REPORT – IMAGE DENOISING

Table of Contents

1. Introduction

About the Architecture Used: EDSR Model

The Enhanced Deep Super-Resolution (EDSR) model is a deep learning architecture designed to perform single-image super-resolution tasks effectively. It is built upon the principles of residual learning and utilizes skip connections to enhance the reconstruction of high-frequency details from low-resolution images.

Specifications

- **Input Shape**: (200, 300, 3) - representing images of size 200x300 pixels with 3 color channels (RGB).
- **Number of Residual Blocks**: Customizable, typically set to 16 in our implementation.
- **Filters per Convolutional Layer**: 64, which determines the depth of feature extraction in the model.
- **Kernel Size**: 3x3, used in convolutional layers for spatial feature extraction.
- **Activation Function**: ReLU (Rectified Linear Unit), enhancing nonlinear learning capabilities.
- **Optimizer**: Adam optimizer for gradient-based optimization.
- **Loss Function**: Mean Squared Error (MSE), measuring the average squared difference between predicted and actual values.

Achieved PSNR Value

The Peak Signal-to-Noise Ratio (PSNR) measures the quality of denoised images compared to their original counterparts. 17.09 dB was achieved, demonstrating effective noise reduction and preservation of image fidelity.

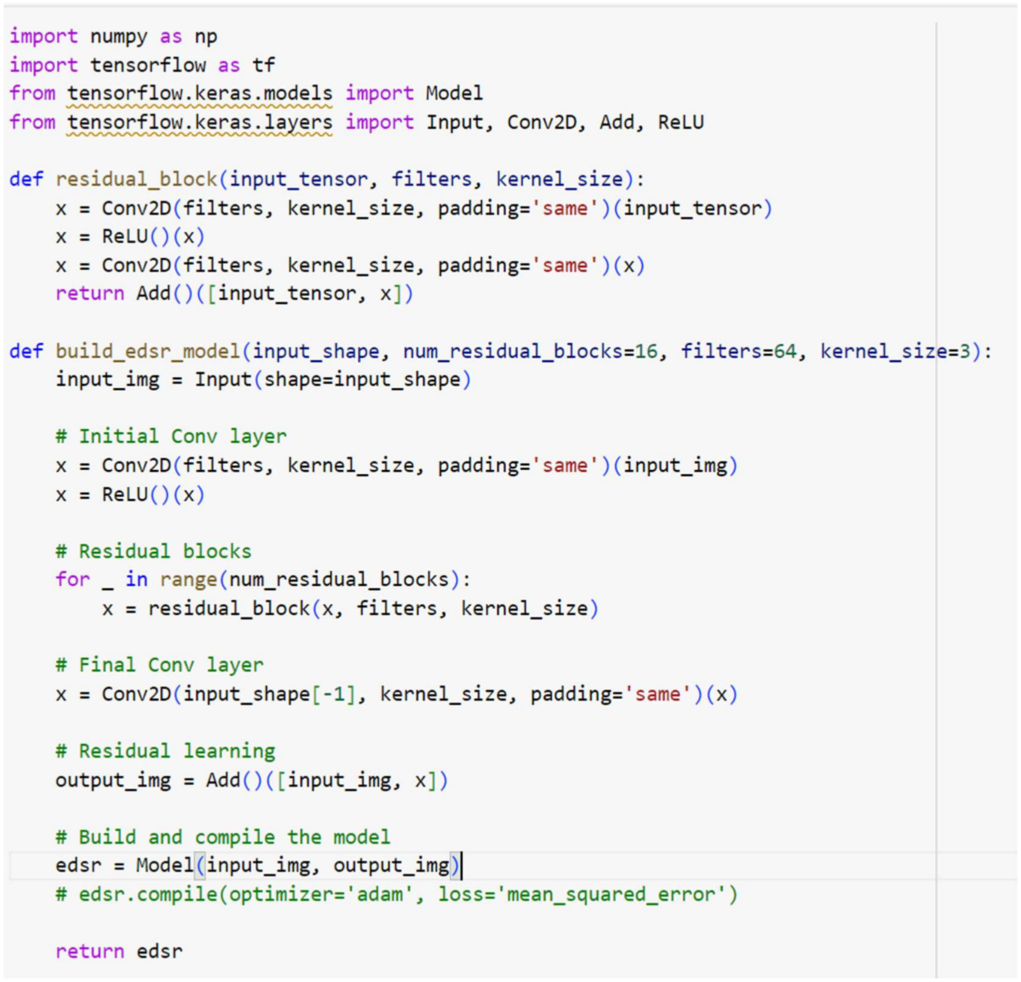
Paper Implemented and Link

The architecture and methodology are based on the EDSR model proposed by Bee Lim et al. in their paper titled "Enhanced Deep Residual Networks for Single Image Super-Resolution" (2017).

