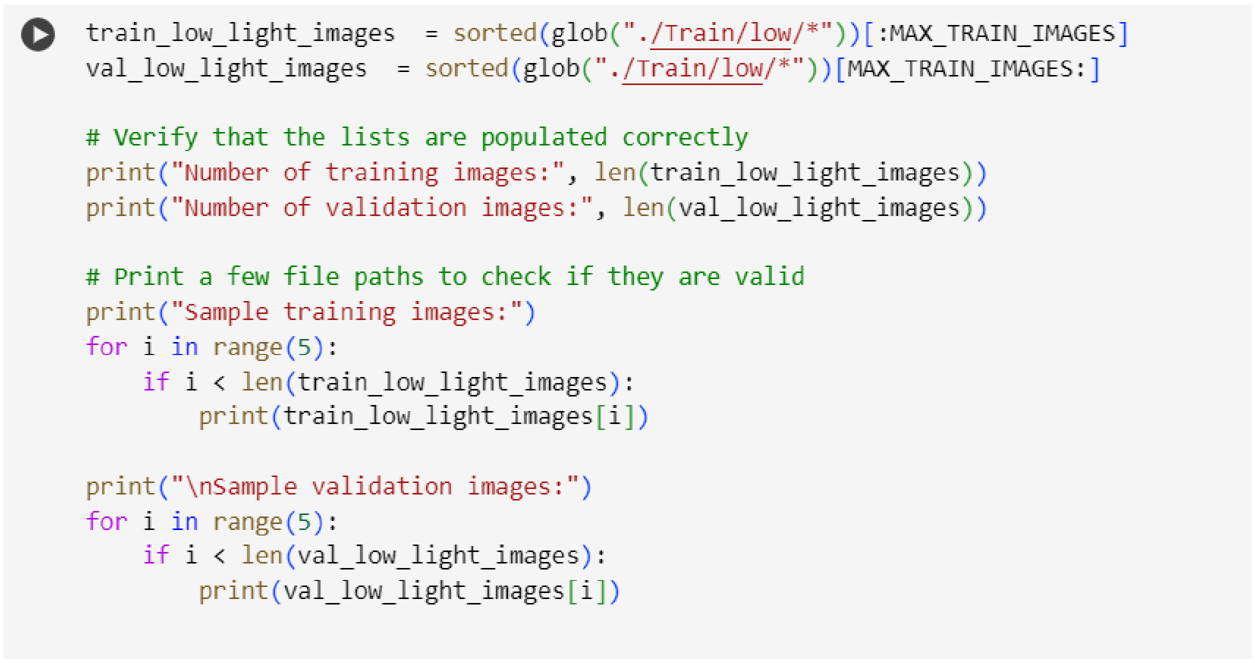
Name : - Priyanshu Joshi
Enrollment number : - 21113118

Image Denoising(Open Project VLG)

1. What is Image denoising ?

Image denoising is a fundamental task in the field of digital image processing and computer vision, aiming to enhance the visual quality of images by removing noise while preserving important features such as edges, textures, and fine details. Noise in images can arise from various sources, including electronic sensor noise, environmental conditions, and transmission errors, leading to visual artifacts that degrade image quality. Effective denoising is crucial for numerous applications, ranging from medical imaging, where clarity can significantly impact diagnosis, to consumer photography, enhancing the aesthetic quality of photos. Techniques for image denoising can be broadly categorized into spatial domain methods, transform domain methods, and learning-based methods. Spatial domain methods, like mean and median filtering, operate directly on the image pixels, smoothing out noise by averaging or using statistical measures, but they often blur fine details. Transform domain methods, such as wavelet transforms and Fourier transforms, work by transforming the image into a different domain, where noise can be more easily separated from the signal, allowing for more selective denoising. Recent advancements have increasingly focused on learning-based methods, particularly deep learning approaches, which leverage large datasets and neural networks to learn complex mappings from noisy to clean images. Convolutional neural networks (CNNs), autoencoders, and generative adversarial networks (GANs) have shown remarkable performance in this domain, achieving state-of-the-art results by effectively capturing spatial dependencies and learning hierarchical features. Each approach has its advantages and trade-offs, balancing the trade-off between noise removal and detail preservation. In practice, the choice of denoising technique often depends on the specific characteristics of the noise and the requirements of the application, necessitating a careful evaluation of performance metrics such as peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM). As image denoising continues to evolve, it remains a vibrant area of research with ongoing efforts to develop more robust, efficient, and generalizable methods, ultimately contributing to improved image quality across a wide range of applications.

