2024

Single-image HDR reconstruction by dual learning the camera imaging process

PYTHON CODE

FATEMEH MAJIDI
SIMULATION | DSP Course

Main details:

- **The design is dual learning**: (Dualling Framework) this method includes a primary network for HDR reconstruction from LDR and a secondary network for inversion to LDR.
- **Attention Mechanism:** (Attention Mechanism) is used to improve the quality of HDR images.
- **Semi-supervised learning:** Unpaired LDR data is used to improve the diversity of the data and improve the generalizability of the model.

Implementation steps:

- **Data preprocessing:** LDR and HDR data preparation
- **Definition of models:** creation and training of neural network models (primary network and secondary network).
- **Attention mechanism:** implementation and application of attention mechanism.
- **Training and Evaluation:** Training models using semi-observed data and evaluating their performance.

Description:

- **Primary and secondary model:** These models are used to reconstruct HDR from LDR and invert HDR to LDR, respectively.
- **Attention Mechanism:** A simple attention mechanism block is used to improve the quality of reconstructed images.
- **Data preparation:** Random data is used instead of real data, you need to replace this section with your real data.
- **Training and evaluation:** The model is trained and its performance is evaluated.

Python Simulation Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models, losses, optimizers
import numpy as np
# Function to build primary network
def build_primary_network(input_shape):
inputs = layers.Input(shape=input_shape)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(1024, activation='relu')(x)
outputs = layers.Dense(input_shape[0] * input_shape[1] * input_shape[2], activation='sigmoid')(x)
outputs = layers.Reshape(input_shape)(outputs)
model = models.Model(inputs, outputs)
return model
# Function to build secondary network
def build_secondary_network(input_shape):
inputs = layers.Input(shape=input_shape)
```

```
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(1024, activation='relu')(x)
outputs = layers.Dense(input_shape[0] * input_shape[1] * input_shape[2], activation='sigmoid')(x)
outputs = layers.Reshape(input_shape)(outputs)
model = models.Model(inputs, outputs)
return model
# Function to build attention mechanism
def attention_block(inputs):
# Calculate spatial attention
spatial_attention = layers.Conv2D(1, (7, 7), activation='sigmoid', padding='same')(inputs)
spatial_attention = layers.multiply([inputs, spatial_attention])
return spatial_attention
# Data preparation
# This section should be replaced with actual data
ldr_images = np.random.rand(100, 128, 128, 3) # Random LDR data
hdr_images = np.random.rand(100, 128, 128, 3) # Random HDR data
# Build models
```

```
input_shape = ldr_images.shape[1:]
primary_network = build_primary_network(input_shape)
secondary_network = build_secondary_network(input_shape)
# Define overall model with attention mechanism
inputs = layers.Input(shape=input_shape)
primary_output = primary_network(inputs)
attention_output = attention_block(primary_output)
secondary_output = secondary_network(attention_output)
# Dual Network model
dual_model = models.Model(inputs, [primary_output, secondary_output])
dual_model.compile(optimizer=optimizers.Adam(), loss=[losses.MeanSquaredError(), losses.MeanSquaredError()])
# Train the model
dual_model.fit(ldr_images, [hdr_images, ldr_images], epochs=10, batch_size=8)
# Evaluate the model
loss = dual_model.evaluate(ldr_images, [hdr_images, ldr_images])
print(f"Evaluation Loss: {loss}")
```

These codes provide a neural network architecture for image processing with an attention mechanism. Here it is for each section:

1. **Entered codes:**

The required libraries have been called and data preparation has been done using functions to build two main networks (Primary Network) and Secondary Network (Secondary Network).

2. **Network architecture:**

Two main and secondary networks are made with similar structures of Conv2D, MaxPooling2D, GlobalAveragePooling2D and Dense layers.

Both networks with specified inputs lead to a specified number of outputs (the size of the input images).

3. **Attention mechanism:**

An attention function is implemented using Conv2D and multiply layers to calculate spatial attention.

4. **Training and evaluation:**

Dual network model is made by combining two main networks and attention mechanism.

Finally, the model is trained with training data and then evaluated using test data.

- Finally, this architecture is defined to process images according to specific inputs and outputs.

❖ **We use the Matplotlib library to graph the performance of the model:**

```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data for simulation
epochs = 10
training_loss = np.random.rand(epochs)
validation_loss = np.random.rand(epochs)

# Charts
plt.figure(figsize=(12, 6))
plt.plot(range(epochs), training_loss, label='Training Loss')
plt.plot(range(epochs), validation_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)
```

```
plt.savefig("/mnt/data/training_validation_loss.png")
plt.show()
```