Tiramisu GAN for Image-to-Image Translation

This codebase implements a Generative Adversarial Network (GAN) architecture based on the Tiramisu network, a Densely Connected Convolutional Network (DenseNet) architecture specifically designed for image-to-image translation tasks.

Architecture

Generator

- 54-layer Tiramisu network inspired by DenseNet
- Encoder-decoder structure with dense blocks and transition layers
- Encoder: 5 dense blocks, each with 4 composite layers (Conv, BatchNorm, ReLU, DropOut)
- Decoder: 5 dense blocks, each with 4 composite layers and transposed convolutions for upsampling
- Skip connections for feature reuse and improved gradient flow
- Generates high-resolution output images from low-resolution inputs

Discriminator

- PatchGAN architecture
- Classifies overlapping patches in an input image as real or fake
- 4 convolutional layers with increasing kernel sizes and filter numbers
- Takes input and generated output images, produces a probability map

Loss Functions

- GAN loss: Standard adversarial loss
- L1 loss: Enforces pixel-wise reconstruction accuracy
- VGG loss: Based on pre-trained VGG-19 network for perceptual quality
- Final generator loss is a weighted sum of GAN, L1, and VGG losses

Training and Validation

- Adversarial training: Generator and discriminator trained alternately
- Validation: Periodic evaluation on a separate validation set
- Checkpointing: Network saved if validation score (PSNR + SSIM) improves
- Sample image generation and saving for visualization

Customization and Flexibility

- Support for custom datasets (input and target image directories)
- Adjustable hyperparameters: Learning rate, batch size, loss weights, architecture parameters
- Training, testing, and inference modes

Evaluation Metrics

- Peak Signal-to-Noise Ratio (PSNR)
- Structural Similarity Index (SSIM)

Implementation Details

- TensorFlow library for model construction and training
- Adversarial training procedure
- Validation and checkpointing mechanisms
- Inference mode for generating new images
- Reproducibility: Environment setup instructions, sample data

Key Highlights

- **Architectural Complexity**: 54-layer Tiramisu network with dense blocks, skip connections, and encoder-decoder structure.
- **Multiple Loss Functions**: GAN loss, L1 loss, and VGG loss for optimizing different aspects of generated images.
- **Evaluation Metrics**: PSNR and SSIM for quantifying image quality and similarity.
- **Customization and Flexibility**: Support for custom datasets, adjustable hyperparameters, and various modes (training, testing, inference).
- **Implementation Details**: Utilizes TensorFlow, adversarial training, validation, checkpointing, and inference modes.
- **Reproducibility**: Environment setup instructions and sample data provided.

This codebase is available as a GitHub repository and was initially built using TensorFlow 1. However, to make it compatible with the latest TensorFlow 2 version, several changes were made to the codebase. These changes include updating the code to use the latest TensorFlow 2 APIs and ensuring compatibility with the new version's features and functionalities.

Keeping in mind the memory limitations in collab only a 4th of the data was loaded at a time , and this was randomly chose so the model is training on different data on each training session .

Training Report

Model: GAN (Generative Adversarial Network) Dataset: Custom dataset of 1700 images

Training Details:

- Batch Size: 3
- Learning Rate: 0.001 (default)
- Total Training Epochs: 200 (default)

Checkpoint Analyzed: gan-1839

Based on the checkpoint name gan-1839, the following statistics summarize the training progress:

Generator/Discriminator Iterations: 919.5 Total Training Examples Seen: 2758.5 Total Training Epochs Completed: Approximately 1.62 epochs

The gan-1839 checkpoint represents fairly early stages of training. At this point, the model has completed only 1.62 epochs out of the specified 200 epochs on the 1700 image training dataset.

While the total number of generator/discriminator iterations is 919.5, the relatively small batch size of 3 means that so far the model has only seen 2758.5 training examples, which is just over 1.5 epochs worth of the dataset.

To fully train the model, significantly more epochs need to be completed to sufficiently cover the entire training dataset multiple times and converge to a good solution

The checkpoint file generated during training saves the absolute path to the weight file on the machine where training occurred. When trying to load this checkpoint file on a different machine for inference, it causes an error because the absolute path is different, and the weight file cannot be located.

To resolve this issue and enable running inference on a different machine, a script was written to generate a new checkpoint file with the correct path for the local machine,This script runs at the beginning of main.py so that the necessary checkpoint file is created