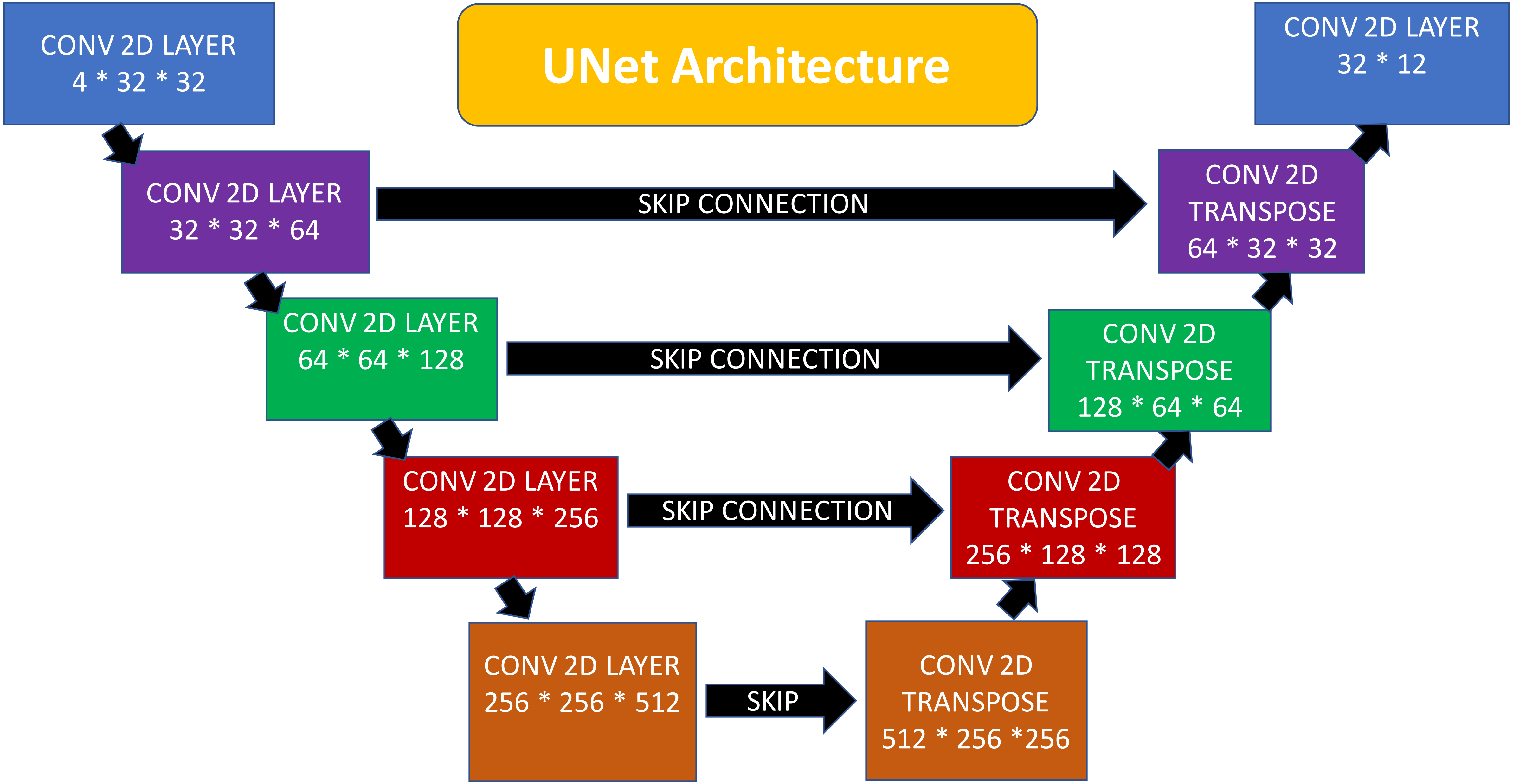
PRE-PROCESSING

Libraries Used
RawPy
NumPy



Halve dimensions → Pack Bayer Raw data into 4 channels → Subtract black levels → Amplify by amplification ratio

