Image Colorization Autoencoder

This Python script implements an image colorization autoencoder using TensorFlow and Keras. The autoencoder is trained to colorize grayscale images.

Requirements

- Python
- NumPy
- Matplotlib
- Pillow (PIL)
- TensorFlow

Code Overview

Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Reshape,
Flatten, Input, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.optimizers import Adam
```

Display Image Function

Displays an image using Matplotlib.

```
# Display image function
def display_image(image_data, title="Image"):
    plt.imshow(image_data)
    plt.title(title)
    plt.axis('off')
    plt.show()
```

Load Images from Directory

Loads images from a specified directory, resizes them, and converts them to RGB format if they are grayscale.

```
# Load images from directory
def load_images(directory, image_size=(128, 128)):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            img = Image.open(os.path.join(directory, filename))
            img = img.resize(image_size)  # Resize to desired pixels
            img = np.array(img)
            if len(img.shape) == 2:
                img = np.expand_dims(img, axis=-1)  # Add channel
dimension if grayscale
                img = np.repeat(img, 3, axis=-1)  # Convert grayscale
to RGB
            images.append(img)
    return np.array(images)
```

Define the Network Architecture

Defines an autoencoder network using Keras.

```
# Define the network architecture using Keras
input_shape = (128, 128, 3)  # Input size for RGB images

input_img = Input(shape=input_shape)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = Conv2D(128, (3, 3), activation='relu', padding='same', strides=(2,
2))(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', strides=(2,
2))(x)
x = Flatten()(x)
encoded = Reshape((32, 32, 256))(x)

x = Conv2DTranspose(256, (3, 3), activation='relu', padding='same',
strides=(2, 2))(encoded)
x = Conv2DTranspose(128, (3, 3), activation='relu', padding='same',
strides=(2, 2))(x)
x = Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())
```

Train Network Function

Trains the autoencoder on the given training data.

```
# Training function
def train_network(train_data, epochs, batch_size):
    autoencoder.fit(train_data, train_data, epochs=epochs,
batch_size=batch_size, shuffle=True)
```

Visualize Results Function

Visualizes the original and colorized images for comparison.

```
# Visualize results
def visualize_results(train_data):
    for img in train_data[:5]:
        img = img.reshape(128, 128, 3)
        colorized_output = autoencoder.predict(img.reshape(1, 128, 128,
3))
        colorized_output = (colorized_output[0] * 255).astype(np.uint8)

        plt.figure(figsize=(12, 4))
        plt.subplot(1, 3, 1)
        display_image(img, title="Original")
        plt.subplot(1, 3, 2)
        display_image(colorized_output, title="Colorized Output")
        plt.subplot(1, 3, 3)
        display_image(img, title="Original")  # Show original again for
comparison
        plt.show()
```

Save Model Function

Saves the trained model weights to a file.

```
# Save the model
def save_model():
    autoencoder.save_weights("autoencoder_weights.h5")
```

Load Model Function

Loads the model weights from a file.

```
# Load the model
def load_model():
    autoencoder.load_weights("autoencoder_weights.h5")
```

Evaluate Network Function

Evaluates the autoencoder on the test data and prints the average loss.

```
# Evaluate the network on test data
def evaluate_network(test_data):
    total_loss = 0
    for img in test_data:
        img = img.reshape(1, 128, 128, 3)
        output_data = autoencoder.predict(img)
        total_loss += np.mean((output_data - img) ** 2)

    avg_loss = total_loss / len(test_data)
    print(f"Test Loss: {avg_loss}")
```

Main Function

Demonstrates the usage of the above functions.