2. Dataset Preparation

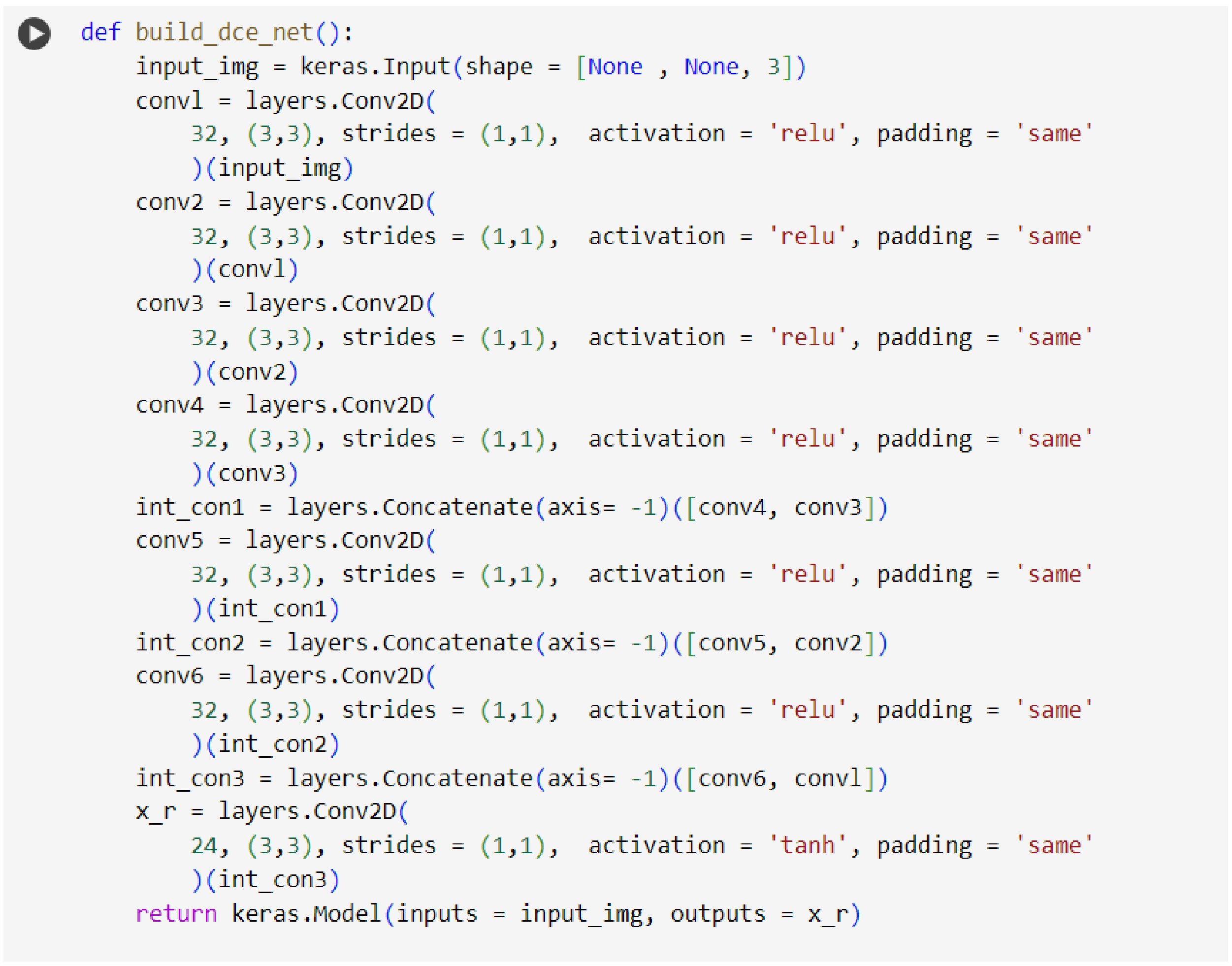


```
train_low_light_images  = sorted(glob("./Train/low/*"))[:MAX_TRAIN_IMAGES]
val_low_light_images  = sorted(glob("./Train/low/*"))[MAX_TRAIN_IMAGES:]

# Verify that the lists are populated correctly
print("Number of training images:", len(train_low_light_images))
print("Number of validation images:", len(val_low_light_images))

# Print a few file paths to check if they are valid
print("Sample training images:")
for i in range(5):
    if i < len(train_low_light_images):
        print(train_low_light_images[i])

print("\nSample validation images:")
for i in range(5):
    if i < len(val_low_light_images):
        print(val_low_light_images[i])
```