Amplification Ratio
= Ratio of exposure times between input and reference images


UNet Architecture
CONV 2D LAYER
4 * 32 * 32
CONV 2D LAYER
32 * 32 * 64
SKIP CONNECTION
CONV 2D
TRANSPOSE
64 * 32 * 32
CONV 2D LAYER
32 * 12
CONV 2D LAYER
64 * 64 * 128
SKIP CONNECTION
CONV 2D
TRANSPOSE
128 * 64 * 64
CONV 2D LAYER
128 * 128 * 256
SKIP CONNECTION
CONV 2D
TRANSPOSE
256 * 128 * 128
CONV 2D LAYER
256 * 256 * 512
SKIP
CONV 2D
TRANSPOSE
512 * 256 *256

MODEL CODE

U-Net architecture:

The first five convolution nets define the encoder architecture of the U-Net, such that we move from (4*32) to (512*512) 2D Convolutional Kernel.

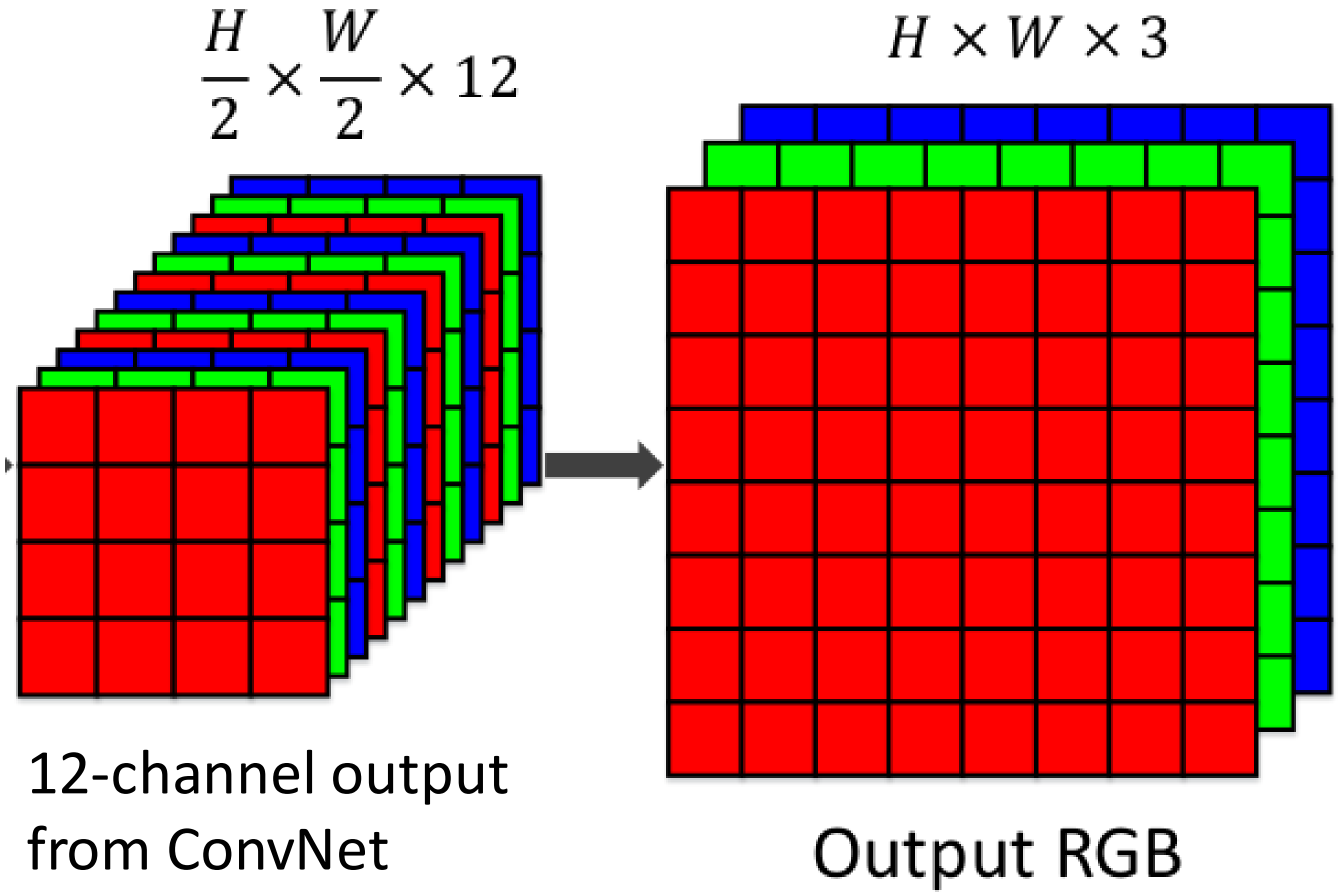
The next 5 layers define the decoder architecture, along with the conserved skip connections