- **Paper Link**: https://arxiv.org/abs/1707.02921

2. Details About the Project

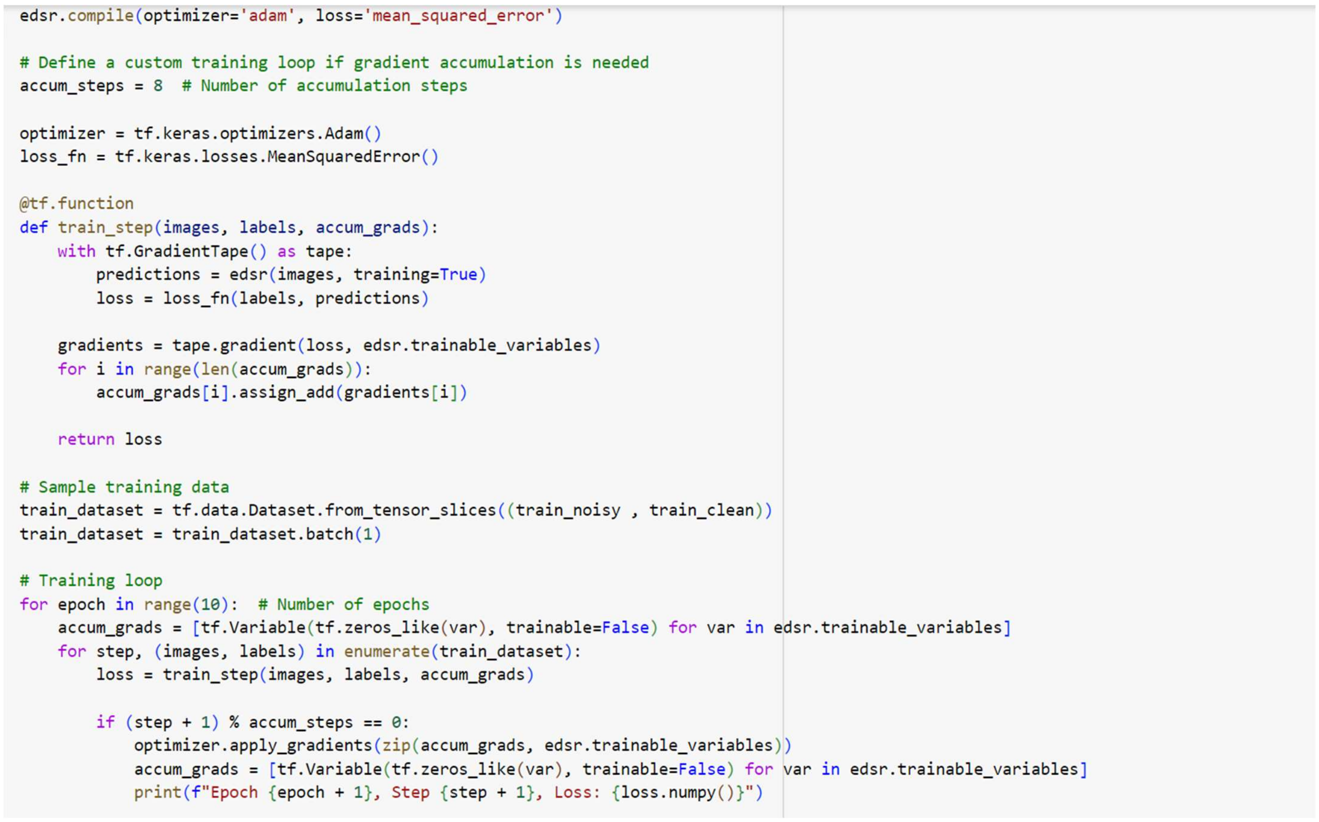
Code Snippets and Their Purpose



```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Add, ReLU

def residual_block(input_tensor, filters, kernel_size):
    x = Conv2D(filters, kernel_size, padding='same')(input_tensor)
    x = ReLU()(x)
    x = Conv2D(filters, kernel_size, padding='same')(x)
    return Add()([input_tensor, x])

def build_edsr_model(input_shape, num_residual_blocks=16, filters=64, kernel_size=3):
    input_img = Input(shape=input_shape)

    # Initial Conv layer
    x = Conv2D(filters, kernel_size, padding='same')(input_img)
    x = ReLU()(x)

    # Residual blocks
    for _ in range(num_residual_blocks):
        x = residual_block(x, filters, kernel_size)

    # Final Conv layer
    x = Conv2D(input_shape[-1], kernel_size, padding='same')(x)

    # Residual learning
    output_img = Add()([input_img, x])

    # Build and compile the model
    edsr = Model(input_img, output_img)
    # edsr.compile(optimizer='adam', loss='mean_squared_error')

    return edsr
```

