SAMSUNG PRISM - DeepCache Implementation

Stable Diffusion Model:

The Stable Diffusion (SD) model is a type of generative model used in machine learning for image generation. It is designed to generate high-quality images by modeling the data distribution in a probabilistic way. The key idea behind diffusion models is to transform a simple noise distribution into the desired data distribution through a series of steps. The "diffusion" refers to the gradual transformation of the noise into the desired data, and stability is introduced to ensure robustness in the training process.

In the context of image generation, the SD model uses a diffusion process to iteratively refine an initial noise signal until it converges to a realistic image. This process involves a series of steps, making it computationally intensive, especially during the inference phase. Approaches like DeepCache aim to improve the inference speed of such models by optimising certain steps.

2. LoRA (Local Rank Alignment):

LoRA is a technique designed to improve the training of deep neural networks by aligning the local ranks of samples in the feature space. It focuses on capturing and aligning the local relationships between samples, aiming to enhance the discriminative power of the learned representations. This technique is often applied in the context of image classification and feature learning.

In the context of your project, combining LoRA with the Stable Diffusion model post-DeepCache implementation may involve using LoRA to further optimise the feature representations learned by the model. The goal is to explore whether aligning local ranks can contribute to more efficient and faster inference without sacrificing the quality of generated images.

2. DeepCache with Stable Diffusion Model:

▶ EP96 - DeepCache: Accelerating Diffusion Models for Free

▶ How AI Image Generators Work (Stable Diffusion / Dall-E) - …

▶ Diffusion Models | Paper Explanation | Math Explained

Summarised Deep Cache

DeepCache is a new way to speed up diffusion models used in image synthesis without needing to retrain them. It works by saving and reusing certain parts of the model as it processes images, which reduces the need for repetitive calculations. By doing this, DeepCache makes diffusion models run faster while maintaining similar quality results. It's shown to be better than other methods like pruning and distillation that require retraining.

DeepCache is a technique designed to speed up the inference process in diffusion models by leveraging the redundancy present in consecutive steps of the reverse diffusion process. It draws inspiration from the caching mechanism used in computer systems, where frequently accessed data is stored for quick retrieval, thus reducing the need for repeated calculations.

Pruning: Pruning is a technique used to reduce the size of neural networks by removing unnecessary connections or weights. This process involves identifying and removing parameters that contribute less to the network's performance, thereby reducing computational costs and memory requirements.

Distillation: Distillation, also known as knowledge distillation, is a method used to transfer knowledge from a large, complex model (teacher model) to a smaller, simpler model (student model). The goal is to train the student model to mimic the behavior of the teacher model by learning from its predictions or representations. This allows for the creation of more efficient models that retain the performance of larger ones.

Here's a breakdown of how DeepCache works:

Utilizing Skip Connections: DeepCache focuses on the skip connections within the U-Net architecture, which offers a dual-pathway advantage. The main pathway requires heavy computation to traverse the entire network, while the skip pathway only needs to go through shallow layers, resulting in a much lower computational load.

Caching Features: DeepCache identifies features within the diffusion process that change slowly over time. These features are cached, meaning they are stored for later use rather than being recalculated in subsequent steps.