The size of convolution kernel is reduced from (512*512) to (32*12) using Transpose Convolutions(which halves the number of layers in the next stage), which finally gives us a 12-Layered Output.

```
def __init__(self, num_classes = 10):
    super(SeeInDark, self).__init__()

    self.conv1_1 = nn.Conv2d(4, 32, kernel_size=3, stride=1, padding=1)
    self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
    self.pool1 = nn.MaxPool2d(kernel_size=2)

    self.conv2_1 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
    self.conv2_2 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
    self.pool2 = nn.MaxPool2d(kernel_size=2)

    self.conv3_1 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
    self.conv3_2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
    self.pool3 = nn.MaxPool2d(kernel_size=2)

    self.conv4_1 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
    self.conv4_2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
    self.pool4 = nn.MaxPool2d(kernel_size=2)

    self.conv5_1 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
    self.conv5_2 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)

    self.upv6 = nn.ConvTranspose2d(512, 256, 2, stride=2)
    self.conv6_1 = nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1)
    self.conv6_2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)

    self.upv7 = nn.ConvTranspose2d(256, 128, 2, stride=2)
    self.conv7_1 = nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1)
    self.conv7_2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)

    self.upv8 = nn.ConvTranspose2d(128, 64, 2, stride=2)
    self.conv8_1 = nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1)
    self.conv8_2 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)

    self.upv9 = nn.ConvTranspose2d(64, 32, 2, stride=2)
    self.conv9_1 = nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1)
    self.conv9_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)

    self.conv10_1 = nn.Conv2d(32, 12, kernel_size=1, stride=1)
```

POST-PROCESSING

Libraries Used
PyTorch
NumPy



Pass **(C x r2)*(H/2)*(W/2)** output through sub-Pixel layer

Obtain **C*((H/2)*r)*((W/2)*r) image**

TRAINING THE NETWORK

Input: short exposure images

Loss Function Used: L1 loss

Optimizer: ADAM

Batch: 512*512 window from original image

Data Augmentation: Random flipping and rotation

Learning Rate : 10^{-4} after 2000 epochs, 10^{-5}

Iterations: 4000 epochs

Ground truth: long-exposure images

RESULTS OBTAINED

TESTING AND EXPERIMENTS

Train & Validation

- **We trained the model on 60 distinct image pairs of Ground Truth exposure 10s and input exposures of 0.1s, 0.04s and 0.033s.**
- **We validated our results on 10 more image pairs from the training set**

Testing

- **We tested our model on 51 distinct unseen test image pairs.**

Experiments

- **Change ADAM optimizer to Adagrad**
- **Change L1 loss to L2 loss**
- **Change Leaky ReLU activation to ReLU activation**

Relevant Links

Dataset Link:

- https://drive.google.com/file/d/1G6VruemZtpOyHjOC5N8Ww3ftVXOydSXx/view

Code Link:

- https://github.com/Computer-Vision-IIITH-2021/project-wandavision

Results Link:

- Test images
- Note: there are images named *_100, *_250 and *_300 denoting the amplification ratio used.

