

VLG SUMMER 2024 PROJECT

Extreme Low Light Image Denoising

#Reference: NTIRE Challenge Models 2024

-INTRODUCTION

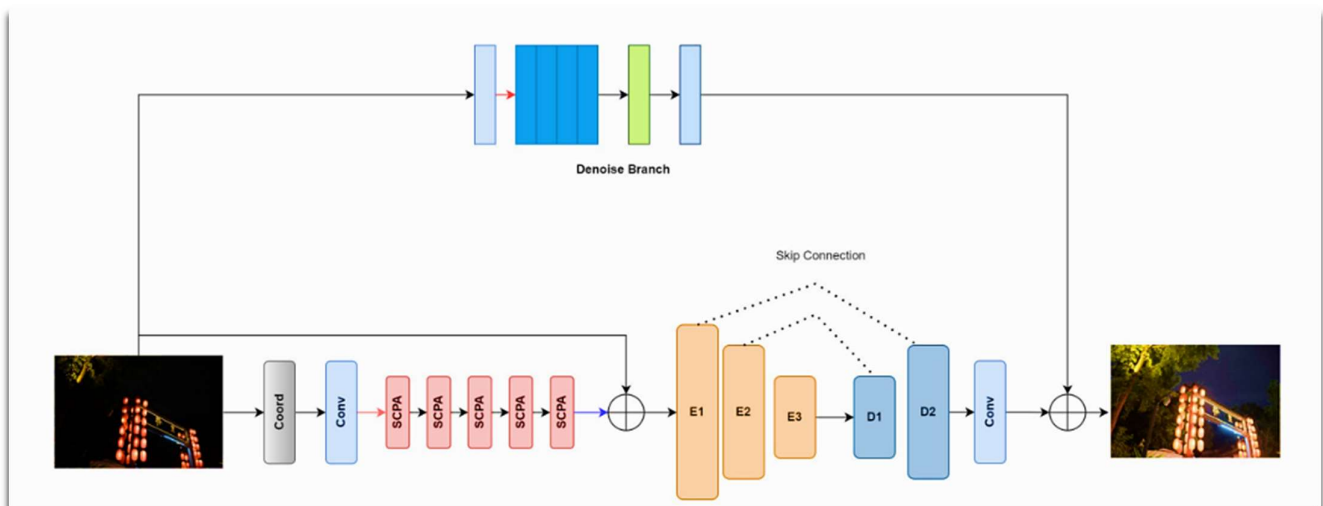
As the name suggests, this project is an important Computer Vision task which aims to eliminate the noise factor from the low light (night light) images, which will make them clearly visible with a good PSNR, helping further for various aspects (healthcare, diagnostics).

There are many models proposed and available out there, with different complex architectures in play; each one having their own unique set of blocks invoked to improve this task's performance.

-MODEL ARCHITECTURE

For this project, I tried to implement model named "Image Lab" as proposed in the NTIRE Challenge 2024 Paper (Ref: https://drive.google.com/file/d/1zRtOgDD0JeGt_5Gj3TpdXjdf-BLkRAYC/view?usp=sharing). This model proposes a unique idea of Pixel Attention Mechanism, implemented with the help of **Self Calibrated Convolutions with Pixel Attention (SCPA)** and **Coordinate Convolution (CoordConv)**.

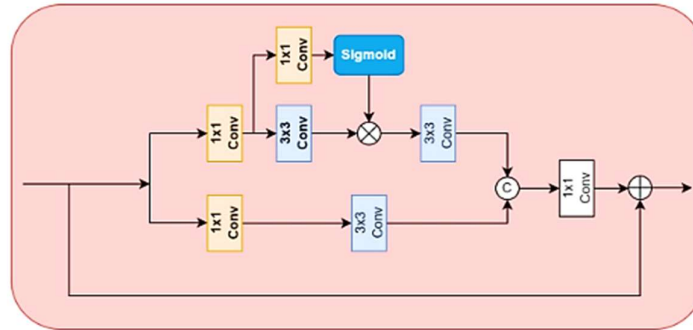
The input image is first passed through a CoordConv layer to add channels containing hard-coded coordinates, enriching the representation with spatial information. This augmented representation is then downsampled and fed into five consecutive SCPA layers with a Pixel Attention Block. SCPA layers enhance the model's capture of intricate spatial patterns and features. The output from the SCPA layers is upsampled and added back to the input image. This combined representation is passed through a **Modified U-Net architecture with Residual Dense Channel Attention (RDCA) blocks**. The U-Net consists of three encoder blocks and two decoder blocks. Each encoder block contains two RDCA blocks followed by **downsampling**, while each decoder block contains two RDCA blocks followed by **upsampling**. This design facilitates the extraction and refinement of features at multiple scales. Simultaneously, the input image is fed into a sophisticated denoising block to produce a three-channel denoised image. The denoising block comprises four inverse convolutional layers followed by an attention mechanism. This mechanism effectively suppresses noise while preserving important image details, enhancing the overall quality of the denoised image. The outputs of the **RDCA-UNet** and the denoising block are added to the input image, resulting in a final enhanced image combining the denoising block's denoised features with the refined features from the RDCA-UNet.



