PROJECT : IMAGE DENOISING

By: Harsh Pratap Rathore, EE 3Y (22115063)

Introduction

This project focuses on utilizing deep learning techniques to enhance low-light images, improving their clarity and usability. The project involves training a model using a dataset of low-light and high-light images, evaluating its performance using metrics such as PSNR (Peak Signal-to-Noise Ratio), and deploying the model for inference. The following report details the steps taken, the methodology employed.

Mounting the drive containing dataset

The initial step in our project involves importing the dataset, which is stored on Google Drive. We need to mount the Google Drive to access the dataset files.

Importing Necessary Libraries

- os, random, numpy: For file handling, randomization, and numerical operations.
- glob: To retrieve files matching specified patterns.
- PIL (Python Imaging Library), ImageOps: For image manipulation.
- matplotlib.pyplot: For plotting images.
- cv2 (OpenCV): For image processing tasks.
- tensorflow, keras, layers: For building and training neural network models.

Image Preprocessing Function

This function is responsible for loading and preprocessing images.

- **tf.io.read_file(image_path)**: This function reads the image file from the specified path as a byte string.
- **tf.image.decode_png(image, channels=3)**: Decodes the byte string to a PNG image tensor. The “channels=3” argument ensures the image is loaded with RGB.
- **tf.image.resize(images=image, size=[256, 256])**: Resizes the image to 256x256 pixels, a standard size for input into neural networks.
- **image / 255.0**: Normalizes the pixel values of the image to the range [0, 1].

Data Pipeline Creation Function

This function efficiently loads, preprocess, and batch the images for training or inference.

- **tf.data.Dataset.from_tensor_slices(low_light_images)**: This function creates a “tf.data.Dataset” from the list of image paths for low_light images.
- **dataset.map(preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)**: The map function applies the “preprocess_image” function to each element in the dataset. The “num_parallel_calls=tf.data.AUTOTUNE” argument enables parallel processing, optimizing the pipeline's performance by utilizing multiple CPU cores.
- **dataset.batch(16, drop_remainder=True)**: Batches the dataset into groups of 16 images.

Dataset Preparation

• **Image Path Retrieval:** We will take first 30 images for testing our results, from 31 to 400 images for training and from 400 onwards for validation.

• **Dataset Creation:**

- create_dataset(train_low_light_images): Preprocesses and batches training images.
- create_dataset(val_low_light_images): Preprocesses and batches validation images.

• **Print Dataset Details:** Outputs information about the datasets to confirm correct loading and preprocessing.

Building the Deep Curve Estimation (DCE) Network

This function “build_image_enhancement_net()” defines a Deep Curve Estimation network architecture using the Keras API from TensorFlow.

- **Input Layer:** Defines the “input_image” variable for the network with “shape [None, None, 3]”.
- **Convolutional Layers:** Sequentially defines several convolutional layers (conv_layer1 to conv_layer4). Each layer uses a 3x3 kernel, “relu” activation function, and same padding to ensure that the output has the same spatial dimensions as the input.
- **Intermediate Concatenation Layers:** Concatenates the output of one with another along the last axis (-1), corresponding to the channel axis.
- **Output Layer:** Final convolutional layer with 24 filters, 3x3 kernel size, tanh activation function, and same padding.
- **Model Compilation:** Returns a Keras Model object initialized with the input and output layers. This model defines the complete architecture of the DCE network.

Function to Calculate Color Constancy Loss

- **Calculate Mean RGB Values**: Computes the mean red (mean_r), green (mean_g), and blue (mean_b) values across the image batch using tf.reduce_mean.
- **Calculate Squared Differences**: Computes d_rg, d_rb, and d_gb as squared differences between mean RGB values.
- **Calculate Color Constancy Loss**: Computes the color constancy loss as sqrt(d_rg^2 + d_rb^2 + d_gb^2), aiming to maintain color consistency across images.

Function to Calculate Exposure Loss

- **Calculate Mean Intensity**: Computes the mean intensity of the input image_tensor across the RGB channels using tf.reduce_mean.
- **Average Pooling**: Performs average pooling on mean_intensity to compute the overall mean with a kernel size (ksize) of 16x16 and strides (strides) of 16.
- **Calculate Exposure Loss**: Computes the exposure loss as the mean squared difference between pooled_mean and target_mean. This loss helps in adjusting the exposure of the images.

<u>Function to Calculate Illumination Smoothness Loss</u>

- **Batch Size and Image Dimensions**: Retrieves the batch size (batch_size), height (h_x), and width (w_x) of the image_tensor.
- **Number of Horizontal and Vertical Pairwise Differences**: Computes count_h and count_w as the total number of horizontal and vertical pairwise differences respectively.
- **Calculate Horizontal and Vertical Total Variation (TV)**: Computes h_tv and w_tv as the sum of squared differences between horizontally and vetically adjacent pixels across all channels in each image of the batch respectively.
- **Convert Counts to Float32**: Converts batch_size, count_h, and count_w to float32 for division to ensure compatibility with TensorFlow operations.
- **Calculate Illumination Smoothness Loss**:

$$\text{illumination\_smoothness\_loss} = \frac{2\times(h\_tv/\text{count\_h}+w\_tv/\text{count\_w})}{\text{batch\_size}}.$$

    - This loss measures how smoothly illumination changes across adjacent pixels in the image tensor, promoting spatial consistency in illumination.