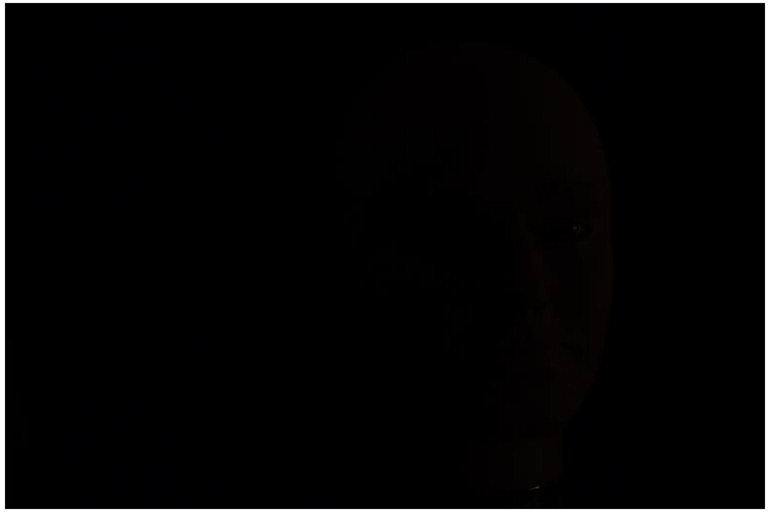
Ground Truth	Input Image	Output Image

Test Image 1: Ground truth Exposure = 10s,Input Exposure = 0.1s

PSNR = 31.02209

Test Image 1: Ground truth Exposure = 10s,Input Exposure = 0.04s

PSNR = 29.6558



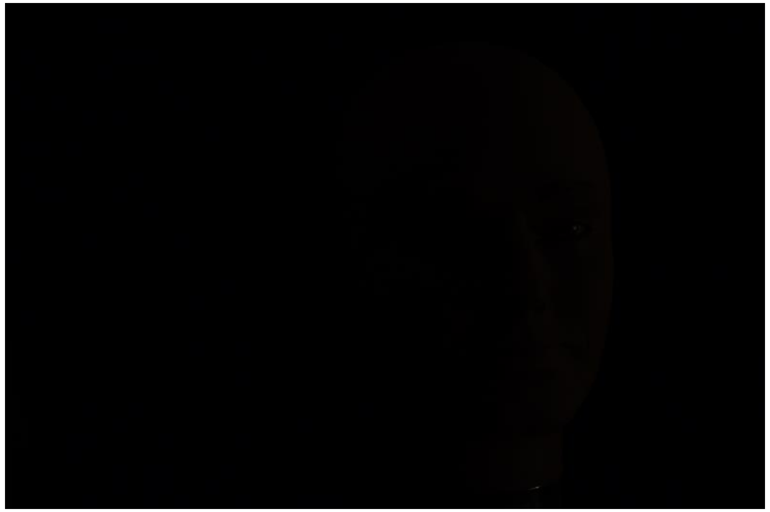
Ground Truth | Input Image | Output Image

