**Implementation and Evaluation of Diffusion Models (DDPM, DDIM, Latent Diffusion Models) in Image Generation**

11-785: Introduction to Deep Learning (Fall 2024)

**Team:**

Aditya Aayush (aaayush)

Yash Jaiswal (yjaiswal)

Priya Lalwani (plalwani)

**1. Introduction**

Diffusion models have emerged as one of the most promising approaches in generative modeling, especially for tasks like image generation. These models use a forward process of adding noise to data and then learn to reverse this process, effectively denoising the data in a controlled manner to generate new samples. This report focuses on the implementation and evaluation of **Denoising Diffusion Probabilistic Models (DDPM)**, **Denoising Diffusion Implicit Models (DDIM)**, **Classifier-Free Diffusion Guidance (CFG)** and **Latent Diffusion Models**.

We will explore the implementation of these models from scratch, discuss their core principles, and evaluate their performance using metrics such as **Frechet Inception Distance (FID)** and **Inception Score (IS)**. The report also includes a detailed explanation of the implemented code, which encompasses various components of the diffusion process, from noise scheduling to model inference.

**2. What are Diffusion Models?**

Diffusion models are generative models that are based on the idea of adding noise to data and then learning how to reverse this process. They consist of two main phases: a forward process and a reverse process.

**Forward Process:**

In the forward process, noise is gradually added to the data, simulating a diffusion process. Starting with a clean image, noise is injected into the image over multiple timesteps until it becomes pure noise. This forward process is typically modeled as a **Markov chain**, where the data at each timestep is dependent on the data from the previous timestep.

**Reverse Process:**

In the reverse process, the model learns to reverse the diffusion, denoising the image step-by-step, and recovering the original data distribution. This reverse process is learned by training the model to predict the noise added at each timestep, which is then subtracted from the noisy image to obtain a cleaner version. By repeating this process for multiple timesteps, the model can generate new data from noise.

**Diffusion Probabilistic Models (DDPM):**

The **Denoising Diffusion Probabilistic Model (DDPM)** is the classical version of diffusion models. It models the forward diffusion process as a sequence of steps where each step adds a small amount of noise. The reverse process is then learned to gradually denoise the data. DDPMs have been shown to generate high-quality images and have become a popular choice in generative modeling.

**3. Architecture of the Models**

**3.1 UNet Architecture**

The UNet architecture is at the heart of the DDPM and DDIM models, designed for processing images in the context of generative tasks. It is a type of encoder-decoder architecture with skip connections, making it ideal for tasks that involve reconstructing images from noisy data. The key components of the UNet architecture are as follows:

* **Input Layer:** The model takes an input image, typically with multiple channels (e.g., RGB). The image is passed through an initial convolution layer (stem) that reduces the spatial dimensions and prepares the feature map for the subsequent layers.
* **Downsampling Blocks:** The network includes a series of downsampling blocks, where each block applies a residual connection (via ResBlock) and optional attention mechanisms. This reduces the spatial resolution while increasing the feature depth, capturing higher-level abstract features.
* **Middle Blocks:** At the bottleneck of the UNet, the middle blocks process the highest-level features with additional residual layers and attention blocks, which help the network capture more complex dependencies in the data.
* **Upsampling Blocks:** The upsampling part of the architecture is responsible for reconstructing the image from the feature representations learned in the downsampling and middle layers. The upsampling blocks increase the spatial resolution while merging features from the corresponding downsampling blocks using skip connections.
* **Output Layer:** Finally, the output is passed through a GroupNorm layer followed by a convolution to map the features to the original image dimensions.

**3.2 VAE Architecture**

For **Latent Diffusion Models**, the VAE (Variational Autoencoder) plays a crucial role in compressing the image into a lower-dimensional latent space. Here's an overview of the VAE architecture:

* **Encoder:** The encoder compresses the input image into a latent representation. It consists of multiple layers of convolutions, followed by normalization and activation functions. The encoder's final output is a pair of vectors representing the mean and log-variance of the Gaussian distribution from which latent codes will be sampled.
* **Latent Space Sampling:** Using the reparameterization trick, the model samples latent codes from the Gaussian distribution defined by the mean and variance vectors. This latent representation captures the essential features of the input data in a compressed form.
* **Decoder:** The decoder takes the sampled latent code and reconstructs the original image. It uses a series of upsampling and convolutional layers to progressively increase the spatial dimensions and refine the image. The final output is an image with the same dimensions as the input.
* **Latent Diffusion:** In the case of latent diffusion, the noise diffusion process is applied directly in the latent space instead of the pixel space, reducing the computational cost while still maintaining the quality of the generated images.

**4. Denoising Diffusion Probabilistic Model (DDPM)**

DDPM is a probabilistic model that uses a Markov chain of diffusion steps to learn to generate data. The model is trained by simulating the forward process of adding noise and then learning the reverse process of denoising the data.

