Image Denoising Model Report

Deepak Nailwal

21113043

Introduction:

Image denoising is a computer vision task in which we remove the unwanted noise from the image while preserving its essential detail.

In our case we deal with extremely low light images and our task was to remove the darkness from the image.

In this project I used Multi Wavelet CNN architecture and incorporated it with Channel Attention Mechanism. We analyze our result and trained our model on different losses like MAE, Perceptual Loss.

Methodology Use:

- Architecture Used:

I used the MWCNN architecture inspired from the MWCNN research paper but in this I experimented this architecture with the Channel Attention Mechanism.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         []

 conv_block (Conv_block)     (None, 256, 256, 64)      76160     ['input_1[0][0]']

 dwt_downsampling (DWT_down  (None, 128, 128, 256)     0         ['conv_block[0][0]']
 sampling)

 conv_block_1 (Conv_block)   (None, 128, 128, 128)     592256    ['dwt_downsampling[0][0]']

 dwt_downsampling_1 (DWT_do  (None, 64, 64, 512)       0         ['conv_block_1[0][0]']
 wnsampling)

 conv_block_2 (Conv_block)   (None, 64, 64, 256)       2368256   ['dwt_downsampling_1[0][0]']

 dwt_downsampling_2 (DWT_do  (None, 32, 32, 1024)      0         ['conv_block_2[0][0]']
 wnsampling)

 conv_block_3 (Conv_block)   (None, 32, 32, 512)       9471488   ['dwt_downsampling_2[0][0]']

 dwt_downsampling_3 (DWT_do  (None, 16, 16, 2048)      0         ['conv_block_3[0][0]']
 wnsampling)

 conv_block_4 (Conv_block)   (None, 16, 16, 512)       1419008   ['dwt_downsampling_3[0][0]']
                                                       0

 batch_normalization (Batch  (None, 16, 16, 512)       2048      ['conv_block_4[0][0]']
 Normalization)

 conv_block_5 (Conv_block)   (None, 16, 16, 512)       7112192   ['batch_normalization[0][0]']

 conv2d_24 (Conv2D)          (None, 16, 16, 2048)      9439232   ['conv_block_5[0][0]']

 iwt_upsampling (IWT_upsamp  (None, 32, 32, 512)       0         ['conv2d_24[0][0]']
 ling)

 add (Add)                   (None, 32, 32, 512)       0         ['iwt_upsampling[0][0]',
                                                                  'conv_block_3[0][0]']

 conv_block_6 (Conv_block)   (None, 32, 32, 512)       7112192   ['add[0][0]']
```

```
conv2d_29 (Conv2D)          (None, 32, 32, 1024)        4719616   ['conv_block_6[0][0]']

iwt_upsampling_1 (IWT_upsa  (None, 64, 64, 256)         0         ['conv2d_29[0][0]']
mpling)

add_1 (Add)                 (None, 64, 64, 256)         0         ['iwt_upsampling_1[0][0]',
                                                                   'conv_block_2[0][0]']

conv_block_7 (Conv_block)   (None, 64, 64, 256)         1778432   ['add_1[0][0]']

conv2d_34 (Conv2D)          (None, 64, 64, 512)         1180160   ['conv_block_7[0][0]']

iwt_upsampling_2 (IWT_upsa  (None, 128, 128, 128)       0         ['conv2d_34[0][0]']
mpling)

add_2 (Add)                 (None, 128, 128, 128)       0         ['iwt_upsampling_2[0][0]',
                                                                   'conv_block_1[0][0]']

conv_block_8 (Conv_block)   (None, 128, 128, 128)       444800    ['add_2[0][0]']

conv2d_39 (Conv2D)          (None, 128, 128, 256)       295168    ['conv_block_8[0][0]']

iwt_upsampling_3 (IWT_upsa  (None, 256, 256, 64)        0         ['conv2d_39[0][0]']
mpling)

add_3 (Add)                 (None, 256, 256, 64)        0         ['iwt_upsampling_3[0][0]',
                                                                   'conv_block[0][0]']

conv_block_9 (Conv_block)   (None, 256, 256, 64)        111296    ['add_3[0][0]']

conv2d_44 (Conv2D)          (None, 256, 256, 128)       73856     ['conv_block_9[0][0]']

conv2d_45 (Conv2D)          (None, 256, 256, 3)         387       ['conv2d_44[0][0]']
```

- MWCNN (Multi-Level Wavelet CNN): It embeds wavelet transform into the CNN architecture to reduce the resolution of feature maps while increasing the receptive field, often based on U-Net architecture for image restoration tasks.

- Channel Attention: Channel Attention mechanism is used to enhance the performance of the model. While filtering the image not all the features are important for the task. So, this mechanism assigns the weightage to the channels so that more importance will be given to the important features.