Test Image 2: Ground truth Exposure = 10s,Input Exposure = 0.1s

PSNR = 33.34167

Test Image 2: Ground truth Exposure = 10s,Input Exposure = 0.04s

PSNR = 31.39138



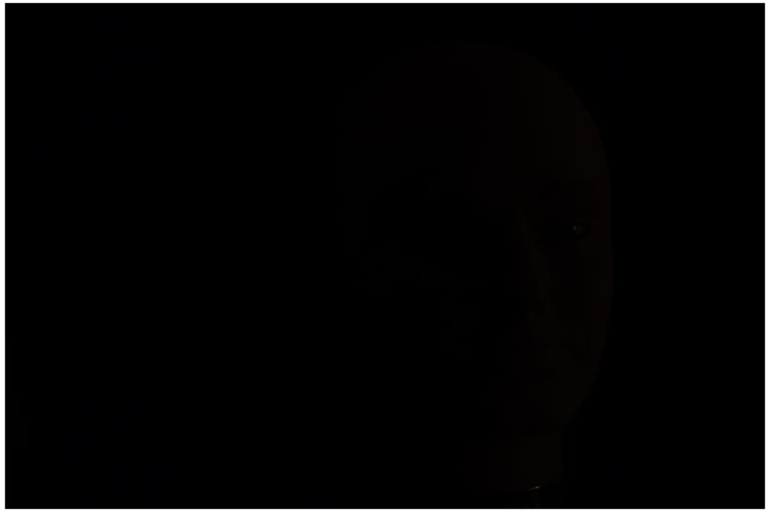
Ground Truth	Input Image	Output Image

Test Image 3: Ground truth Exposure = 10s,Input Exposure = 0.1s

PSNR = 21.84601

Test Image 3: Ground truth Exposure = 10s,Input Exposure = 0.04s

PSNR = 20.68907



Ground Truth	Input Image	Output Image

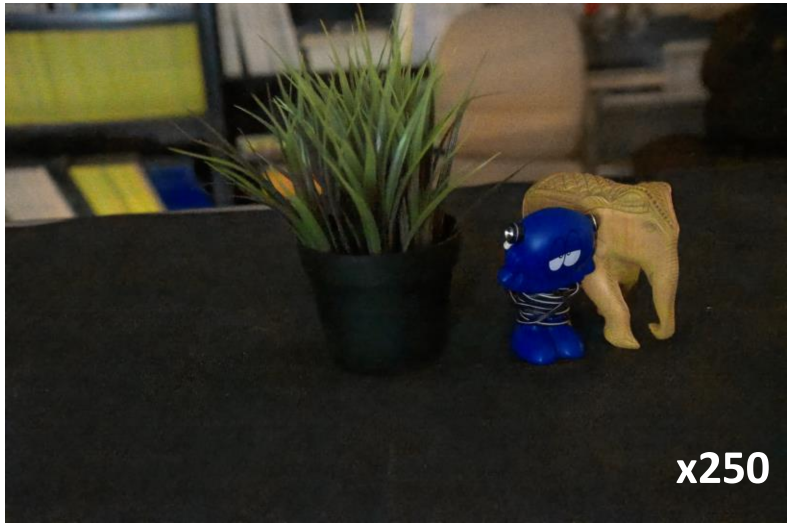
Test Image 4: Ground truth Exposure = 10s,Input Exposure = 0.1s

PSNR = 32.50815

Test Image 4: Ground truth Exposure = 10s,Input Exposure = 0.04s

PSNR = 30.23718

Ground Truth exposure = 10s



With input exposure = 0.1s	With input exposure = 0.04s	With input exposure = 0.033s
PSNR = 30.882503	PSNR = 24.537966	PSNR = 23.445992

QUANTIFYING RESULTS

Peak Signal-to-Noise Ratio(PSNR)

Peak Signal to Noise Ratio is a ratio between the maximum signal power and the power of corrupting noise that affects the fidelity of the representation.

Since the signals have a very dynamic and wide range, PSNR is usually expressed as a logarithmic quality using the decibel scale.

If we define two images as I and K, then MSE between two images is defined as-

$$MSE = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

MAXI represents the maximum pixel value in the image. Then finally, the PSNR can be defined as-

$$PSNR = 10 \times \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

Structural Similarity Index Measure (SSIM)

What is SSIM?

SSIM is a method for predicting the perceived quality of images and considers the structural similarity of two images, as opposed to PSNR, which is an absolute metric.

A Higher SSIM value indicates better structural similarity between two images.