**Code Breakdown of the DDPM Scheduler:**

1. **Initialization of Timesteps, Betas, and Alphas**: The **DDPMScheduler** class initializes the parameters necessary for the diffusion process, including betas, alphas, and the cumulative product of alphas.

**A screenshot of a computer program

Description automatically generated**

* **Betas**: These represent the noise schedule and are calculated as a linearly spaced tensor between beta\_start and beta\_end.
* **Alphas**: These values correspond to how much of the original data remains after noise is added at each timestep.
* **Cumulative Alphas**: This is the cumulative product of alphas, representing the total effect of noise addition up to each timestep.

1. **Setting Timesteps for Inference**: The set\_timesteps method prepares the discrete timesteps for the reverse diffusion process, ensuring that the number of timesteps for inference matches the model's capabilities.

**A screenshot of a computer program

Description automatically generated**

* **Timesteps**: The method calculates and stores the specific timesteps to be used during inference, ensuring they align with the training process.

1. **Noise Addition**: The add\_noise method adds noise to the input data based on the current timestep, creating the noisy image required for training.

**A screenshot of a computer program

Description automatically generated**

**Noise Addition**: The method applies the noise at each timestep based on the schedule, mixing the noise with the original image to create a noisy sample for training.

**4. DDIM Scheduler**

DDIM (Denoising Diffusion Implicit Models) is an improvement over DDPM that reduces the number of timesteps required to generate high-quality images by introducing an alternative noise schedule.

1. **Variance Calculation**: In the **DDIM** scheduler, the variance is computed differently, allowing for faster image generation while maintaining image quality.

**A computer screen shot of a code

Description automatically generated**

**Variance**: DDIM uses a modified calculation for variance, leading to more efficient reverse diffusion steps. This allows the model to produce high-quality images with fewer inference steps compared to DDPM.

**5. DDPMPipeline**

The **DDPMPipeline** orchestrates the diffusion process by connecting the model, scheduler, and class embedder. It manages the reverse diffusion steps, generating images by applying the learned model to noisy samples.

The pipeline handles the end-to-end generation process, starting from random noise and applying reverse diffusion using the model and scheduler. It also supports class-guided generation when necessary.

**6. Class-Free Guidance (CFG)**

**Class-Free Guidance (CFG)** is an extension of the original diffusion models, primarily aimed at improving the generation of images by providing conditional guidance without the need for explicit class labels. This technique involves conditioning the diffusion process on both the noisy image and an auxiliary signal (such as class information), leading to more controlled generation, while maintaining the flexibility of unconditional generation.

**How CFG Works:**

* In CFG, the model is trained to predict the reverse diffusion process conditioned on a class or additional information. Instead of requiring a specific class to guide the generation, the guidance is provided without explicitly using the class label during inference.
* This method uses an auxiliary function to steer the denoising process, encouraging the generation to move toward desired characteristics while avoiding overfitting to a particular class.
* The model learns to generate images that are diverse yet adhere to some target property, making it a useful tool in image generation tasks like super-resolution, inpainting, or style transfer.

**7. Training Pipeline**

A diagram of a process

Description automatically generated

The training pipeline for implementing DDPM and related methods is illustrated below:

1. **Dataset:**
   * Start with preprocessed datasets such as CIFAR-10 and ImageNet-128.
2. **Pre-Processing and Transformations:**
   * Resize, normalize pixel values to [-1, 1], and convert to tensors.
3. **DataLoader:**
   * Efficiently load data in batches for processing.
4. **Training Loop:**
   * Encode input images using a pre-trained Variational Autoencoder (VAE).
   * Scale latent representations to maintain unit variance.
   * Sample random time step ttt and Gaussian noise.
   * Add noise using the DDPM/DDIM forward diffusion framework.
   * Predict noise using a U-Net model with the noisy image, the time step, and conditional class embedding as inputs.
   * Calculate Mean Squared Error (MSE) loss between predicted and true noise.
   * Perform backpropagation and update weights using an optimizer.

A diagram of a process flow

Description automatically generated**8. Post-Epoch Validation: Inference Pipeline**

This is a separate pipeline that takes place after training is completed or at the end of each epoch to validate the model’s performance. It focuses on generating new images and evaluating their quality.

The purpose of this module is to analyze and visualize the quality of generated images to understand how well the model captures the data distribution.

