***DataAugmentationForMoreDataSyntheticGen.py***

**Overview**

This script performs extensive data augmentation on image datasets to increase the training data available for StyleGAN models. It applies multiple transformations to each source image to create varied synthetic versions while preserving the original image characteristics.

**Features**

- Applies multiple augmentations to each source image

- Supports configurable augmentation intensity and probability

- Preserves directory structure from input to output

- Includes options to retain original images alongside augmented versions

- Uses deterministic random seed for reproducibility

- Progress tracking with tqdm

- Error handling to continue processing despite individual image failures

**Augmentation Types**

- Geometric: rotation, horizontal flips, elastic transforms, grid distortion

- Color/Intensity: brightness, contrast adjustment, CLAHE enhancement, sharpening

- Noise: Gaussian noise, multiplicative noise

- Blur: Gaussian blur, motion blur

**Requirements**

- Python 3.6+

- OpenCV (cv2)

- NumPy

- Albumentations

- tqdm

**Configuration**

Adjust these parameters at the top of the script:

- `INPUT\_DIR`: Directory containing source images

- `OUTPUT\_DIR`: Directory where augmented images will be saved

- `NUM\_AUGMENTATIONS`: Number of augmented versions to create per image

- `ROTATION\_RANGE`: Maximum rotation angle in degrees

- `BRIGHTNESS\_RANGE`: Maximum brightness adjustment factor

- `CONTRAST\_RANGE`: Maximum contrast adjustment factor

- `FLIP\_PROB`: Probability of horizontal flipping

- `BLUR\_PROB`: Probability of applying blur

- `NOISE\_PROB`: Probability of adding noise

- `INCLUDE\_ORIGINALS`: Whether to copy original images to output directory

- `RANDOM\_SEED`: Seed for random number generation

**Usage**

1. Set the configuration variables at the top of the script

2. Run the script: `python DataAugmentationForMoreDataSyntheticGen.py`

**Output**

- The script generates augmented versions of each input image with naming format:

`{original\_filename}\_aug\_{augmentation\_number}.{extension}`

- All images are saved to the output directory, preserving any subdirectory structure from the input.

***DCFace FineTune 5X5.py***

**Overview**

This script implements a comprehensive pipeline for fine-tuning the DCFace GAN model to generate specific facial emotions. It builds on the 5x5 DCFace architecture to create high-quality, controllable emotional expressions on synthetic faces.

**Features**

- Complete GAN training pipeline (generator and discriminator)

- Supports resuming interrupted training sessions

- Mixed precision training for improved performance

- Gradient clipping for training stability

- Label smoothing for discriminator robustness

- Comprehensive visualization and progress tracking

- Automatic report generation with training metrics

- Learning rate scheduling with cosine annealing

- Configurable model architecture

**Requirements**

- Python 3.8+

- PyTorch 1.9+

- CUDA-compatible GPU

- torchvision

- matplotlib

- tqdm

- Pillow (PIL)

- numpy

**Architecture**

- Generator: 6-layer upsampling network with 5x5 convolutions

- Discriminator: 6-layer downsampling network with 5x5 convolutions

- Input: 512-dimension latent vector

- Output: 256x256 RGB images

**Usage**

1. Configure the settings in the `CONFIG` dictionary:

- Set data paths

- Adjust training parameters

- Configure model parameters

- Set checkpoint locations

2. Run the script

Configs:

- `data\_root`: Path to the preprocessed emotion dataset

- `emotion`: Target emotion to generate (e.g., "disgust", "angry", "fear")

- `pretrained`: Path to the pretrained DCFace model

- `batch\_size`: Number of images per batch

- `epochs`: Total training epochs

- `lr\_g/lr\_d`: Learning rates for generator and discriminator

- `mixed\_precision`: Whether to use mixed precision training

- `clip\_grad`: Whether to apply gradient clipping

**Output**

- Checkpoints saved periodically during training

- Sample images generated throughout training

- Training visualizations (loss curves, discriminator scores)

- Comprehensive HTML training report

- Image evolution grid showing progression over time

**Notes**

- For optimal results, use a preprocessed dataset with cropped, aligned faces

- Training time varies based on dataset size and GPU performance

***DCFace256ImageGeneration.py***

**Overview**

This script generates emotional expression images using a fine-tuned DCFace model. It supports multiple generation modes: random generation, ID-seeded generation, and style mixing between identity and emotion images for controlled expression transfer.

**Features**

- Multiple generation modes:

- Random generation with truncation for quality control

- Identity-seeded emotion generation

- Style mixing between identity and emotional expression images

- Image enhancement with configurable contrast, brightness, and sharpness

- Batch processing of image sets

- Grid visualization of generated results

- Individual image saving with detailed naming

- Simple encoder for projecting reference images into latent space

**Requirements**

- Python 3.7+

- PyTorch 1.8+

- torchvision

- Pillow (PIL)

- numpy

- matplotlib

- tqdm

**Configuration**

The script uses a configuration dictionary with the following key parameters:

- `model\_path`: Path to the fine-tuned DCFace generator model

- `output\_dir`: Directory where generated images will be saved

- `id\_dir`: Optional directory containing identity reference images

- `style\_dir`: Optional directory containing emotion reference images

- `num\_images`: Number of images to generate in random mode

- `truncation`: Truncation factor for quality control (lower = better quality, less diversity)

- `alpha`: Style mixing strength (0.0 = only ID, 1.0 = only emotion)

- `enhance`: Whether to apply post-processing enhancement

- `resolution`: Output image resolution (default: 256x256)

**Usage**

1. Adjust the configuration settings at the top of the script

2. Run the script: `python DCFace256ImageGeneration.py`

**Generation Modes**

The script automatically selects the appropriate generation mode based on the provided directories:

1. Random Generation: When neither ID nor style directories are provided

2. ID-Seeded Generation: When only ID directory is provided

3. Style Mixing: When both ID and style directories are provided

**Output**

For each generation mode, the script produces:

- A grid of all generated images

- Individual images (if `save\_individual` is set to True)

- Visual comparison between original and generated images

- Matplotlib visualization of the results

**Notes**

- For best results, use aligned face images in the ID directory

- The style encoder is a simple approximation; results may vary

- Adjust the truncation parameter to control quality vs. diversity

- The alpha parameter controls how strongly the emotion is applied