- Loading Image path
- Loading images
- Data generator
- The code imports the required libraries and specifies constants like IMAGE_SIZE, BATCH_SIZE, and MAX_TRAIN_IMAGES in order to prepare the dataset. It divides picture file paths into training and validation sets by loading and sorting them using glob. TensorFlow's tf.data is used to generate a data generator function.The dataset resizes, normalises, decodes, and reads the photos. To facilitate effective data feeding during model training, this method generates batch datasets. Consistent processing is ensured by this organised method, which allows the deep learning model to train efficiently on standardised input.

3. Model Building

- DCE network architecture
- TensorFlow and Keras are used in the model development process to define the DCE network design. ReLU activation is used in each layer of a deep convolutional network built by the build_dce_net() function. In order to preserve information, skip connections combine characteristics from many levels. Tanh activation is used by the output layer to create improvement maps. This architecture, together with unique loss functions and training logic, are integrated into the ZeroDCE class. Through the

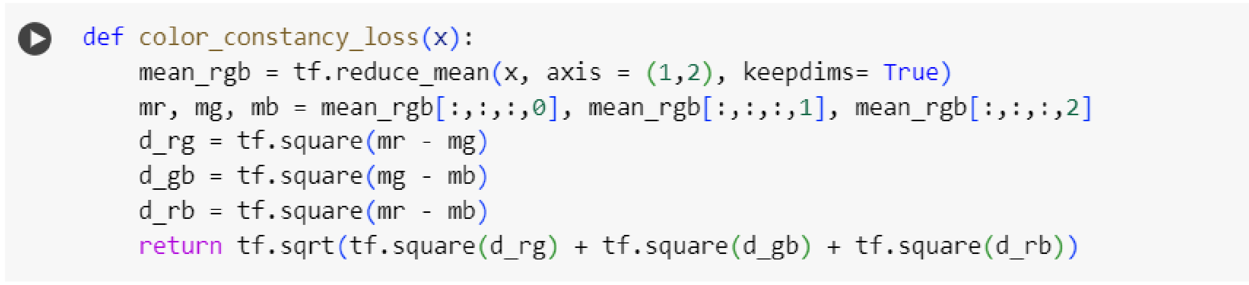


```
def build_dce_net():
    input_img = keras.Input(shape = [None , None, 3])
    conv1 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(input_img)
    conv2 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(conv1)
    conv3 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(conv2)
    conv4 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(conv3)
    int_con1 = layers.Concatenate(axis= -1)([conv4, conv3])
    conv5 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(int_con1)
    int_con2 = layers.Concatenate(axis= -1)([conv5, conv2])
    conv6 = layers.Conv2D(
        32, (3,3), strides = (1,1),  activation = 'relu', padding = 'same'
        )(int_con2)
    int_con3 = layers.Concatenate(axis= -1)([conv6, conv1])
    x_r = layers.Conv2D(
        24, (3,3), strides = (1,1),  activation = 'tanh', padding = 'same'
        )(int_con3)
    return keras.Model(inputs = input_img, outputs = x_r)
```

repeated application of learnt enhancement maps, it computes improved pictures, guaranteeing efficient denoising and contrast improvement.