```python
class SEBlock(tf.keras.layers.Layer):
    def __init__(self, filters, reduction=16, **kwargs):
        super(SEBlock, self).__init__(**kwargs)
        self.filters = filters
        self.reduction = reduction
        self.global_avg_pool = GlobalAveragePooling2D()
        self.reshape = Reshape((1, 1, filters))
        self.dense1 = Dense(filters // reduction, activation='relu', kernel_initializer='he_normal', use_bias=False)
        self.dense2 = Dense(filters, activation='sigmoid', kernel_initializer='he_normal', use_bias=False)

    def call(self, input_tensor):
        se = self.global_avg_pool(input_tensor)
        se = self.reshape(se)
        se = self.dense1(se)
        se = self.dense2(se)
        return Multiply()([input_tensor, se])

    def get_config(self):
        config = super(SEBlock, self).get_config()
        config.update({
            'filters': self.filters,
            'reduction': self.reduction
        })
        return config
```

As shown in the code I used the SE Block Class which will calculate the attention parameters by converting the channels into (1*1*n_channels) patches and then passing this through Dense layer of perceptron.

For loss calculation we use the losses given below:

- MAE: Using Mean Squared Error which is calculating the direct difference between the pixels value of the image.
- Perceptual Loss: This is implemented using the Feature Maps extracted using the VGG 19 this is used to capture the edges details from the images.

```
vgg=VGG19(include_top=False, weights='imagenet', input_shape=(256,256,3))
vgg.trainable=False
def perceptual_loss(y_true, y_pred):
    true_features=vgg(y_true)
    pred_features=vgg(y_pred)
    return MeanSquaredError()(true_features, pred_features)
```

In the above code we are capturing the feature maps from the VGG19 of true image and predicted image and than calculating the mean squared error of the these feature maps.

- SSIM Loss: Structural Similarity Loss this is also used to capture the structural similarity loss of the image.

- Wavelet Trasformation Code:
  - First is of Discrete Wavelet Transformation (Downscaling) , it is making the 4 new tensors from the input tensors by taking the pixel values corresponding to, (odd_rows,odd_cols),(even_rows,odd_cols),(even_rows,even_cols),(odd_rows,even_cols).
    And then calculating the LL(Low Level Details), LH(Horizontal Details), HL(Vertical Detail),HH(Diagonal Details) details of the image.
  - Second is of Inverse Wavelet Transformation (Upscaling), now this code doing reverse of the downsampling wavelet operation by calculating the pixels value from the LH,HL,LL,HH.

```python
class DWT_downsampling(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, x):
        x1 = x[:, 0::2, 0::2, :] #x(2i-1, 2j-1)
        x2 = x[:, 1::2, 0::2, :] #x(2i, 2j-1)
        x3 = x[:, 0::2, 1::2, :] #x(2i-1, 2j)
        x4 = x[:, 1::2, 1::2, :] #x(2i, 2j)

        x_LL = x1 + x2 + x3 + x4
        x_LH = -x1 - x3 + x2 + x4
        x_HL = -x1 + x3 - x2 + x4
        x_HH = x1 - x3 - x2 + x4

        return Concatenate(axis=-1)([x_LL, x_LH, x_HL, x_HH])
```

```python
class IWT_upsampling(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, x):
        x_LL = x[:, :, :, 0:x.shape[3]//4]
        x_LH = x[:, :, :, x.shape[3]//4:x.shape[3]//4*2]
        x_HL = x[:, :, :, x.shape[3]//4*2:x.shape[3]//4*3]
        x_HH = x[:, :, :, x.shape[3]//4*3:]

        x1 = (x_LL - x_LH - x_HL + x_HH)/4
        x2 = (x_LL - x_LH + x_HL - x_HH)/4
        x3 = (x_LL + x_LH - x_HL - x_HH)/4
        x4 = (x_LL + x_LH + x_HL + x_HH)/4

