AI-Driven Image Synthesis for Data Augmentation

Nathan Fargo
Linear Fox Labs
ntfargo@proton.me

Project Published 2023.06.15
Github: https://github.com/Linear-Fox-Labs/AISynthDataAug

Abstract

This project aims to develop an AI-driven image synthesis tool using deep learning techniques for data augmentation in computer vision tasks. Traditional data augmentation methods often lack diversity and fail to generate realistic variations. The proposed tool leverages generative adversarial networks (GANs) to create high-quality synthetic images that closely resemble real-world data. The project involves steps such as dataset preparation, GAN model training, image synthesis, data augmentation, and model training and evaluation. By integrating the generated synthetic images with the original dataset, the tool enhances data diversity, improves model generalization, reduces data collection efforts, and adapts to various computer vision tasks. This project offers the potential to boost the performance and accuracy of computer vision models, even for individual developers working with Python.

Keywords: AI-driven image synthesis, data augmentation, deep learning, computer vision, generative adversarial networks (GANs), Python.

1 How it works

The AI-driven image synthesis data augmentation tool works through a series of steps to generate synthetic images for data augmentation:

- **Dataset Preparation:** A representative dataset is collected for the target computer vision task, such as image classification or object detection. The dataset is annotated with relevant labels or annotations to facilitate model training.
- **Image Synthesis:** The trained GAN model is utilized to generate synthetic images that mimic the characteristics of the original dataset. By providing input noise to the generator network, it generates new images based on the learned patterns and features from the training dataset. The synthesis process can be controlled by specifying desired variations, such as lighting conditions, occlusions, or backgrounds.
- **Data Augmentation:** The generated synthetic images are integrated with the original dataset to create an augmented dataset. This augmented dataset contains a combination of real and synthetic images, increasing the diversity and variability of the training data. Randomization techniques are applied during training to avoid overfitting, such as randomly selecting augmented images and shuffling their order.
- **Model Training and Evaluation:** The augmented dataset is used to train existing computer vision models, such as Convolutional Neural Networks (CNNs), using popular machine learning frameworks like TensorFlow or PyTorch. The models are trained on the augmented

dataset to learn from the expanded and diverse data. The performance of the trained models is evaluated on separate test datasets, comparing the results with models trained solely on the original dataset.

- **Benefits and Impact:**
    1. Increased data diversity
    2. Enhanced model generalization
    3. Reduced data collection efforts
    4. Adaptability to various computer vision tasks

2 Code Explanation — GAN Train

The code consists of two main functions: `train` and `save_generated_images`.

The `train` function begins by loading the MNIST dataset using `mnist.load_data()`. The images are normalized to the range $[-1, 1]$ by dividing by 127.5 and subtracting 1.0. This normalization is performed to improve the training process.

The `latent_dim` variable represents the dimensionality of the generator's input noise. In this case, it is set to 100.

Next, the generator, discriminator, and GAN models are created using the `build_generator`, `build_discriminator`, and `build_gan` functions, respectively. The generator takes the `latent_dim` as input, while the discriminator takes the shape of the input images (excluding the batch size).

Both the discriminator and GAN models are compiled with the binary cross-entropy loss and the Adam optimizer. The learning rate is set to 0.0002, and the beta parameter (`beta_1`) is set to 0.5.

Training loop begins, iterating over the specified number of epochs. For each epoch, a random batch of real images is selected from the training dataset. Additionally, a batch of noise vectors is generated to feed into the generator. The generator produces fake images based on this noise.

The discriminator is trained twice in each epoch: once with the real images and once with the fake images. The discriminator loss is calculated as the average of the losses from real and fake images. Generator is then trained by generating another batch of noise vectors and using them to produce fake images. The GAN model is trained by passing the generated noise and target labels (ones) to the GAN model. The generator tries to fool the discriminator into classifying the fake images as real.

2.1 Usage

To use this code, you need to have the `model.py` file in the same directory, which should contain the definitions for the `build_generator`, `build_discriminator`, and `build_gan` functions.

You can call the `train` function with the desired number of epochs, batch size, and save interval. The training process will begin, and the discriminator and generator losses will be printed after each epoch. Generated images will be saved every `save_interval` epochs.

```
train(epochs=100, batch_size=128, save_interval=10)
```

This code is a simplified implementation and may not produce high-quality results. Further improvements and optimizations can be made based on specific use cases.

```
f'Epoch {epoch+1}/{epochs} - D loss: {d_loss:.4f} - G loss: {g_loss:.4f}'
```

3 Future and next plans

Progressive Growing of GANs: Implement the progressive growing technique for GANs, where the generator and discriminator are gradually trained on images of increasing resolution. This approach can lead to generating higher-resolution and more realistic images.

Training on Larger Datasets: Training the GAN model on larger and more diverse datasets beyond MNIST. Datasets such as CIFAR-10, CelebA, or ImageNet can provide a more challenging and realistic training environment, leading to improved image generation capabilities.

- Evaluation Metrics
- Hyperparameter Optimization
- Conditional GANs