VLG Open Project 2024

IMAGE DENOISING

Somesh Verma

s_verma@ma.iitr.ac.in
21323035

Low-light image enhancement is a critical task in computer vision, aiming to improve the visual quality of images captured in suboptimal lighting conditions. The goal is to enhance these images while preserving natural color and details, ultimately improving their usability in various applications such as photography, surveillance, and medical imaging.

My main goal in this project was to explore basics of different DL models used in Image denoising. This project involved the implementation and comparison of different neural network architectures, including basic AutoEncoder techniques, dnCnn, Deep Curve Estimation (Zero-DCE) model and a transformer-based network with perceptual and Charbonnier losses. The primary goal was to improve the Peak Signal-to-Noise Ratio (PSNR) of denoised images.

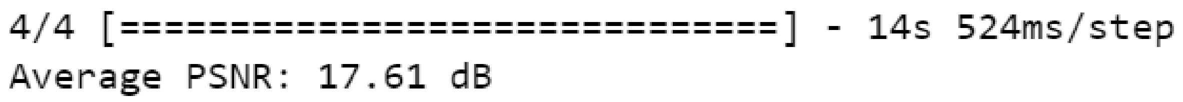
Background Research

- **Basic Autoencoder**

An autoencoder is an unsupervised neural network designed to learn efficient representations of data, often for the purpose of dimensionality reduction or denoising. The network consists of two main parts: an encoder and a decoder. The encoder compresses the input image into a latent representation, and the decoder reconstructs the image from this representation.

In its architecture, multiple convolutional layers with ReLU activation functions are used to capture spatial features. Each layer progressively reduces the image size while increasing the number of feature maps. This part acts as an **Encoder**. This compressed image is passed through Symmetrical deconvolutional layers. To reconstruct the image from the latent representations (called Bottleneck). This end part acts as a **Decoder**.

When applied on the given data the obtained PSNR value was 17.61 dB.



```
4/4 [==============================] - 14s 524ms/step
Average PSNR: 17.61 dB
```

This result can be improved a lot by tuning the hyperparameters, the above result is just the initial result.

The autoencoder is trained to minimise the reconstruction loss, typically measured as the mean squared error between the input and output images.

- **DnCNN**

DnCNN is a deep convolutional neural network designed specifically for image denoising. It utilizes residual learning, where the network is trained to predict the noise component of an image rather than the denoised image itself. This approach simplifies the learning process and improves performance.

In its architecture, we start with multiple convolutional layers, followed by batch normalisation. These layers extract hierarchical features from the input image. Then using the concept of Residual Learning the network learns to predict the noise present in the image. The final denoised image is obtained by subtracting the predicted noise from the input noisyimage.

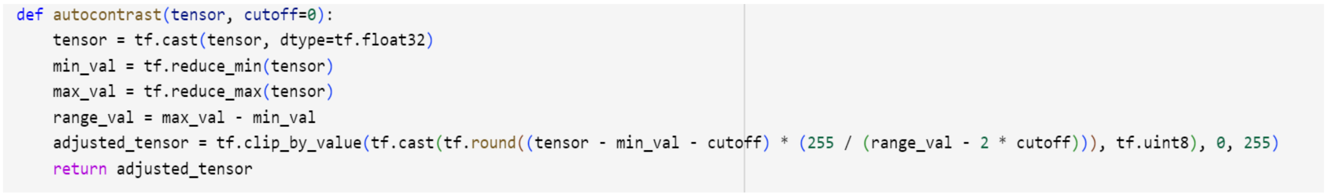
DnCNN is trained using a large dataset of noisy and clean image pairs, with the loss function being the mean squared error between the predicted noise and the actual noise.

Key Features of the Model Implemented

AutoContrast Function and Augmentation

The `autocontrast` function is designed to enhance the contrast of an image automatically. Here's a breakdown of its purpose and implementation:

- **Purpose**: In low-light conditions, images often suffer from poor contrast, making details less discernible. Autocontrast aims to adjust the contrast of the image such that the full intensity range (0-255) is utilized, thereby improving the visibility of details.
- **Implementation**: The function calculates the minimum and maximum pixel values in the image and scales all pixel values to stretch across the entire intensity range, effectively enhancing the contrast.