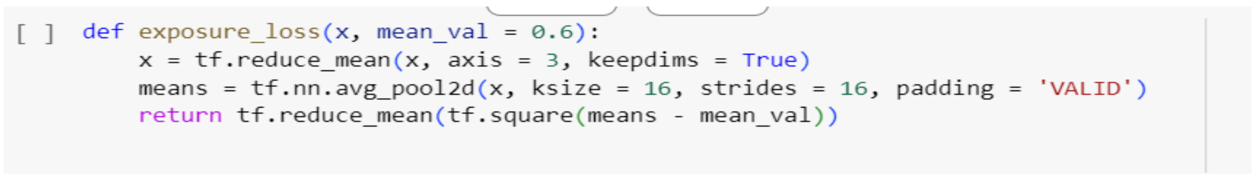
4. Loss Function

- Color Constancy Loss



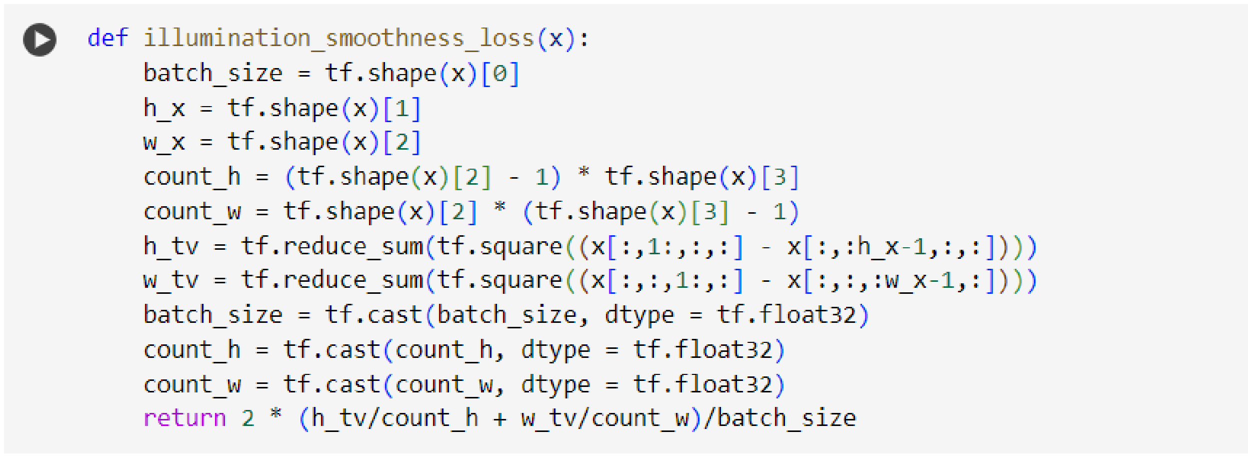
```
def color_constancy_loss(x):
    mean_rgb = tf.reduce_mean(x, axis = (1,2), keepdims= True)
    mr, mg, mb = mean_rgb[:,:,:,0], mean_rgb[:,:,:,1], mean_rgb[:,:,:,2]
    d_rg = tf.square(mr - mg)
    d_gb = tf.square(mg - mb)
    d_rb = tf.square(mr - mb)
    return tf.sqrt(tf.square(d_rg) + tf.square(d_gb) + tf.square(d_rb))
```

- Exposure loss



```
[ ]  def exposure_loss(x, mean_val = 0.6):
         x = tf.reduce_mean(x, axis = 3, keepdims = True)
         means = tf.nn.avg_pool2d(x, ksize = 16, strides = 16, padding = 'VALID')
         return tf.reduce_mean(tf.square(means - mean_val))
```

- Illumination Smoothness loss



```
def illumination_smoothness_loss(x):
    batch_size = tf.shape(x)[0]
    h_x = tf.shape(x)[1]
    w_x = tf.shape(x)[2]
    count_h = (tf.shape(x)[2] - 1) * tf.shape(x)[3]
    count_w = tf.shape(x)[2] * (tf.shape(x)[3] - 1)
    h_tv = tf.reduce_sum(tf.square((x[:,1:,:,:] - x[:,:h_x-1,:,:])))
    w_tv = tf.reduce_sum(tf.square((x[:,:,1:,:] - x[:,:,:w_x-1,:])))
    batch_size = tf.cast(batch_size, dtype = tf.float32)
    count_h = tf.cast(count_h, dtype = tf.float32)
    count_w = tf.cast(count_w, dtype = tf.float32)
    return 2 * (h_tv/count_h + w_tv/count_w)/batch_size
```

- Spatial Consistency Loss

Loss Function	Description	Purpose
Color Constancy Loss	Calculates the color consistency by measuring the differences in mean RGB values.	Ensures color balance and natural appearance.
Exposure Loss	Measures the deviation of the image's average brightness from a target mean value.	Maintains proper exposure and prevents under/overexposure.
Illumination Smoothness Loss	Computes the smoothness of the illumination map by summing the squared differences of adjacent pixels.	Promotes smooth and gradual lighting changes.
Spatial Consistency Loss	Uses convolutional kernels to compare local changes in mean values between original and enhanced images.	Preserves spatial coherence and structure of the image.

5. Model Defining

- **ZeroDCE Class**: The `ZeroDCE` class inherits from `keras.Model` and includes the DCE network as a sub-model. It also defines the training and test steps, loss computation, and image enhancement process.
- **Compile and Training**: the model is compiled with an Adam optimizer and trained for 100 epochs.

