Report on the DCGAN Pipeline for Generating High-Quality Images

Introduction

The purpose of this report is to provide a detailed explanation of a deep convolutional generative adversarial network (DCGAN) pipeline for generating high-quality images from a given dataset. The pipeline involves several steps, including preprocessing the input images, defining a Kernel Inception Distance (KID) metric for evaluating the quality of the generated images, applying an Adaptive Augmenter to enhance the model's ability to generalize to new data, and training the DCGAN model using the Adam optimizer. Additionally, the report discusses how the generated images are post-processed using bicubic interpolation for resizing, median blur for denoising, and a filter for sharpening. Finally, the Enhanced Super-Resolution Generative Adversarial Networks (ESRGAN) method is used to improve the resolution of the generated images.

Preprocessing the Input Images

Before training the DCGAN model, the input images need to be preprocessed to ensure they are suitable for the model. In the implemented code, the images are read from a folder and resized to 32x32 pixels using the tf.image.resize function. This is done to ensure all images have the same shape. For this particular case, all images are of the same shape, however on some datasets, the individual images may vary in size, so it is a good practice to resize all images to the same size. This way we ensure that the dataset given to the model is uniform and it will be able to process the images properly.

Additionally, the pixel values are converted to the range [0, 1] by dividing by 255.0 and removing the fourth channel (if it exists). This is a common preprocessing step for image data.

Regarding the size of the images, their being only 32 pixels presents both some advantages and disadvantages. On one hand, the smaller amount of data points per image leads to shorter training times for the DCGAN model. However, the small resolution of the images also affects the resolution and quality of the generated images and causes them to be less detailed than expected.

Overall, this size presents a good balance between the resolution of the generated images and the training time required.

Defining the KID Metric

To evaluate the quality of the generated images, a Kernel Inception Distance (KID) metric is defined. The KID metric measures the distance between the distributions of features extracted from real and generated images using a pre-trained InceptionV3 model. This is a popular metric for evaluating the similarity between two sets of images. The KID metric measures the distance between the distributions using a polynomial kernel function. This metric provides a quantitative measure of the similarity between the real and generated images, allowing for the model to be optimized to generate high-quality images.

Applying an Adaptive Augmenter

To enhance the model's ability to generalize to new data and prevent overfitting, an Adaptive Augmenter is used. This technique randomly applies different predefined data augmentation techniques with varying probabilities during the training process. The augmentation probability is adjusted based on the

discriminator's accuracy on the real images. The idea is that if the discriminator can distinguish real images from generated images too easily, then the augmentation should be increased to make the training more challenging. Conversely, if the discriminator struggles to distinguish real and generated images, then the augmentation should be decreased to reduce overfitting. By randomly applying different transformations to the input images during training, the model can learn to be more robust to variations in the input data.

Exponential Moving Average (EMA)

During training, the generator's weights are tracked by two models: the original generator model and an EMA version of the generator model. The EMA generator model has its weights updated with a moving average of the original generator's weights. The idea is to use the EMA generator model for generating images during inference, as its weights are more stable and produce better quality images than the original generator.

DCGAN Model Architecture

The architecture of the DCGAN model is defined using convolutional and deconvolutional layers. Convolutional layers are used to extract features from the input images, while deconvolutional layers are used to generate new images from a noise vector. The discriminator network uses convolutional layers to distinguish between real and generated images. Same convolutions in the generator allow for more data to be interpreted without increasing the size, which can reduce the risk of overfitting.

The generator architecture consists of a sequence of transposed convolutional layers followed by batch normalization and ReLU activation functions. The transposed convolutional layers progressively increase the spatial dimensions of the output while maintaining the number of channels until the desired image size is reached. During this process, same convolutions are also applied to increase the number of channels and so the features to be examined, while maintaining their size. The batch normalization and ReLU activation functions help stabilize the training and introduce non-linearity into the model.

The discriminator architecture consists of a sequence of convolutional layers followed by batch normalization and LeakyReLU activation functions. The convolutional layers progressively decrease the spatial dimensions of the output until a one-dimensional score is reached. The LeakyReLU activation function introduces non-linearity into the model and prevents the gradients from vanishing.

Training the DCGAN Model

Once the architecture of the GAN model was defined, the next step was to train the model using the chosen dataset. Here it is important to note, that I decided to use the individual Cifar datasets at a time for training due to limited computing power and time. In early trials to process the whole dataset it quickly became very clear, that that process would last for a very long time. Training one dataset at a time still takes a very long time, but it is doable.

In order to train the DCGAN model, I chose to use the Adam optimizer and binary cross-entropy loss function, which are standard settings for GAN training.