**Steps in the Inference Pipeline**

1. **Run Conditional Image Generation Through the Reverse Pipeline:**
   * Begin with a sample of random Gaussian noise as the starting point (*xt*​).
   * Apply the reverse diffusion process iteratively to remove the noise step-by-step.
     + At each step *t*, the model predicts and removes noise based on:
       - The current noisy image *xt*.
       - The time step *t*.
       - The conditional embedding.
   * This process continues until the model reconstructs an image x0x\_0x0​ from the initial noise.
   * The reverse diffusion is performed using either DDPM or the faster DDIM.
   * Conditional Class Embeddingisused to guide the model to generate images of a specific category.
2. **Rescale Generated Outputs:**
   * The output from the reverse pipeline is typically in the latent space or scaled to specific ranges.
   * Rescale the generated outputs to a normalized range, such as [−1, 1], for compatibility with the decoder.
3. **Decode Outputs Using VAE:**
   * Pass the rescaled latent representations into the Variational Autoencoder’s decoder.
   * The decoder reconstructs high-quality images from the latent representations by mapping them back to pixel space.

**9. Experiment Results**

We tested several combinations of models, datasets, schedules, and training epochs:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Experiment | Model | Dataset | Scheduler | CFG | Epochs |
| A | DDPM | ImageNet-128 | Cosine Beta | Yes | 10 |
| B | DDPM | ImageNet-128 | Linear Beta | Yes | 17 |
| C | DDPM | ImageNet-128 | Linear Beta | No | 10 |
| D | DDIM | CIDAR-10 | Linear Beta | No | 100 |
| E | DDPM | ImageNet-128 | Cosine Beta | Yes | X  (KL\_VAE included) |

**Generated Image Results**

The following subsection showcases the generated images from the model for various experimental configurations. Each set of images highlights the results obtained under specific conditions, including different beta schedulers, presence or absence of classifier-free guidance (CFG), and variations in datasets:

* 1. **Experiment A  
     **
  2. **Experiment B  
       
     A collage of images of a crocodile

     Description automatically generated**
  3. **Experiment C  
       
     A close up of food

     Description automatically generated**
  4. **Experiment D  
       
     A close up of a cat's face

     Description automatically generated**
  5. **Experiment E  
       
     A close-up of a picture

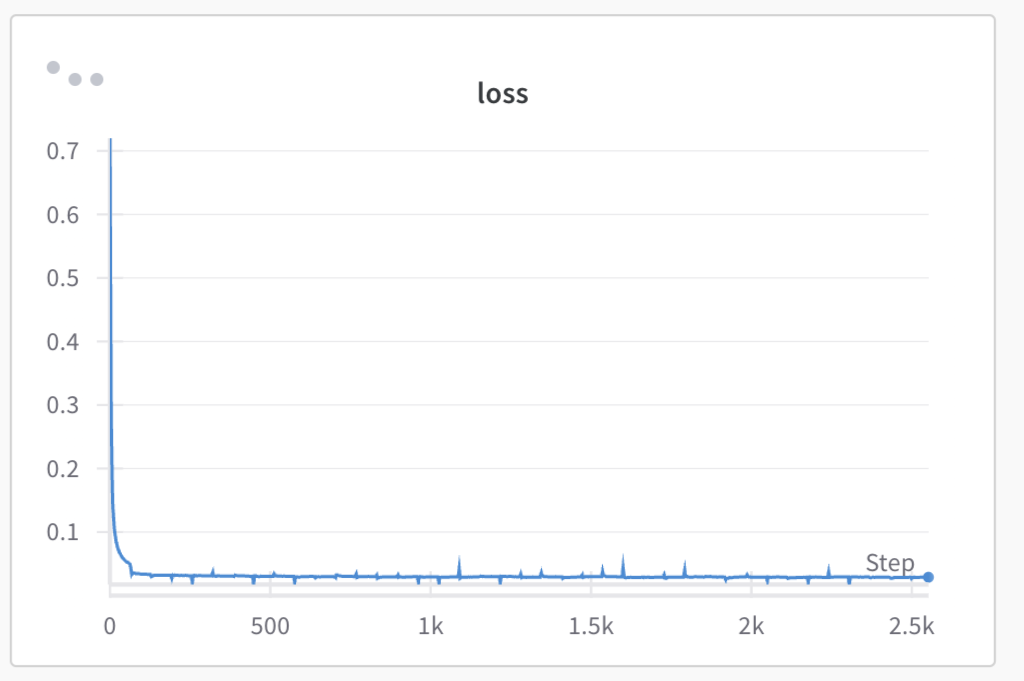
     Description automatically generated**

This gallery provides a visual comparison of the outputs, demonstrating how model configurations influence image fidelity, diversity, and adherence to conditional guidance. The progression across epochs and experimental conditions is evident in the varying levels of detail and quality in the generated images.

**Observations:**

* Models trained with Cosine Beta scheduling demonstrated higher fidelity and smoother outputs.
* Conditional generation with CFG resulted in better alignment of generated images with target classes.
* DDIM offered significant speedup.

**DDPM Results:**



**DDIM Results:**