<u>Spatial Consistency Loss Class</u>

- **Constructor (__init__ method)**:
    - Initializes the class as a subclass of keras.losses.Loss with reduction="none", indicating that the loss will be computed per batch sample without reducing across the batch.
    - Defines four convolutional kernels (left_kernel, right_kernel, up_kernel, down_kernel) using TensorFlow constants (tf.constant). These kernels are used to compute differences in the spatial consistency of illumination between the original (y_true) and enhanced (y_pred) images.

- **Mean Intensity Calculation**: Computes the mean intensity (original_mean, enhanced_mean) of the original (y_true) and enhanced (y_pred) images along the channel axis (axis=3) using tf.reduce_mean.
- **Average Pooling**: Performs average pooling (tf.nn.avg_pool2d) on original_mean and enhanced_mean with a kernel size (ksize) of 4x4 and strides of 4x4 (strides=[1, 4, 4, 1]) to obtain pooled representations (original_pool, enhanced_pool) with reduced spatial dimensions.
- **Spatial Differences Computation**:
    - Computes spatial differences (d_original_left, d_original_right, d_original_up, d_original_down) between adjacent pixels in original_pool using convolution operations (tf.nn.conv2d) with predefined kernels (left_kernel, right_kernel, up_kernel, down_kernel).
    - Similarly, computes spatial differences (d_enhanced_left, d_enhanced_right, d_enhanced_up, d_enhanced_down) between adjacent pixels in enhanced_pool.
- **Squared Differences**: Computes the squared differences (d_left, d_right, d_up, d_down) between d_original_* and d_enhanced_* tensors.

It returns the sum of squared differences (d_left + d_right + d_up + d_down) and measures how well the enhanced image preserves the spatial arrangement of illumination from the original image.

Zero-DCE Model Class

- **Constructor(__init__ method)**: Inherits from keras.Model and initializes the dce_model attribute by calling build_image_enhancement_net(), which constructs the Deep Curve Estimation (DCE) network for image enhancement.

- **Compilation (compile method)**:
  - Overrides the compile method of keras.Model.
  - Configures the optimizer as Adam with a specified learning_rate.
  - Initializes spatial_constancy_loss as an instance of SpatialConsistencyLoss with reduction="none".

- **Image Enhancement (get_enhanced_image method)**: Enhances the input data using the outputs (output) from the DCE network.
  - Splits output into eight components (r1 to r8).
  - Iteratively enhances data using these components based on a formula involving element-wise operations with the components.

- **Forward Pass (call) Method**:
  - Overrides the call method of keras.Model.
  - Executes the DCE network (dce_model) on the input data and enhances the image using get_enhanced_image.

- **Training Step (train_step method)**:
  - Overrides the train_step method of keras.Model.
  - Computes gradients of the total loss with respect to trainable weights of dce_model using tf.GradientTape.
  - Updates weights of dce_model using the optimizer based on computed gradients.
  - Returns losses computed during the step.

- **Testing (test_step method)**:
  - Overrides the test_step method of keras.Model.
  - Performs forward pass through dce_model to get outputs and computes losses using compute_losses.
  - Returns losses computed during testing.

Training the Zero-DCE Model

- **Model Initialization**: It creates an instance of the Zero-DCE model.
- **Compilation**:
  - zero_dce_model.compile(learning_rate=1e-4): Configures the model for training.
  - Uses Adam optimizer with a learning rate of 1e-4.
  - Initializes losses including spatial_constancy_loss defined in SpatialConsistencyLoss class.

- **Training**: It trains the model on train_dataset and evaluates on val_dataset for 70 epochs and returns a history object that contains training/validation loss values over epochs.

"perform_inference" Function

- Converts the PIL image to a numpy array (image_array).
- Normalizes pixel values to the range [0, 1].
- Expands the dimensions to match the model's input shape (adds a batch dimension).
- Passes the preprocessed image through the Zero-DCE model (zero_dce_model).
- Scales the output image back to the [0, 255] range and converts it to uint8.
- Converts the numpy array back to a PIL image (output_image).
- output_image: Enhanced output image as a PIL image.

Summary of Findings

In this project, we implemented the Zero-Reference Deep Curve Estimation (Zero-DCE) network for enhancing low-light images. The Zero-DCE model estimates light enhancement curves (LE-curves) to improve image quality iteratively. Key outcomes include:

- **Enhanced Image Quality**: The model effectively enhances brightness and contrast while maintaining natural colors and details.
- **Effective Loss Functions**: Utilized non-reference loss functions such as spatial consistency, exposure control, color constancy, and illumination smoothness to guide the enhancement process.
- **Quantitative Improvement**: The model's performance, measured by PSNR, showed significant improvement in image quality.

Methods to Further Improve the Project

1. **Enhanced Network Architecture**: Experiment with deeper networks and integrate attention mechanisms.
2. **Advanced Loss Functions**: Incorporate perceptual and adversarial loss functions for more visually pleasing and realistic enhancements.
3. **Data Augmentation**: Use larger, more diverse datasets and advanced augmentation techniques to improve model robustness.
4. **Real-time Enhancements**: Optimize for faster inference times and develop lightweight versions for mobile and embedded devices.
5. **User Interactivity**: Allow users to adjust enhancement parameters and gather feedback to refine the model.
6. **Cross-domain Applications**: Extend the model for video enhancement and multispectral imaging applications.

The link of the research paper I took help from is given
https://ar5iv.labs.arxiv.org/html/2001.06826

We have tested first 30 images and displayed the corresponding predicted and high light images along with their psnr value. **The average psnr value in my results is around 28.21 dB.**