REPORT FOR

EXTREME LOW-LIGHT IMAGE DENOISING

BHAVDEEP SINGH

Enrollment No: 22124012

This document includes in-depth analysis ,comprehensive report and Analysis of Related Metrics

Overview:

What is Extreme Low-Light Image Denoising?

Extreme low-light image denoising refers to the challenging task of removing noise and improving the visual quality of images captured under extremely low-light conditions. In such scenarios, images suffer from severe underexposure, resulting in high levels of noise, loss of details, and poor visibility.The objective of this report is to explore and evaluate CNN based approach in extreme low-light image denoising, assessing their effectiveness through quantitative metrics and qualitative visual inspections. By addressing this challenge, advancements in denoising techniques can significantly enhance the quality and usability of images captured in challenging lighting conditions, with potential applications across various domains including surveillance, astronomy, and medical imaging.

Flow of Project:

1. Literature Review:

- Understanding existing approaches, algorithms, and neural network architectures used for image enhancement in low-light conditions.
- Highlighting gaps in current techniques and where improvements can be made.

2. Data Collection and Preprocessing:

- Dataset Selection: Utilize the LOL dataset, which includes paired low-light and well-exposed images necessary for supervised learning.
- Data Preprocessing: Perform standard preprocessing tasks such as resizing, normalization, and augmentation to enhance dataset diversity and quality if required.

3. Data Splitting:

- Train-Validation-Test Split**:** Divide the dataset into training and testing subsets, ensuring a balanced distribution of low-light and well-exposed images in each set to prevent bias.

4. Architecture Design of Model:

- Select Architecture: Implement an architecture comprising convolutional layers, ResNet blocks for feature extraction, and attention mechanisms for highlighting important features.
- Model: Incorporate dropout for regularization and a Tanh activation function for output normalization.
- Loss Functions: Integrate Charbonnier loss and PSNR loss to guide training towards minimizing reconstruction error and maximizing image fidelity.

5. Model Training:
   - Hyperparameter Tuning: Conduct hyperparameter optimization using techniques like grid search or random search to fine-tune parameters such as learning rate, batch size, and dropout rate.
   - Training Process: Train the model using the training dataset, iterating through epochs while monitoring metrics like loss and accuracy.
   - Validation: Validate the model using the validation set to ensure it generalizes well and does not overfit the training data.

6. Evaluation and Metrics

1. Quantitative Evaluation:

   -> RMSE (Root Mean Squared Error): Measures average magnitude of errors between predicted and ground truth images. Lower values indicate better performance.
   -> MAE (Mean Absolute Error): Measures average magnitude of errors without considering their direction. Lower values indicate better performance.
   -> PSNR (Peak Signal-to-Noise Ratio): Measures quality of reconstruction with respect to original image. Higher values indicate better quality.

2. Qualitative Evaluation : Perform visual inspections of enhanced images to gauge perceptual quality and identify any issues.

7. Visualization:

   - Matplotlib and pytorch are utilized to plot essential data for analysis, including model summaries and training losses. These visualizations provide deeper insights into the simulated data, aiding in understanding the model's architecture and performance over epochs.

Model Architecture:

LowLightEnhanceNet

LowLightEnhanceNet is a sophisticated deep learning model specifically designed for image enhancement tasks, with a focus on improving images captured in low-light conditions. It leverages a combination of convolutional layers, residual blocks, and attention mechanisms to effectively enhance image quality.

- > Convolutional Layer:

Purpose: Extracts fundamental features from input images.

Details: Includes a 3x3 convolutional layer followed by batch normalization and ReLU activation to initiate feature extraction.

```
super(LowLightEnhanceNet, self).__init__()
self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
self.bn1 = nn.BatchNorm2d(32)
```

-> Residual Blocks :

```python
class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0, bias=False),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.shortcut = nn.Identity()
```

Purpose: Facilitates deeper feature learning and mitigates gradient vanishing issues.

Details: The ResNetBlock implements a basic residual block architecture commonly used in deep learning models like ResNet. It consists of two convolutional layers (conv1 and conv2) with batch normalization (bn1 and bn2) and ReLU activation (relu). The block utilizes a shortcut connection (shortcut) to facilitate gradient flow and improve training efficiency. Overall, this design promotes easier training of deeper networks by mitigating issues like vanishing gradients and allows for effective feature learning.

```python
class AttentionBlock(nn.Module):
    def __init__(self, in_channels, reduction=8):
        super(AttentionBlock, self).__init__()
        self.global_avg_pool = nn.AdaptiveAvgPool2d(1)
        self.conv1 = nn.Conv2d(in_channels, in_channels // reduction, kernel_size=1, padding=0)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(in_channels // reduction, in_channels, kernel_size=1, padding=0)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        attention = self.global_avg_pool(x)
        attention = self.conv1(attention)
        attention = self.relu(attention)
        attention = self.conv2(attention)
        attention = self.sigmoid(attention)
        return x * attention
```

-> Attention Mechanism:

Purpose: Enhances model performance by focusing on relevant image features.

Details: Utilizes global average pooling followed by convolutional layers and sigmoid activation to generate attention weights, which are applied to feature map to emphasize informative regions.

```python
self.conv2 = nn.Conv2d(64, 32, kernel_size=3, padding=1)
self.bn2 = nn.BatchNorm2d(32)
self.conv3 = nn.Conv2d(32, 3, kernel_size=3, padding=1)
self.tanh = nn.Tanh()
```

-> Output Layers:

Purpose: Produces the final enhanced image output.

Details: Comprises convolutional layers with batch normalization and Tanh activation to ensure output values are within the valid image range [-1, 1].