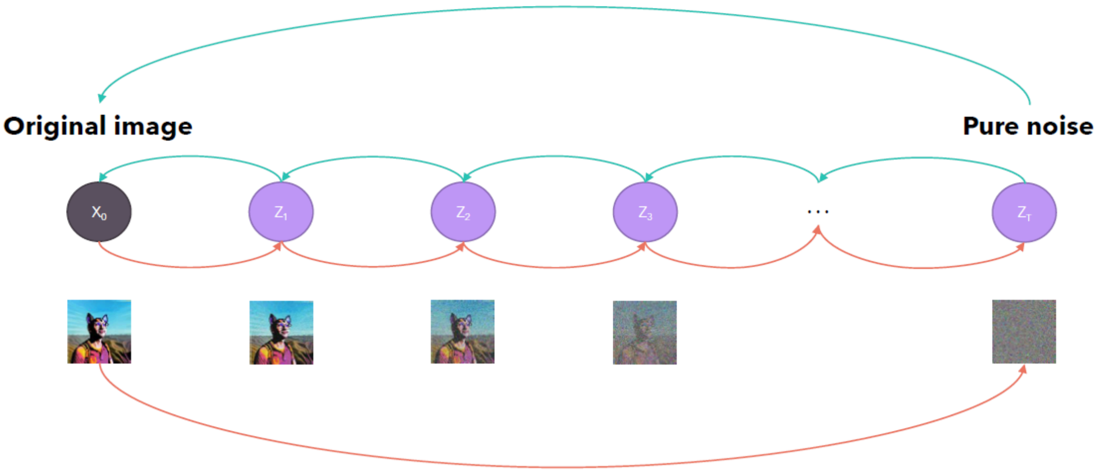
Dynamic Inference: During the inference process, DeepCache dynamically decides which parts of the network need to be recalculated and which parts can be retrieved from the cache. This reduces redundant computations, especially in regions where the features change slowly.

Partial Inference: Instead of performing a full inference at each step, DeepCache performs partial inference by recalculating only the necessary parts of the network and retrieving cached features for the rest. This significantly speeds up the process while maintaining accuracy.

Extending to Multiple Steps: DeepCache can be extended to cover multiple consecutive steps of the diffusion process. Cached features are reused across these steps, reducing computational overhead further.

Overall, DeepCache optimizes the inference process in diffusion models by strategically caching features and dynamically adjusting the computation load. This results in faster inference times without sacrificing the quality of the output.