        y1 = K.stack([x1,x3], axis=2)
        y2 = K.stack([x2,x4], axis=2)
        shape = K.shape(x)
        return K.reshape(K.concatenate([y1,y2], axis=-1), K.stack([shape[0], shape[1]*2, shape[2]*2, shape[3]//4]))
```

Training:

First Way:

In the training of the model, I trained the model for first 20 epoch on MAE loss only but due to this the Model was not able to capture the edges detail of the

image and after that for another 20 epoch I use Perceptual loss with this keeping the weightage of perceptual loss half of the MAE loss and after that for last 10 epoch I assigned the weightage of 100:10 of MAE to Perceptual.

In the last 10 Epoch My Models started overfitting the data so I proposed the model that is trained on 40 epoch.

I use Reduced LR on Plateau which is decreasing my Learning Rate by the factor of 0.67 after every 2 epoch of non-improvement in my Validation Loss.

Used Adam Optimizer keeping the parameter used in the MWCNN Research Paper.

Dataset is of 1552 Augmented Training Images 388 Validation Images with Batch size of 24.

Second Way:

In this I have trained the Model on MAE and Combined Loss (Perceptual + SSIM). Starting from the learning rate of 0.001 with weightage 10:1 (MAE: Combined Loss) and keeping the parameter of Adam Optimizer same as above after reaching to the epoch 20 validation loss was not improving but we keep trained the model for 9 more epoch until early stopping call.

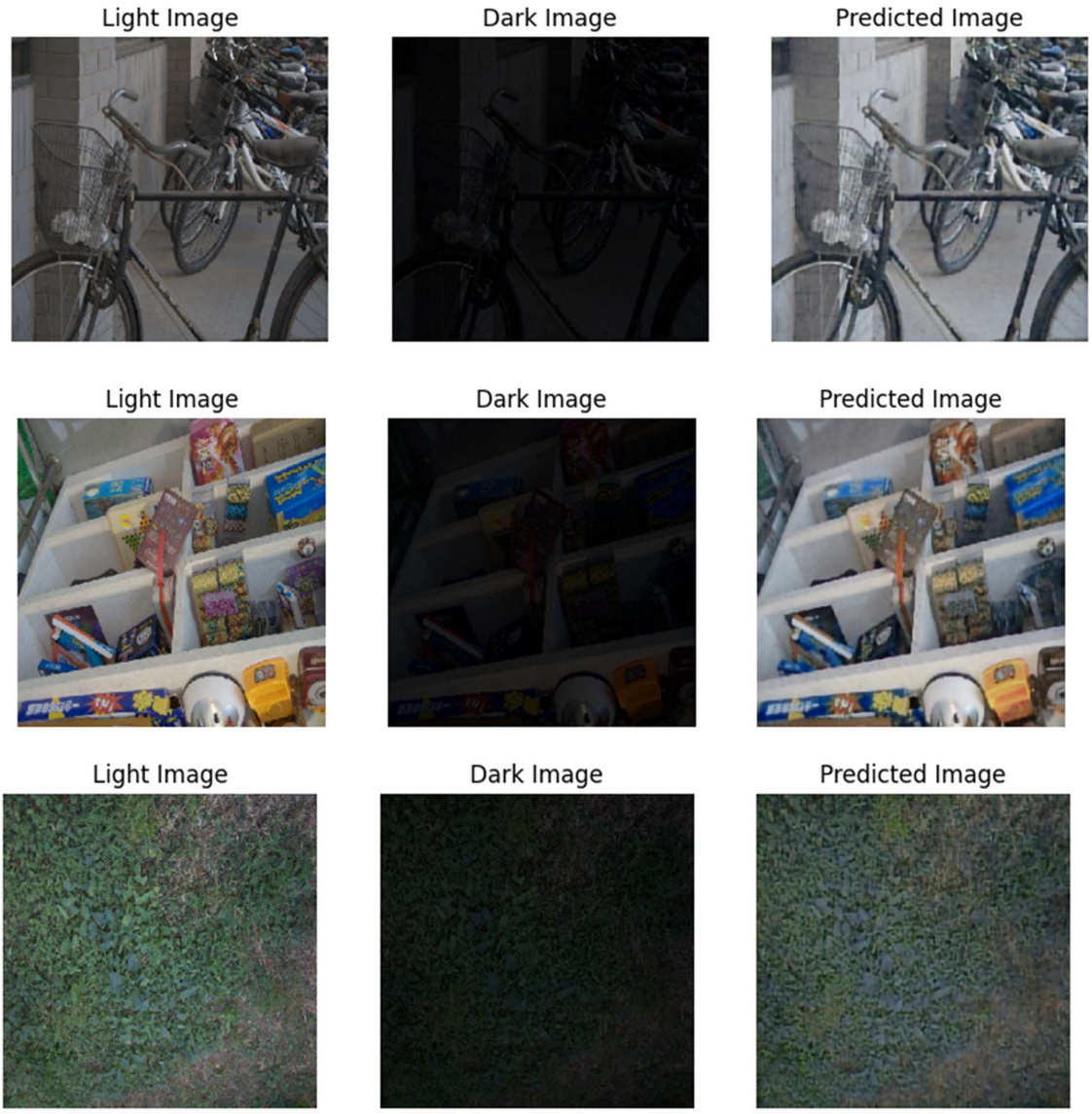
Dataset is of the size 1843 Augmented Training Images and 97 validation Images.

Results:

Results on Training Data:

Architecture	PSNR	MSE	MAE
MWCNN+ Channel Attention	22.79	0.0079	0.0659

Image Result:



Discussion:

In this project I implement MWCNN architecture with Channel Attention mechanism.

After trying training in different ways every time the model starts overfitting after reaching to the PSNR of 22.When Model is reaching to the PSNR of 24 to 25 it is not giving the good result on validation data.