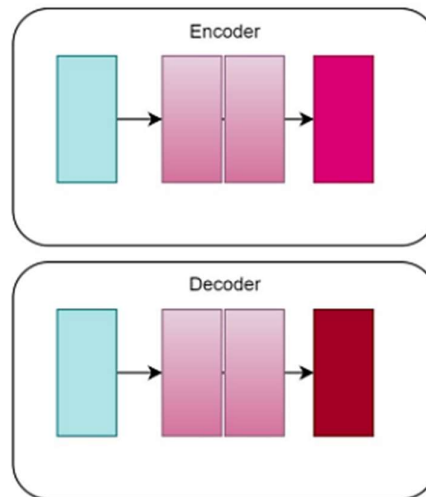
VLG SUMMER 2024 PROJECT

Different Blocks used in the above architecture:

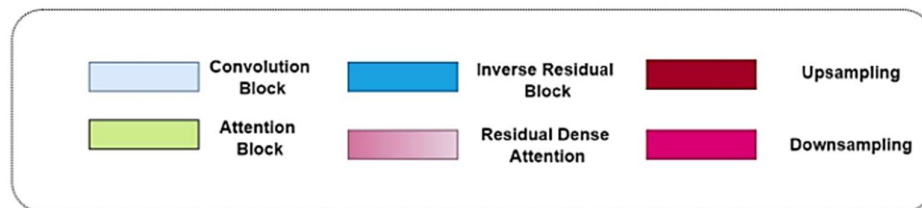
I. The SCPA (Self Calibrated Convolution with Pixel Attention) Block



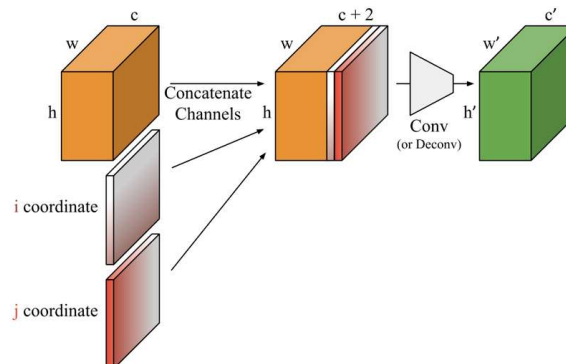
II. The Encoder-Decoder Structure



III. Miscellaneous (all blocks defined)



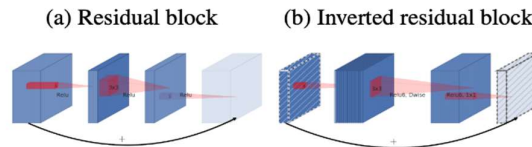
IV. The CoordConv Block



VLG SUMMER 2024 PROJECT

-Inverted Residual Block (IRB):

It is a type of Residual Block with an inverted structure, for efficiency reasons. The architecture is very similar to the original Residual Block but got a change in the input strategy. A traditional Residual Block has wide>narrow>wide type structure. The input has a high number of channels, which are compressed with a 1x1 convolution. The number of channels is then increased again with a 1x1 convolution so input and output can be added. On the other hand, Inverted Residual Block has a narrow>wide>narrow type structure. We first widen the 1x1 convolution, then use a 3x3 convolution, and then again narrowing to 1x1 so that input and output can be added.



```
class inverted_residual_block(nn.Module):
    super(inverted_residual_block, self).__init__()
    self.stride=stride
    hidden_dmi=in_channels*expansion_factor
    self.use_residual=self.stride==1 and in_channels==out_channels

    layers=[]
    if expansion_factor!= 1:
        layers.append(nn.Conv2d(in_channels,hidden_dmi,kernel_size=1,bias=False))
        layers.append(nn.BatchNorm2d(hidden_dmi))
        layers.append(nn.ReLU(inplace=True))

    layers.append(nn.Conv2d(hidden_dmi,hidden_dmi,kernel_size=3,stride=1,padding=1,groups=hidden_dmi,bias=False))
    layers.append(nn.BatchNorm2d(hidden_dmi))
    layers.append(nn.ReLU(inplace=True))

    layers.append(nn.Conv2d(hidden_dmi, out_channels, kernel_size=1, bias=False))
    layers.append(nn.BatchNorm2d(out_channels))

    self.conv=nn.Sequential(*layers)

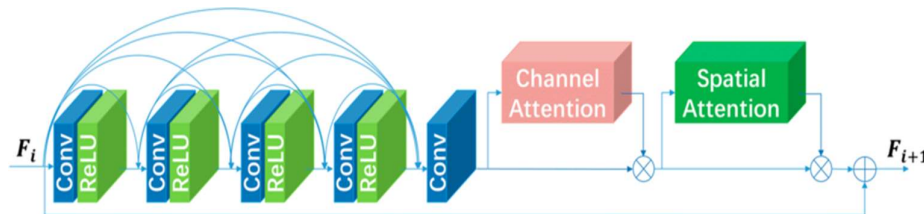
    def forward(self,x):
        if self.use_residual:
            return x+self.conv(x)
        else:
            return self.conv(x)
```

Feature: In the above snippet of the IRB, the hidden dimension refers to the expanded dimension of the feature maps in the middle layers of the architecture using a 1x1 convolution.

