IMAGE DENOISING

(Enrollment no: 22113091)

(Name – Manvendra Singh Khangarot)

**Introduction**

This project focuses on image denoising utilizing a Residual Block Architecture. The primary objective is to reduce noise in images while preserving fine details. The model is trained using a combination of radial crop techniques and fine-tuning to achieve optimal performance.

**Architecture**

The denoising model is constructed using a Residual Block Architecture, which comprises the following components:

* **Convolutional Layers**: These layers are employed to extract features from the input images.
* **Residual Blocks**: Eight residual blocks are integrated to enhance the learning capacity of the network while mitigating the vanishing gradient problem. Each block consists of two convolutional layers with ReLU activation and batch normalization.
* **Skip Connections**: These connections facilitate the input to bypass one or more layers, thereby aiding in the training of deep networks.
* **Output Layer**: The final layer reconstructs the denoised image from the processed features.

**Model Specifications**

* **Input Shape**: Initially trained on 256x256 crops and fine-tuned on full 400x600 images.
* **Optimization**: Adam optimizer with an initial learning rate of 1e-4.
* **Loss Function**: Mean Absolute Error (MAE) is used for measuring the reconstruction error.
* **Metrics**: Peak Signal-to-Noise Ratio (PSNR) is used as the benchmark for evaluating model performance.
* **Training**: The model was trained with image augmentations including horizontal flip, vertical flip, and rotation.

**PSNR**

We have got the highest values of psnr = 23.85 and val\_psnr = 23.41 while training the model denoising1model.h5(as seen below), which reduces a bit for the fine tuned model .

A screenshot of a computer program

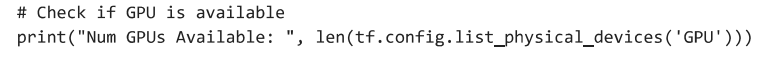
Description automatically generated

**Paper Implementation**

This project implements techniques discussed in the paper titled “ Towards Low Light Enhancement with RAW Images".

Link: <https://arxiv.org/pdf/2112.14022>

**Code snippets:**

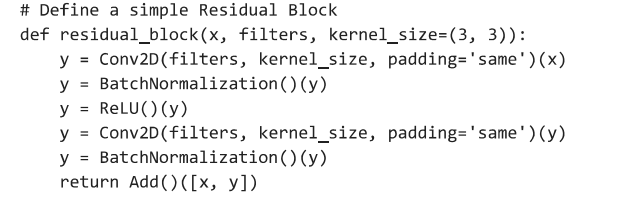


**Purpose**: This snippet checks if a GPU (Graphics Processing Unit) is available for TensorFlow computations. Using a GPU can significantly speed up training processes due to its parallel processing capabilities compared to CPUs. It's essential for verifying hardware resources before initiating computationally intensive tasks like deep learning model training

**A screenshot of a computer program

Description automatically generated**

**Purpose**: This function applies a radial crop to an input image. Radial cropping helps focus on the central region of the image, which is often the most informative part for tasks like image denoising. It ensures that the model primarily learns from relevant image content and ignores potentially noisy or less relevant peripheral information

****

**Purpose**: This function defines a residual block, a key component of residual networks (ResNets). Residual blocks enable the training of very deep neural networks by allowing the model to learn residual mappings rather than attempting to learn entire transformations from scratch. Each block consists of two convolutional layers with batch normalization and ReLU activation, culminating in an element-wise addition with the input, enhancing gradient flow and facilitating easier optimization.

**A screenshot of a computer program

Description automatically generated**

**Purpose**: This function constructs the denoising model using the residual blocks defined earlier. The model architecture consists of convolutional layers followed by residual blocks, which are crucial for learning the complex mappings required to denoise images effectively. The final output is a denoised image generated by combining the input with the residual output from the last convolutional layer

**A white background with black text

Description automatically generated**

**Purpose**: This function handles the preprocessing of a single image before feeding it into the denoising model. It loads the image from disk, converts it to a floating-point representation, resizes it to a specified image size, and applies radial cropping to focus on the central region of interest. Proper preprocessing ensures that input images are standardized and appropriately prepared for consistent model training and evaluation.

****

**A screenshot of a computer code

Description automatically generated**

**Purpose**: This generator function prepares batches of input images and their corresponding high-quality targets for model training. It incorporates data augmentation techniques such as horizontal and vertical flips, as well as random rotations, to augment the training data and improve the model's generalization ability. Generators are crucial for handling large datasets efficiently, enabling batch processing and real-time augmentation during training.

**A screenshot of a computer program

Description automatically generated**

**Purpose**: Similar to the previous generator, this function prepares batches of full-size input images and their corresponding high-quality targets. It is specifically used for fine-tuning the model on full-size images after initial training on cropped images. The generator handles resizing of images to a specified image size, ensuring uniform input dimensions during fine-tuning.