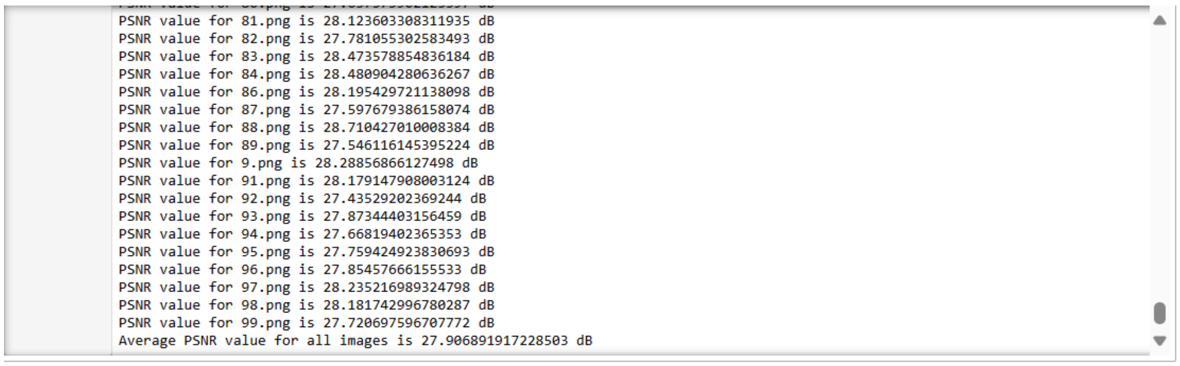
6. Visualization:

- **Plot:**
  - **Plotting Function Definition**: The `plot_result` function is defined to plot the training and validation losses for different loss components.
  - **Plotting Losses**: The function is called for different loss metrics (`total loss`, `illumination_smoothness_loss`, `spatial_constancy loss`, `color_constancy loss`, and `exposure loss`) to plot their values over epochs.
- **Visualization and Inference:**The visualization and inference part is designed to compare the original low-light images with the enhanced images produced by the trained Zero-DCE model. Here's the breakdown:
  - **Plotting Function Definition:** A `plot_result` function is defined to display original, auto-contrasted, and enhanced images side by side.
  - **Inference Function**: The `infer` function processes an input image through the Zero-DCE model to generate an enhanced image.
  - **Comparative Visualization**: For each test image, the original, auto-contrasted, and enhanced images are displayed.

7. PSNR Value:

- **Libraries Import:** The script imports necessary libraries (`cv2`, `numpy`, `glob`, and `os`).
- **PSNR Calculation Function**: A function `calculate_psnr` is defined to compute PSNR between two images. It calculates the Mean Squared Error (MSE) between the images and then computes the PSNR using the appropriate formula , where `max_pixel` is 255.
- **Directories Setup**: The script defines directories for original and low-light images and retrieves sorted lists of image file paths from these directories.
- **Assertions**: It asserts that the number of images in both directories is the same, ensuring one-to-one correspondence for PSNR calculation.

- **PSNR Calculation Loop**: The script iterates over the image pairs, loads each pair using `cv2.imread`, checks that the dimensions match, and calculates the PSNR value. Each PSNR value is stored in a list and printed along with the corresponding image file name.
- **Average PSNR Calculation**: Finally, the script calculates and prints the average PSNR value across all image pairs.
- The script successfully calculated the PSNR values for the image pairs in the specified directories. The mean PSNR value obtained from the comparisons is 27.906891917228503 dB, indicating a moderate level of quality in the low-light images relative to the original reference images.



```
PSNR value for 81.png is 28.123603308311935 dB
PSNR value for 82.png is 27.781055302583493 dB
PSNR value for 83.png is 28.473578854836184 dB
PSNR value for 84.png is 28.480904280636267 dB
PSNR value for 86.png is 28.195429721138098 dB
PSNR value for 87.png is 27.597679386158074 dB
PSNR value for 88.png is 28.710427010008384 dB
PSNR value for 89.png is 27.546116145395224 dB
PSNR value for 9.png is 28.28856866127498 dB
PSNR value for 91.png is 28.179147908003124 dB
PSNR value for 92.png is 27.43529202369244 dB
PSNR value for 93.png is 27.87344403156459 dB
PSNR value for 94.png is 27.66819402365353 dB
PSNR value for 95.png is 27.759424923830693 dB
PSNR value for 96.png is 27.85457666155533 dB
PSNR value for 97.png is 28.235216989324798 dB
PSNR value for 98.png is 28.181742996780287 dB
PSNR value for 99.png is 27.720697596707772 dB
Average PSNR value for all images is 27.906891917228503 dB
```