```
if __name__ == "__main__":
    # Example usage

    # Load images from a directory
    train_data = load_images(r"/content/pics")

    # Normalize data
    train_data = train_data.astype(np.float32) / 255.0

    # Train the network
    train_network(train_data, epochs=50, batch_size=16)

    # Visualize some results
    visualize_results(train_data)

    # Save the model
    save_model()

    # Load the model (for testing purposes)
    load_model()

    # Evaluate on test data (using training data for simplicity)
    evaluate_network(train_data)
```

Usage

1. **Load Images**: Ensure your images are in the specified directory (`/content/pics`).
2. **Run the Script**: Execute the script to train the autoencoder, visualize results, save the model, and evaluate it.

python colorization_autoencoder.py

3. **Pre-trained Model**: If you have a pre-trained model, place `autoencoder_weights.h5` in the same directory and load it using the `load_model` function.

Notes

- Ensure you have sufficient training data for effective learning.
- The colorization task may require more sophisticated models and larger datasets for better performance.
- Adjust the number of epochs and batch size as necessary based on your dataset and computational resources.

Source Code

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Reshape,
Flatten, Input, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.optimizers import Adam
```

In [ ]:

```
# Display image function
def display_image(image_data, title="Image"):
    plt.imshow(image_data)
    plt.title(title)
    plt.axis('off')
    plt.show()
```

In [ ]:

```
# Load images from directory
def load_images(directory, image_size=(128, 128)):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            img = Image.open(os.path.join(directory, filename))
            img = img.resize(image_size)  # Resize to desired pixels
            img = np.array(img)
            if len(img.shape) == 2:
                img = np.expand_dims(img, axis=-1)  # Add channel dimension
if grayscale
```

```
                img = np.repeat(img, 3, axis=-1)  # Convert grayscale to
RGB
            images.append(img)
    return np.array(images)
```

In [ ]:

```
# Define the network architecture using Keras
input_shape = (128, 128, 3)  # Input size for RGB images

input_img = Input(shape=input_shape)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = Conv2D(128, (3, 3), activation='relu', padding='same', strides=(2,
2))(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', strides=(2,
2))(x)
x = Flatten()(x)
encoded = Reshape((32, 32, 256))(x)

x = Conv2DTranspose(256, (3, 3), activation='relu', padding='same',
strides=(2, 2))(encoded)
x = Conv2DTranspose(128, (3, 3), activation='relu', padding='same',
strides=(2, 2))(x)
x = Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())
```

In [ ]:

```
# Training function
def train_network(train_data, epochs, batch_size):
    autoencoder.fit(train_data, train_data, epochs=epochs,
batch_size=batch_size, shuffle=True)
```

In [ ]:

```
# Visualize results
def visualize_results(train_data):
    for img in train_data[:5]:
        img = img.reshape(128, 128, 3)
        colorized_output = autoencoder.predict(img.reshape(1, 128, 128, 3))
        colorized_output = (colorized_output[0] * 255).astype(np.uint8)

        plt.figure(figsize=(12, 4))
        plt.subplot(1, 3, 1)
        display_image(img, title="Original")
        plt.subplot(1, 3, 2)
        display_image(colorized_output, title="Colorized Output")
        plt.subplot(1, 3, 3)
        display_image(img, title="Original")  # Show original again for
comparison
        plt.show()
```

In [ ]:

```
# Save the model
def save_model():
    autoencoder.save_weights("autoencoder_weights.h5")
```

In [ ]:

```
# Load the model
def load_model():
    autoencoder.load_weights("autoencoder_weights.h5")
```

In [ ]:

```
# Evaluate the network on test data
def evaluate_network(test_data):
    total_loss = 0
    for img in test_data:
        img = img.reshape(1, 128, 128, 3)
        output_data = autoencoder.predict(img)
        total_loss += np.mean((output_data - img) ** 2)

    avg_loss = total_loss / len(test_data)
    print(f"Test Loss: {avg_loss}")
```

In [1]:

```
if __name__ == "__main__":
    # Example usage

    # Load images from a directory
    train_data = load_images(r"/content/pics")

    # Normalize data
    train_data = train_data.astype(np.float32) / 255.0

    # Train the network
    train_network(train_data, epochs=50, batch_size=16)

    # Visualize some results
    visualize_results(train_data)

    # Save the model
    save_model()

    # Load the model (for testing purposes)
    load_model()

    # Evaluate on test data (using training data for simplicity)
    evaluate_network(train_data)
```

Output

Colorized Output

Original

Colorized Output

Original