SSIM formula for images x (Img1) and y (Img2)

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Notation used above:

μ_x - The mean of pixels of image 1
μ_y - The mean of pixels of image 2
σx^2 - The variance of pixels of image 1
σy^2 - The variance of pixels of image 2
σxy - The co-variance between pixels of image 1 and image 2
$c_1 = (k_1 L)^2 and c_2 = (k_2 L)^2$ - The constants to stabilise a weak denominator
L - The entire range of values of pixels, [0, 255]
$k_1 = 0.01$ and $k_2 = 0.03$

Code snippets for PSNR & SSIM

SSIM

```python
def ssim(img1, img2):
    C1 = (0.01 * 255)**2
    C2 = (0.03 * 255)**2

    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    kernel = cv2.getGaussianKernel(11, 1.5)
    window = np.outer(kernel, kernel.transpose())

    mu1 = cv2.filter2D(img1, -1, window)[5:-5, 5:-5]
    mu2 = cv2.filter2D(img2, -1, window)[5:-5, 5:-5]
    mu1_sq = mu1**2
    mu2_sq = mu2**2
    mu1_mu2 = mu1 * mu2
    sigma1_sq = cv2.filter2D(img1**2, -1, window)[5:-5, 5:-5] - mu1_sq
    sigma2_sq = cv2.filter2D(img2**2, -1, window)[5:-5, 5:-5] - mu2_sq
    sigma12 = cv2.filter2D(img1 * img2, -1, window)[5:-5, 5:-5] - mu1_mu2

    ssim_map = ((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq + C1) *
                                                            (sigma1_sq + sigma2_sq + C2))
    return ssim_map.mean()
```

```python
def calc_ssim(img1, img2):
    if not img1.shape == img2.shape:
        raise ValueError('Input images must have the same dimensions.')
    if img1.ndim == 2:
        return ssim(img1, img2)
    elif img1.ndim == 3:
        if img1.shape[2] == 3:
            ssims = []
            for i in range(3):
                ssims.append(ssim(img1, img2))
            return np.array(ssims).mean()
        elif img1.shape[2] == 1:
            return ssim(np.squeeze(img1), np.squeeze(img2))
    else:
        raise ValueError('Wrong input image dimensions.')
```

PSNR

```python
def calc_psnr(img1, img2):
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    if img1.ndim == 2:
        mse = np.mean((img1 - img2)**2)
    elif img1.ndim == 3:
        mse1 = np.mean((img1[:,:,0] - img2[:,:,0])**2)
        mse2 = np.mean((img1[:,:,1] - img2[:,:,1])**2)
        mse3 = np.mean((img1[:,:,2] - img2[:,:,2])**2)
        mse = (mse1 + mse2 + mse3)/3
    if mse == 0:
        return float('inf')
    psnr = 20 * math.log10(255.0 / math.sqrt(mse))
    return psnr
```

QUANTITATIVE RESULTS OF EXPERIMENTS

EXP. NO.	OPTIMIZER	LOSS FUNCTION	ACTIVATION FUNCTION	PSNR	SSIM
1	**ADAM**	**L1 LOSS**	**LEAKY RELU**	**24.481**	**0.6489**
2	ADAGRAD	L1 LOSS	LEAKY RELU	21.587	0.6251
3	ADAM	L2 LOSS	LEAKY RELU	23.698	0.6045
4	ADAM	L1 LOSS	RELU	22.668	0.6102

Pretrained best model: GitHub model

LIMITATIONS

Loss of some hues in the output images with very low exposure

Amplification ratio is chosen externally based on ratio of exposure times.

Currently the model is trained on only images from the Sony α7S II camera

Not suited for real-time applications at full resolutions

The SID dataset is limited in that it does not contain humans and dynamic objects.

REFERENCES

1. C. Chen, Q. Chen, J. Xu and V. Koltun, "Learning to See in the Dark," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 3291-3300, doi: 10.1109/CVPR.2018.00347.

2**. Original Project Website-**
http://cchen156.github.io/SID.html

3. Demo YouTube Video for the original Project-
https://youtu.be/qWKUFK7MWvg

4. PSNR :
https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

5. SSIM:
https://en.wikipedia.org/wiki/Structural_similarity

THANK YOU