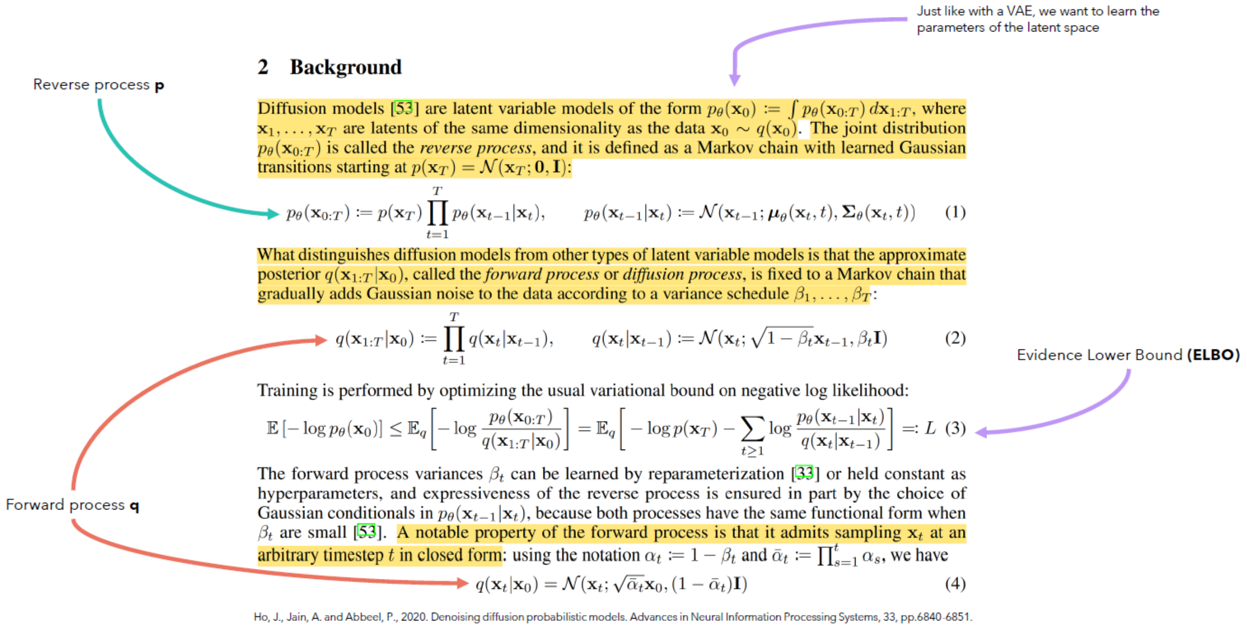
Reverse process: **Neural network**



Original image

Pure noise

Forward process: **Fixed**



Just like with a VAE, we want to learn the parameters of the latent space

Reverse process **p**

2 Background

Diffusion models [53] are latent variable models of the form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are latents of the same dimensionality as the data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The joint distribution $p_\theta(\mathbf{x}_{0:T})$ is called the *reverse process*, and it is defined as a Markov chain with learned Gaussian transitions starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{1}$$

What distinguishes diffusion models from other types of latent variable models is that the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, called the *forward process* or *diffusion process*, is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, \ldots, \beta_T$:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2}$$

Evidence Lower Bound **(ELBO)**

Training is performed by optimizing the usual variational bound on negative log likelihood:

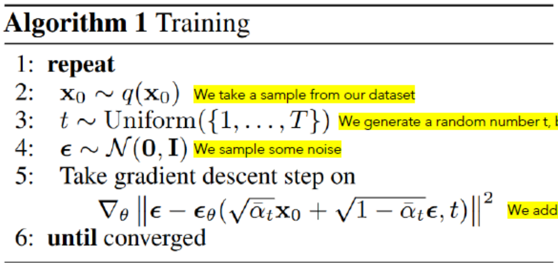
$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] =: L \tag{3}$$

Forward process **q**

The forward process variances β_t can be learned by reparameterization [33] or held constant as hyperparameters, and expressiveness of the reverse process is ensured in part by the choice of Gaussian conditionals in $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, because both processes have the same functional form when β_t are small [53]. A notable property of the forward process is that it admits sampling $\mathbf{x}_t$ at an arbitrary timestep t in closed form: using the notation $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, we have

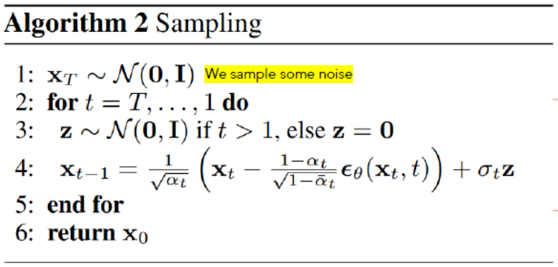
$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{4}$$

Ho, J., Jain, A. and Abbeel, P., 2020. Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems, 33, pp.6840-6851.



Algorithm 1 Training

```
repeat
    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$   We take a sample from our dataset
    $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$   We generate a random number t, between 1 and T
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$   We sample some noise
    Take gradient descent step on
        $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$   We add noise to our image, and we train the model to learn to predict the amount of noise present in it.
until converged
```

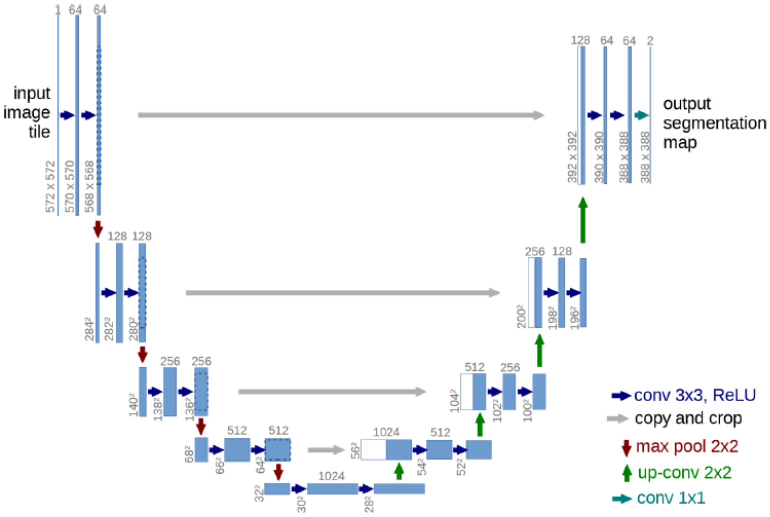


Algorithm 2 Sampling

```
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$   We sample some noise
for $t = T, \ldots, 1$ do
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$
```