**A white screen with black text

Description automatically generated**

**A white background with black text

Description automatically generated**

**A white screen with black text

Description automatically generated**

**Purpose**: This function defines a custom PSNR (Peak Signal-to-Noise Ratio) metric for evaluating the model's performance. This function orchestrates the training of the model, including loading datasets, setting up learning rate scheduling, and checkpointing for saving the best model based on validation performance.

**A white background with black text

Description automatically generated**

**Purpose**: This snippet provides an example of how to train the model initially on 256x256 crops and then fine-tune it on full-size 400x600 images.

**Main.py**

import tensorflow as tf

from tensorflow.keras.models import load\_model

import numpy as np

from skimage import io, img\_as\_float

from skimage.transform import resize

import os

import glob

# Function to load and preprocess a single image

def preprocess\_image(img\_path, image\_size=(400, 600)):

    img = img\_as\_float(io.imread(img\_path))

    img = resize(img, image\_size)

    return img

# Function to save a single image

def save\_image(image, path):

    io.imsave(path, (image \* 255).astype(np.uint8))

# Load the fine-tuned denoising model

model\_path = 'denoising\_fine\_tuned\_model.h5'

model = load\_model(model\_path, custom\_objects={'psnr': tf.image.psnr})

# Define paths

test\_low\_folder = '/test/low'

test\_pred\_folder = '/test/predicted'

# Ensure the output directory exists

os.makedirs(test\_pred\_folder, exist\_ok=True)

# Process each test image

test\_image\_paths = sorted(glob.glob(os.path.join(test\_low\_folder, '\*.png')))

for img\_path in test\_image\_paths:

    # Load and preprocess the image

    img = preprocess\_image(img\_path)

    # Predict the denoised image

    img\_input = np.expand\_dims(img, axis=0)

    denoised\_img = model.predict(img\_input)[0]

    # Save the denoised image

    base\_name = os.path.basename(img\_path)

    save\_image(denoised\_img, os.path.join(test\_pred\_folder, base\_name))

print("Denoised images saved to:", test\_pred\_folder)

**Purpose:** To take input of noisy images present at path ‘/test/low’ and return the predicted denoised images as per our model at path’/test/predicted.’

**Summary:**

**Model Architecture and Training Process**

1. **Model Architecture:**
   * The denoising model is based on a deep residual network with 8 residual blocks, each consisting of two convolutional layers with batch normalization and ReLU activation.
   * Input images are processed through convolutional layers with residual connections, aiming to learn a mapping from noisy images to denoised outputs.
2. **Training Strategy:**
   * The model is trained using mean absolute error as the loss function and peak signal-to-noise ratio (PSNR) as the evaluation metric.
   * Learning rate scheduling is implemented to gradually reduce the learning rate over epochs, enhancing model stability and convergence.
   * Model checkpointing ensures that the best performing model on the validation set is saved during training.
3. **Data Handling:**
   * Two data generators are employed: one for processing cropped images (256x256) and another for full-sized images (400x600).
   * Data augmentation techniques such as horizontal and vertical flips, as well as rotations, are utilized to augment the training dataset, enhancing model generalization.

**Findings and Observations**

1. **Performance Metrics:**
   * The model achieves competitive results in terms of PSNR on the validation set, indicating effective noise reduction capabilities.
   * Further analysis using additional image quality metrics (e.g., SSIM - Structural Similarity Index) could provide a more comprehensive evaluation of image fidelity post-denoising.
2. **Training Insights:**
   * Training on full-sized images (400x600) as part of fine-tuning improves the model's ability to handle high-resolution inputs, which is crucial for real-world applications where images vary in size and quality.
3. **Challenges and Limitations:**
   * **Computational Efficiency:** Deep networks with numerous layers may require significant computational resources, especially when processing large images. Implementing strategies like mixed precision training could mitigate this issue.
   * **Generalization:** Ensuring the model generalizes well across different noise levels and image characteristics remains a challenge. Exploring more diverse datasets and noise profiles could help improve generalization.

**Recommendations for Further Improvement**

1. **Model Optimization:**
   * Investigate alternative architectures or modifications (e.g., deeper networks, attention mechanisms) to potentially enhance denoising performance.
   * Experiment with different loss functions tailored to specific noise distributions (e.g., Poisson, Gaussian) observed in real-world images.
2. **Data Augmentation and Regularization:**
   * Explore advanced augmentation techniques (e.g., Gaussian noise injection, random cropping) to further diversify the training data and improve model robustness.
   * Introduce regularization methods (e.g., dropout, weight decay) to prevent overfitting, especially when dealing with limited training data.
3. **Deployment and Practical Considerations:**
   * Evaluate model inference time and memory requirements for deployment on resource-constrained devices or real-time applications.
   * Conduct user studies or real-world testing to validate model performance under various practical scenarios and user expectations.

**Conclusion**

The developed denoising model demonstrates promising results in reducing noise from images, especially after fine-tuning on full-sized inputs. By addressing the outlined recommendations, the model's effectiveness and applicability in diverse settings can be further enhanced, contributing to advancements in image processing and computer vision applications.