Y-Net Autoencoder for Color Image Compression and Reconstruction

Overview

This program implements a Y-Net-based autoencoder architecture designed for color image compression and reconstruction. The network comprises two branches (left and right) and employs convolutional, pooling, upsampling, and dense layers to process and encode RGB images. The code includes loading, preprocessing, training, and saving the model for future use.

Table of Contents

---

1. Loading and Preprocessing the Input Image

Code

```
# Load the color image (replace 'path/to/your/image.jpg' with your image path)
image = cv2.imread('kodim07.jpg')

# Display the original color image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()

# Resize the image to 256x256
resized_image = cv2.resize(image, (256, 256), interpolation=cv2.INTER_AREA)
```

```
# Save the resized image
cv2.imwrite('kodim07_256_Colour.jpg', resized_image)

# Normalize and prepare the image for the neural network
image1 = resized_image / 255.0
image1 = image1.astype(np.float32).reshape((1, 256, 256, 3))
```

Output

- The image is resized to 256x256 and normalized to the range [0, 1].

---

2. Network Architecture

Left Branch

The left branch uses 3 convolutional layers with MaxPooling2D to downsample the input features.

```
left_inputs = Input(shape=input_shape)
x = left_inputs
for i in range(3):
    x = Conv2D(filters=filters, kernel_size=kernel_size, padding='same', activation='relu')(x)
    x = Dropout(dropout)(x)
    x = MaxPooling2D()(x)
    filters *= 2
```

Right Branch

The right branch mirrors the left but incorporates dilation in convolutional layers.

```
right_inputs = Input(shape=input_shape)
y = right_inputs
for i in range(3):
    y = Conv2D(filters=filters, kernel_size=kernel_size, padding='same', activation='relu',
dilation_rate=2)(y)
    y = Dropout(dropout)(y)
    y = MaxPooling2D()(y)
    filters *= 2
```

Merging and Reconstruction

The outputs of the two branches are concatenated and upsampled to reconstruct the image.

```
# Merge left and right branches
merged = concatenate([x, y])

# Upsample and reconstruct
for i in range(4):
    merged = UpSampling2D()(merged)
    merged = Conv2D(filters=filters // 2, kernel_size=kernel_size, padding='same',
activation='relu')(merged)
    filters //= 2

output_img = Conv2D(3, kernel_size=(3, 3), padding='same', activation='sigmoid')(merged)
```

---

3. Training the Autoencoder

Code

```
# Compile and train the model
model.compile(optimizer='adam', loss='mse')
model.fit([image_rgb, image_rgb], ground_truth, epochs=10, batch_size=1)
```

Description

- **Optimizer**: Adam
- **Loss Function**: Mean Squared Error (MSE)
- **Epochs**: 10
- **Batch Size**: 1

---

4. Model Saving and Visualization

Save the Model

```
model.save('YnetAutoencoder_model_rgb.h5')
```

Visualize the Input and Reconstructed Images

```
# Display input image
plt.imshow(image_rgb)
plt.title('Input Image')
plt.show()

# Display reconstructed image
plt.imshow(reconstructed_img)
plt.title('Reconstructed Image')
plt.show()
```

Save the Reconstructed Image

```
cv2.imwrite('kodim07reconstructed_image.jpg', cv2.cvtColor(reconstructed_img,
cv2.COLOR_RGB2BGR))
```

Results

- The reconstructed image is visually comparable to the input, demonstrating the effectiveness of the autoencoder.

Notes

1. Ensure the input image paths are correctly specified.
2. Training on a larger dataset will improve reconstruction quality.
3. Hyperparameters like `encoded_dim`, `dropout`, and `filters` can be adjusted for optimization.

References

- TensorFlow/Keras Documentation
- OpenCV Library
- Scikit-Image Library