To ensure that the generator and discriminator networks are trained evenly, I alternated between training the discriminator network and the generator network. The discriminator was trained on both real and fake images and the generator was trained to generate realistic-looking images that could fool the

discriminator. The learning rate for the optimizer was set to 0.0002, which I found to be optimal for this model configuration and allows for quick learning, especially for the first epochs.

The batch size for training was set to 128, which was found to be a good trade-off between training time and model performance. The model should be trained for 500 epochs for good results, with the training loss for both the generator and discriminator being tracked and monitored.

In my process to choose this architecture, hyperparameters, and model design I tested different configurations and progressively improved various parts of the model. This way, I trained different model configurations with distinct hyperparameters and for a different number of epochs.

Post Processing

After training the model, the generated images can be further improved through post-processing techniques. In this particular case, the generated images are resized, and their resolution is improved using ESRGAN (Enhanced Super-Resolution Generative Adversarial Networks). The post-processing is done using Python code that implements a resizing function, a resolution improvement function, and a plotting function.

The post-processing starts with the post-process function, which resizes the image to twice its original size using the bicubic method. The bicubic method is a commonly used interpolation method for resizing images that produces a high-quality output, resulting in smoother and less pixelated images. The resized image is then denoised using OpenCV's median filter to reduce noise in the image. Finally, the SHARPEN filter is applied using the PIL library to enhance the sharpness of the image and increase its clarity and detail.

While the generated images looked quite good after the first post-processing step, they still suffered from low resolution, which limited their detail and made them less realistic. To address this issue, I applied the Enhanced Super-Resolution Generative Adversarial Networks (ESRGAN) method to improve the resolution of the generated images. ESRGAN is a state-of-the-art method for image super-resolution that uses a GAN architecture to generate high-resolution images from low-resolution input images. The ESRGAN model was trained on a dataset of high-resolution images and then used to increase the resolution of the generated images produced by the DCGAN model.

So, the next post-processing step is to load the ESRGAN model using TensorFlow Hub. The resize_res function prepares the image for the ESRGAN model by first applying median blurring to reduce the noise in the image. The function then expands the dimensions of the image and scales it by 255 to make them suitable for the model. The ESRGAN model is applied to the image to produce an improved version of it with a bigger size and higher resolution. The resulting image is then scaled back down and returned. This method significantly improved the resolution and detail of the generated images, making them look much more realistic and high-quality.

The plot_images_post function generates a large number of images and selects the n best images based on the output from the discriminator. For each selected image, the function applies two rounds of resizing and sharpening using the post-process function. Then, the image is further improved using the ESRGAN model to improve its resolution. Finally, the function plots the original image, the two resized and sharpened versions, and the two improved versions side-by-side for visual comparison.

Post-processing techniques are commonly used to improve the quality of images generated by GANs. In this particular case, the post-processing techniques include two steps: image resizing and resolution improvement using the ESRGAN model. The bicubic interpolation method is used for resizing, while median blurring is used for denoising and the sharpen filter is applied to enhance the sharpness of the image. These techniques are used to enhance the quality of the generated images and improve their visual appearance.

Results

After training the DCGAN model, it was evaluated by generating new images using the trained generator network. The generated images were then compared with the original images from the Cifar dataset.

The generated images were found to be visually similar to the original images in terms of their overall shape, texture, and color. The generated images also displayed a similar level of diversity compared to the original images, indicating that the model was able to capture the underlying distribution of the dataset. Here it is also very important to mention that the images used for training have a very small size and a less-than-optimal resolution. This consequently causes the generated images to display a similar, not too high quality, level of resolution and detail, but still depict the same underlying patterns and shapes.

Conclusion

In conclusion, the DCGAN model was successful in generating new images that were visually similar to the original images from the Cifar dataset. The model was able to capture the underlying distribution of the dataset and generate new images that displayed the same level of diversity as the original images.

The approach of using a DCGAN model with a convolutional neural network architecture and the Adam optimizer proved to be an effective method for training a GAN to generate new images. The use of convolutional layers in the architecture allowed the model to capture both local and global features of the images, while the Adam optimizer provided fast and stable convergence during training.

Overall, the successful implementation of the DCGAN model demonstrates the potential of GANs for generating realistic and diverse images for a wide range of applications. Particularly, this small implementation presented many limitations including the resolution of the given images, the computing power, and even the amount of data used for training and the time. Even with those limitations and the short time invested into its development, this DCGAN already showed promising results and only demonstrates the potential of this innovative technology. I, myself already have plenty of ideas on how to improve the performance of the model, methods that could be used to improve GANs in general, and its many possible implementations. With further research and development, GANs are likely to become an increasingly important tool for generating new and innovative images.