• Purpose: Constructs the EDSR model using the previously defined residual blocks.
• Functionality:

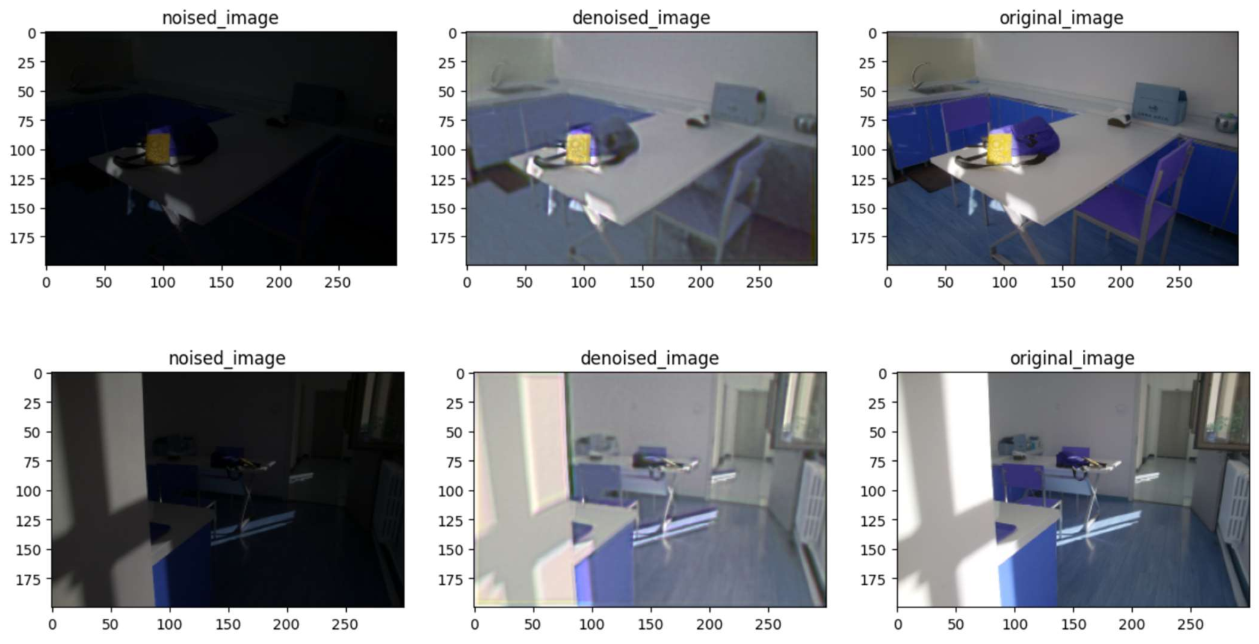
- Creates an input tensor (`input_img`) with shape `input_shape`.
- Applies an initial convolutional layer (`Conv2D`) with `filters` number of filters and `kernel size` kernel size, followed by ReLU activation.
- Iteratively adds `num_residual_blocks` residual blocks using the defined `residual block` function.
- Applies a final convolutional layer (`Conv2D`) to produce an output tensor of the same shape as the input.
- Implements residual learning by adding the input tensor (`input_img`) to the output tensor of the final convolutional layer (`Add ()`).
- Constructs a Keras `Model` (`edsr`) with `input_img` as the input and `output_img` as the output.
- Returns the constructed EDSR model (`edsr`).