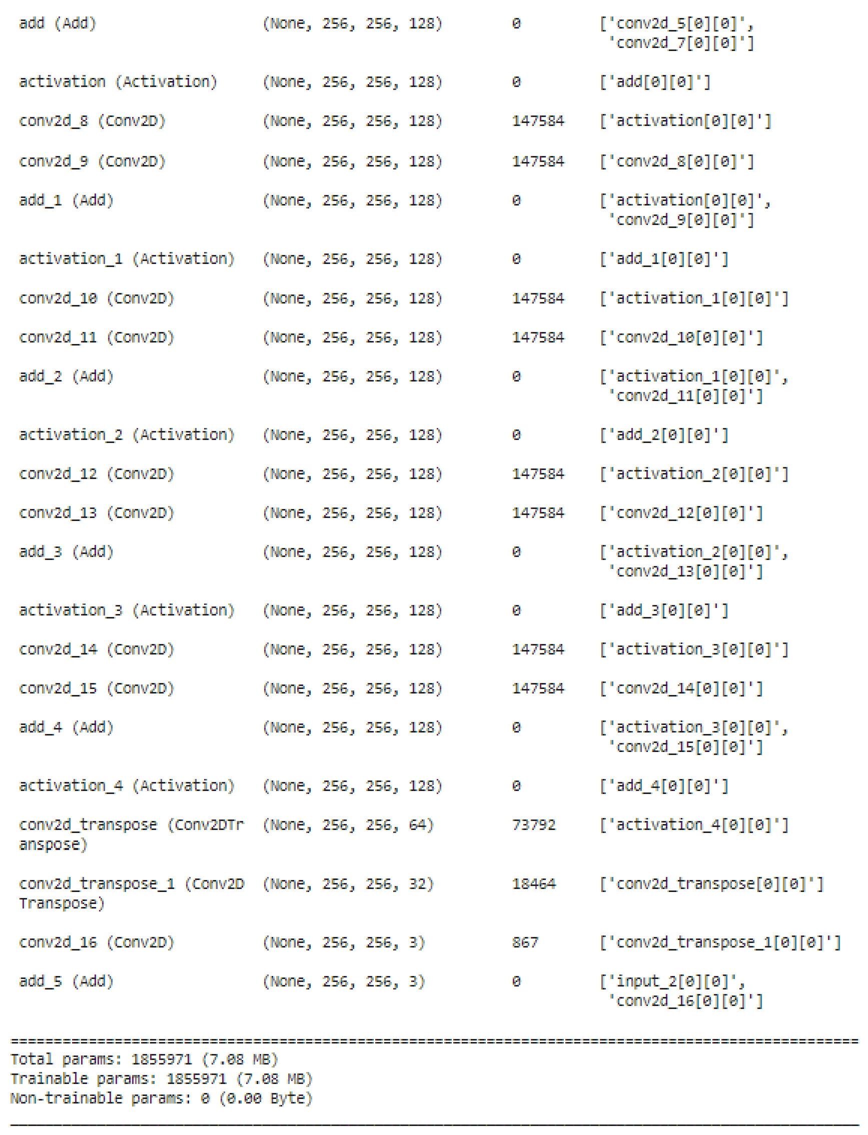
```python
def autocontrast(tensor, cutoff=0):
    tensor = tf.cast(tensor, dtype=tf.float32)
    min_val = tf.reduce_min(tensor)
    max_val = tf.reduce_max(tensor)
    range_val = max_val - min_val
    adjusted_tensor = tf.clip_by_value(tf.cast(tf.round((tensor - min_val - cutoff) * (255 / (range_val - 2 * cutoff))), tf.uint8), 0, 255)
    return adjusted_tensor
```

This preprocessing step is very impactful. After this step, we use **Random Cropping.** This technique augments the dataset by taking random crops of the images, providing more varied training samples, and helping the model generalise better.

Transformer Network Architecture

The core of the code is the transformer network, a deep convolutional neural network designed to enhance low-light images. Here's a detailed look at its components:

- **Convolutional Layers**: The initial layers use convolution operations to extract low-level features from the input image. These features include edges, textures, and patterns.
- **Residual Blocks**: Residual blocks are fundamental components in deep learning architectures that help learn residual functions. By adding the input of the block to its output, the block learns to map the input directly if needed, preserving the original information. They also facilitate the flow of gradients during backpropagation, making it easier to train deeper networks.
- **Deconvolutional Layers**: These layers upsample the feature maps to reconstruct the enhanced image. They play a role in translating the learned features back into pixel values.



```
 add (Add)                   (None, 256, 256, 128)        0         ['conv2d_5[0][0]',
                                                                      'conv2d_7[0][0]']

 activation (Activation)     (None, 256, 256, 128)        0         ['add[0][0]']

 conv2d_8 (Conv2D)           (None, 256, 256, 128)        147584    ['activation[0][0]']

 conv2d_9 (Conv2D)           (None, 256, 256, 128)        147584    ['conv2d_8[0][0]']

 add_1 (Add)                 (None, 256, 256, 128)        0         ['activation[0][0]',
                                                                      'conv2d_9[0][0]']

 activation_1 (Activation)   (None, 256, 256, 128)        0         ['add_1[0][0]']

 conv2d_10 (Conv2D)          (None, 256, 256, 128)        147584    ['activation_1[0][0]']

 conv2d_11 (Conv2D)          (None, 256, 256, 128)        147584    ['conv2d_10[0][0]']

 add_2 (Add)                 (None, 256, 256, 128)        0         ['activation_1[0][0]',
                                                                      'conv2d_11[0][0]']

 activation_2 (Activation)   (None, 256, 256, 128)        0         ['add_2[0][0]']

 conv2d_12 (Conv2D)          (None, 256, 256, 128)        147584    ['activation_2[0][0]']

 conv2d_13 (Conv2D)          (None, 256, 256, 128)        147584    ['conv2d_12[0][0]']

 add_3 (Add)                 (None, 256, 256, 128)        0         ['activation_2[0][0]',
                                                                      'conv2d_13[0][0]']

 activation_3 (Activation)   (None, 256, 256, 128)        0         ['add_3[0][0]']

 conv2d_14 (Conv2D)          (None, 256, 256, 128)        147584    ['activation_3[0][0]']

 conv2d_15 (Conv2D)          (None, 256, 256, 128)        147584    ['conv2d_14[0][0]']

 add_4 (Add)                 (None, 256, 256, 128)        0         ['activation_3[0][0]',
                                                                      'conv2d_15[0][0]']

 activation_4 (Activation)   (None, 256, 256, 128)        0         ['add_4[0][0]']

 conv2d_transpose (Conv2DTr  (None, 256, 256, 64)         73792     ['activation_4[0][0]']
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 256, 256, 32)         18464     ['conv2d_transpose[0][0]']
 Transpose)

 conv2d_16 (Conv2D)          (None, 256, 256, 3)          867       ['conv2d_transpose_1[0][0]']

 add_5 (Add)                 (None, 256, 256, 3)          0         ['input_2[0][0]',
                                                                      'conv2d_16[0][0]']

==================================================================================================
Total params: 1855971 (7.08 MB)
Trainable params: 1855971 (7.08 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Loss Functions

Two specialised loss functions are used to guide the training model:

- **Perceptual Loss**: This loss leverages the VGG19 network pre-trained on ImageNet to compute high-level feature differences between the true and predicted images. It ensures that the enhanced image not only matches the pixel values of the ground truth but also preserves perceptual details such as textures and edges.

- **Charbonnier Loss**: A smooth approximation of L1 loss, which is less sensitive to outliers and provides better gradient behaviour.

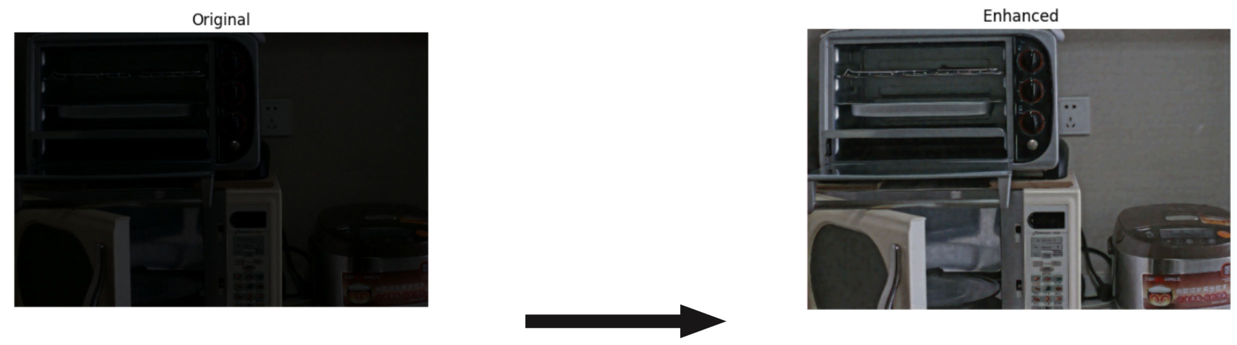
A weighted combination of perceptual and Charbonnier losses is taken to balance fine detail preservation and perceptual quality. And since **PSNR** is the criteria for this project, it is used as a metric for the evaluation of our model.

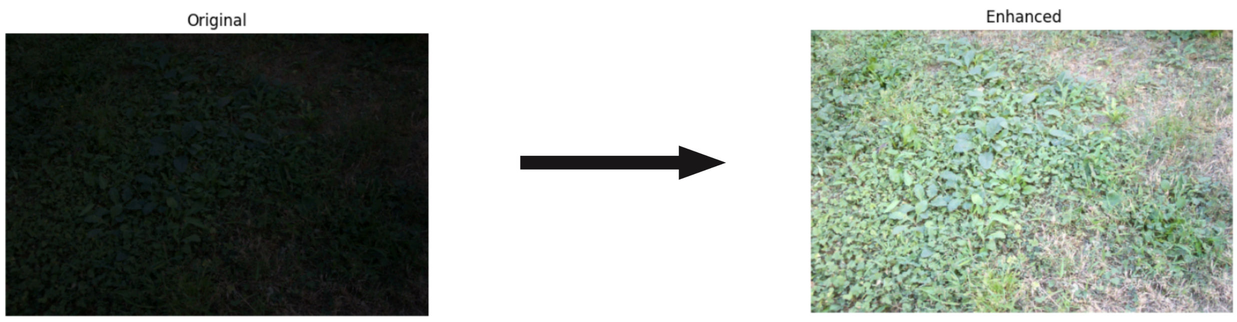
Conclusion

The overall average PSNR calculated over the validation Images sums up to 18.87 dB

```
23/23 [==============================] - 12s 486ms/step - loss: 3.8035 - peak_signal_noise_ratio: 18.8699
Average PSNR over validation dataset: 18.87 dB
```

Now checking our model over some of the images from the validation set.





As mentioned earlier autocontrast function helps significantly in some of the case like in the above case would look something like-