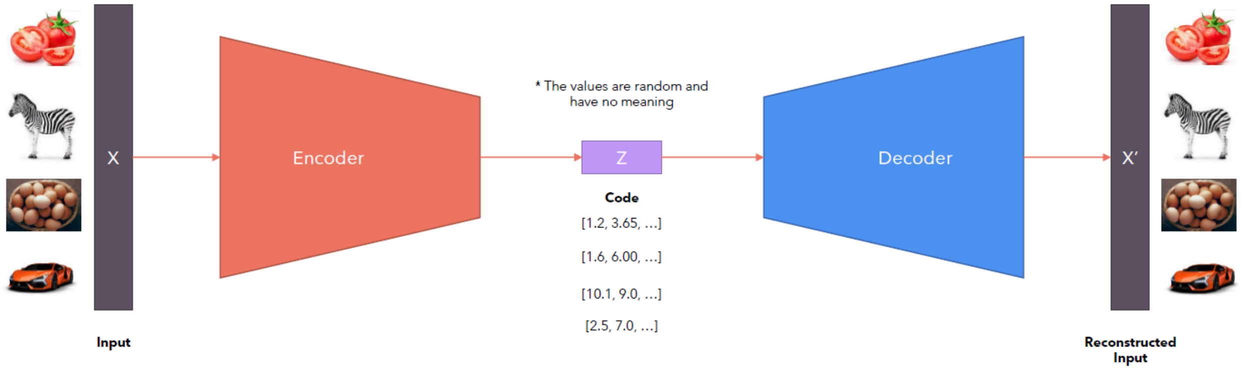
We keep denoising the image progressively for T steps.

U-Net

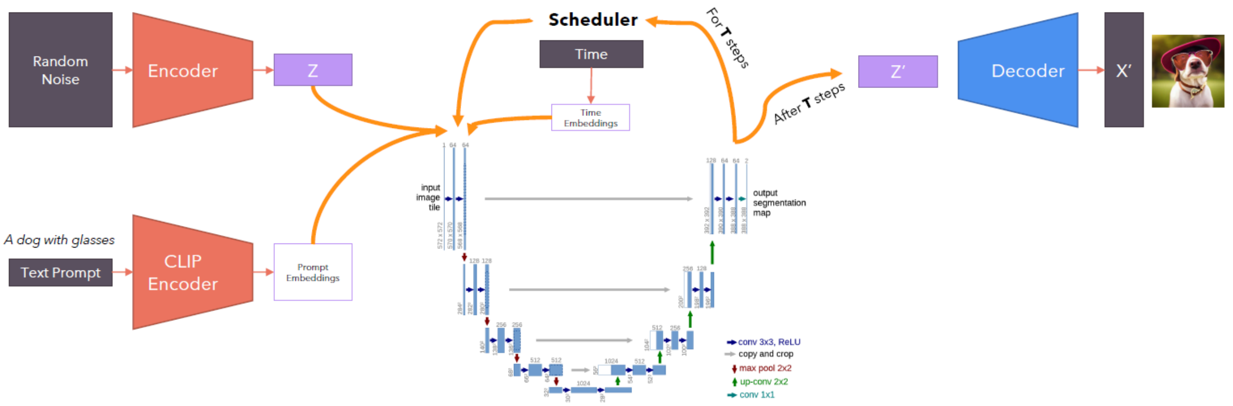


Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18* (pp. 234-241). Springer International Publishing.

AutoEncoder



Working Flow Diagram



Reference Links:

https://www.youtube.com/watch?v=YRhxdVk_sIs – CNN’s ▶ Visualizing Convolutional Filters from a CNN - Visual Representation

https://deeplizard.com/resource/pavq7noze3 - CNN visualisation (MAX POOLING CONCEPTS)