-Coordination Convolution (CoordConv) [FIG. IV]:

CoordConv is a specific type of Convolution layer which basically is an extension to the feature map of the image, concatenating two more channels, i.e., the x & y coordinates of the important features (the hard-coded coordinates). The CoordConv layer keeps the properties of few parameters and efficient computation from convolutions, but allows the network to learn to keep or to discard translation invariance as is needed for the task being learned.

-Residual Dense Attention (RDA):



This block is basically an interplay of three different architectures: the Residual Block, the Dense Network, & the Spatial Attention Block. The first RCAB include projection unit (RCABP) to reduce the number of channels in the residual group. For example, let k be the number of channels of the initial feature f_1 and also the growth rate of dense connection between residual groups. The number of input

VLG SUMMER 2024 PROJECT

channels of the i -th residual group is $i \times k$ and each residual group generates k channel features. To the element-wise sum with different number of channels between input and output, RCABP reduces the number of channels from $i \times k$ to k by using 1×1 convolution layer at the residual connection and at the first convolution layer in the residual block.

RCAB: Residual Channel Attention Block

-DATASET

Link to the training dataset:

<https://vlgopenspace.slack.com/files/U068N7XK803/F0749TG0ZSS/train.zip>

I used Data Augmentation techniques for the training purpose, which helped me enhance the size of dataset (almost tripled). Dataset includes high-resolution low light and high light images. Each image has got 3 channels (RGB), with a size 400×600 pixels.

```
def augment_and_save_image(input_folder, output_folder, num_augmentations=3):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    data_transforms = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(brightness=0.1, contrast=0.2, saturation=0.2),
        transforms.RandomRotation(5),
        transforms.ToTensor(),
        transforms.Normalize()
    ])

    for img_name in os.listdir(input_folder):
        img_path = os.path.join(input_folder, img_name)
        image = Image.open(img_path).convert("RGB")

        for i in range(num_augmentations):
            augmented_image = data_transforms(image)
            augmented_img_name = f"{os.path.splitext(img_name)[0]}_aug_{i+1}{os.path.splitext(img_name)[1]}"
            augmented_img_path = os.path.join(output_folder, augmented_img_name)
            augmented_image.save(augmented_img_path)
```

-TRAINING

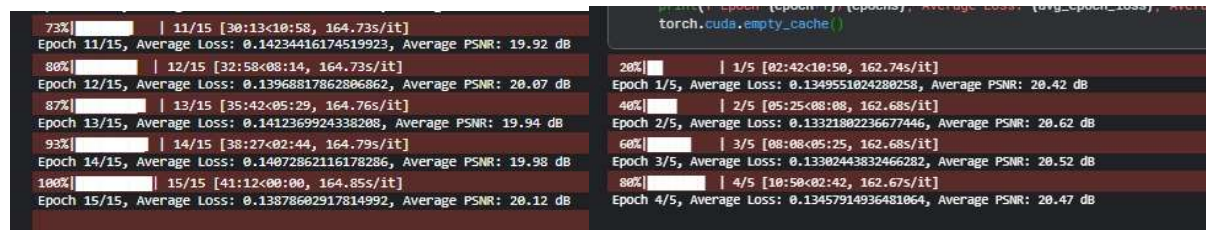
The training phase involved preparing the dataset for the input, transforming them to the tensor format and normalizing the pixel values. The model was trained using Kaggle's GPU T4's environment with over 50 epochs, having a loss function L defined by:

$$L = 0.1 * L_{ssim} + L_1 + L_{grad}$$

Optimized by Adam optimizer, with a learning rate of $1e-3$.

Ref: SSIM (Structural Similarity Index Measure); Grad (Gradient Loss)

Higher the SSIM, higher the similarity between pictures (the Ground truth and the Enhanced Image).











PROBLEMS FACED: The major problem faced during the training was overfitting of the model; the tradeoff between the batch size and learning rate. Constantly tried to figure out the problem behind the model's architecture, but it was prepared accordingly; when the batch was lower (i.e. 1,2), the model worked perfectly fine with a training psnr of 23+, giving 21+ values on validation set, but as the batch size was increasing, (4,8,10), the model performed poorly, giving a training psnr between 18-19 and values of 16-17 on validation set.

VLG SUMMER 2024 PROJECT

-RESULTS

After testing on sample images, the results obtained, and their metrics like PSNR, MAE & MSE are listed below:

		<i>PSNR (db.)</i>
Original Image	Output Image	24.75
		
Original Image	Output Image	22.43
		
Input Image	Output Image	23.01
		
Input Image	Output Image	20.10
		
Input Image	Output Image	23.48