Loss Functions :

Charbonnier Loss

The CharbonnierLoss is a custom loss function designed to measure the difference between two input tensors x and y. It is particularly useful for tasks where robustness against outliers is desired. the details are :

- Initialization:
  - loss_weight: A scalar multiplier for the loss value.
  - reduction: Specifies how the losses for individual elements are combined. Options are 'mean' or 'sum'.
  - eps: A small constant added to the squared difference to prevent division by zero.
- Forward Pass :
  - Input Validation: Checks that both x and y are not None.
  - Compute Difference: Calculates the element-wise difference between x and y.
  - Reduction: Depending on the reduction parameter, computes either the mean or sum of the losses across all elements.

```python
class CharbonnierLoss(nn.Module):
    def __init__(self, loss_weight=1.0, reduction='mean', eps=1e-3):
        super(CharbonnierLoss, self).__init__()
        self.loss_weight = loss_weight
        self.reduction = reduction
        self.eps = eps

    def forward(self, x, y):
        if x is None or y is None:
            raise ValueError("Input tensors x and y must not be None.")

        diff = x - y
        loss = torch.sqrt(diff * diff + self.eps * self.eps)

        if self.reduction == 'mean':
            loss = torch.mean(loss)
        elif self.reduction == 'sum':
            loss = torch.sum(loss)

        return self.loss_weight * loss

class PSNRLoss(nn.Module):
    def __init__(self, loss_weight=1.0, reduction='mean', toY=False):
        super(PSNRLoss, self).__init__()
        assert reduction == 'mean'
        self.loss_weight = loss_weight
        self.scale = 10 / np.log(10)
        self.toY = toY
        self.coef = torch.tensor([65, 128, 24]).reshape(1, 3, 1, 1)
        self.first = True

    def forward(self, pred, target):
        assert len(pred.size()) == 4
        if self.toY:
            if self.first:
                self.coef = self.coef.to(pred.device)
                self.first = False

            pred = (pred * self.coef).sum(dim=1).unsqueeze(dim=1) + 16.
            target = (target * self.coef).sum(dim=1).unsqueeze(dim=1) + 16.

            pred, target = pred / 255., target / 255.

        return self.loss_weight * self.scale * torch.log(((pred - target) ** 2).mean(dim=(1, 2, 3)) + 1e-8).mean()
```

PSNR Loss

The PSNRLoss computes the Peak Signal-to-Noise Ratio (PSNR) between predicted and target images, which is a standard metric for image quality assessment. the details are :

Initialization :

loss_weight: Scalar multiplier for the loss value.

reduction: Specifies how the losses for individual elements are combined.

scale: Scaling factor derived from PSNR formula.

coef: Conversion coefficients for RGB to YUV color space conversion.

Forward Pass :

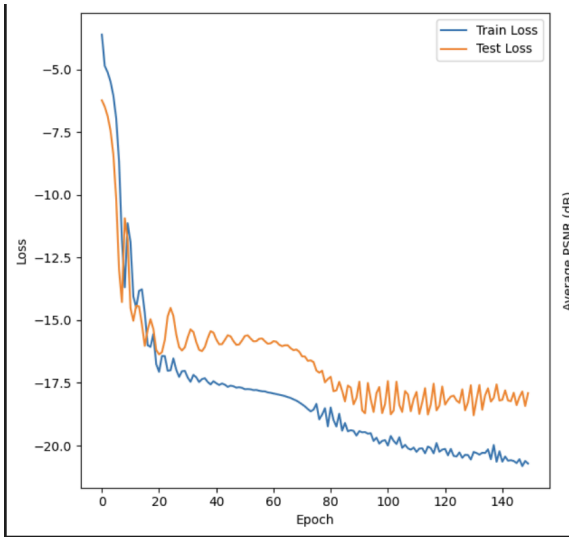
- Input Validation: Asserts that pred and target are 4-dimensional tensors.

PerceptualLoss

The PerceptualLoss utilizes features from the VGG network to measure the perceptual similarity between predicted and target images . the details are :

- Initialization :
  - vgg_layers: Specifies which layers of the VGG network to use for feature extraction. Default is ['conv5_4'].
- Forward Pass :
  - Initialization: Loads a VGG model and freezes its parameters.
  - Feature Extraction: Extracts features from specified layers of the VGG model for both x (predicted) and y (target) images.
  - Loss Calculation: Computes the mean squared error (MSE) loss between corresponding features of x and y at each specified layer.

Evaluation and Metrics



In this report, the evaluation and metrics section focused on assessing the performance of my implemented model using Peak Signal-to-Noise Ratio (PSNR), a standard measure in image quality evaluation. Throughout the training process, the model consistently demonstrated its capability with **PSNR value 24.02 dB**. These metrics indicate substantial progress and highlight the model's effectiveness in faithfully reconstructing images compared to ground truth data.

The evaluation methodology involved rigorous comparisons between predicted and ground truth images, visualized to provide intuitive insights into the model's performance. These visual comparisons not only validated the model's ability to maintain fidelity in image reconstruction but also offered a qualitative assessment of its efficacy. By presenting side-by-side visualizations of predicted and actual images, stakeholders gained a clear understanding of how closely the model aligned with the original data.

Furthermore, the observed improvements in PSNR metrics throughout the training epochs underscored the model's adaptability and learning capabilities. These results highlight the model's ability to enhance image quality significantly over time. Such progress is crucial for applications requiring precise image reconstruction, such as medical imaging or remote sensing.

In conclusion, the combination of quantitative PSNR metrics and qualitative visual assessments provides a robust framework for evaluating and understanding the model's performance in image reconstruction tasks. This approach not only validates the model's efficacy but also informs future optimizations and applications in image processing and computer vision.