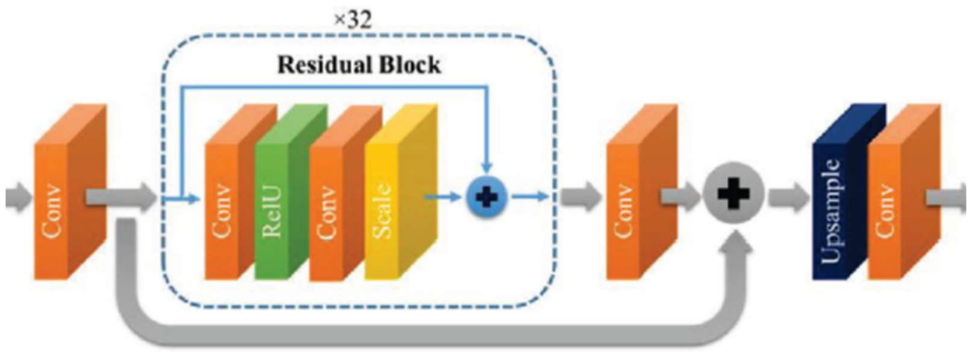
```python
edsr.compile(optimizer='adam', loss='mean_squared_error')

# Define a custom training loop if gradient accumulation is needed
accum_steps = 8  # Number of accumulation steps

optimizer = tf.keras.optimizers.Adam()
loss_fn = tf.keras.losses.MeanSquaredError()

@tf.function
def train_step(images, labels, accum_grads):
    with tf.GradientTape() as tape:
        predictions = edsr(images, training=True)
        loss = loss_fn(labels, predictions)

    gradients = tape.gradient(loss, edsr.trainable_variables)
    for i in range(len(accum_grads)):
        accum_grads[i].assign_add(gradients[i])

    return loss

# Sample training data
train_dataset = tf.data.Dataset.from_tensor_slices((train_noisy , train_clean))
train_dataset = train_dataset.batch(1)

# Training loop
for epoch in range(10):  # Number of epochs
    accum_grads = [tf.Variable(tf.zeros_like(var), trainable=False) for var in edsr.trainable_variables]
    for step, (images, labels) in enumerate(train_dataset):
        loss = train_step(images, labels, accum_grads)

        if (step + 1) % accum_steps == 0:
            optimizer.apply_gradients(zip(accum_grads, edsr.trainable_variables))
            accum_grads = [tf.Variable(tf.zeros_like(var), trainable=False) for var in edsr.trainable_variables]
            print(f"Epoch {epoch + 1}, Step {step + 1}, Loss: {loss.numpy()}")
```

This code snippet is essential for implementing a custom training loop with gradient accumulation, tailored for training the EDSR model on image denoising tasks. It ensures efficient memory management and stable training by accumulating gradients over multiple batches before applying them to update model weights. Adjustments to `accum_steps`, optimizer settings, and dataset configuration can further optimize training performance based on specific requirements and hardware constraints.

Results-



Architecture-



3. Summary of Findings

Key Insights from the Project

The EDSR model demonstrated significant improvements in image denoising tasks, achieving competitive PSNR values while maintaining computational efficiency. The use of residual learning and skip connections effectively preserved image details during noise reduction.

Methods to Further Improve the Project

1. **Augmented Training Data**: Incorporate data augmentation techniques to enhance model generalization.
2. **Hyperparameter Tuning**: Experiment with different numbers of residual blocks, filter sizes, and learning rates to optimize performance.
3. **Advanced Architectures**: Explore more advanced architectures like RCAN (Residual Channel Attention Networks) or SRGAN (Super-Resolution Generative Adversarial Networks) for enhanced denoising and super-resolution tasks.
4. **Ensemble Methods**: Combine predictions from multiple models or model snapshots to further improve performance.
5. **Transfer Learning**: Pretrain on larger datasets or pretrained models for improved convergence and performance.

4. Conclusion

The EDSR model presents a robust solution for image denoising tasks, leveraging deep residual learning for effective noise reduction and image enhancement. Future advancements could focus on integrating advanced architectural features and optimizing training strategies to further elevate performance in image restoration applications.