Video Frame Interpolation using Conditional Latent Diffusion Models

Yu Cheuk Hei
HKUST
chyuam@connect.ust.hk

Tsui Yuk Hang
HKUST
yhtsui@connect.ust.hk

Abstract

Most existing works rely on estimating bidirectional flows between neighbor frames to interpolate the intermediate frame. Although these methods have been shown to produce images with quality results, they are less capable of capturing perceptual differences between the images. In this work, we attempt to formulate video frame interpolation as an image synthetic task conditioned on neighbor frames. In particular, we use a VQGAN with perceptual loss for encoding, and latent diffusion models for interpolation. Results show that they have great potential and could be further investigated but at the same time, they also require a significantly large amount of computational power and training time.

1. Introduction

Watching videos with high resolution and frame rate demands a stable and high-speed network connection. However, this is not always possible especially in rural places or due to unforeseeable incidents that bring down parts of the network. Hence, we would like to propose a solution that enhances FPS via Video Frame Interpolation so that users can download less content while watching videos with enjoyment.

Video Frame Interpolation (VFI) is the process of synthesizing images in between given a set of neighbor images. It is often used for tasks such as video compression or to create slow-motion effects. [9] Existing VFI methods can be classified into three categories, which are phase-based, flow-based, and kernel-based. These methods are mostly PSNR-oriented, which targets to minimize the L1 loss between the synthetic and the original frame. While capable of producing interpolated images with decent quality, they fail to capture style and perceptual differences [1].

Further, among all existing methods on VFI, none of them are based on diffusion models at the time of project proposal. It was later discovered that LDMVFI [2], published on March 2023, also uses latent diffusion models for the interpolation task. With diffusion models shown to outperform GANs and VAEs in synthetic tasks [5], we propose a novel solution using conditional latent diffusion models in tackling this problem. Our work would be similar to LDMVFI [2], with a simpler model architecture. Since they still have not open-sourced their code, our work could be viewed as a verification on the effectiveness of diffusion models in VFI. We expect diffusion models to be capable of generating high-quality intermediate frames conditioned on their neighbors.

In this work, we attempt to use latent diffusion models for video frame interpolation. We first use a VQGAN with perceptual loss to guide the encoding and decoding process. Then, we fix the encoders and the decoders for latent diffusion. The code is available at https://github.com/haydenych/comp5214-project. Although results are not comparable with SOTA models, we introduce a novel approach toward Video Frame Interpolation and also see its potential in this task.

2. Related Works

2.1. DDPM

Denoising Diffusion Probabilistic Models (DDPM) [4] consists of a forward and a backward process. The forward process adds Gaussian noise at each of the T steps, producing T noisy samples $x_1, \dots, x_T$ with decreasing clarity. It aims to transform a complex unknown data distribution into the standard Gaussian distribution, or any other prior known distribution. The backward process, on the contrary, tries to reverse this process. It generates a Gaussian noisy input and gradually removes noise at each of the T steps to form a realistic sample. Its goal is to learn the true distribution given some prior distribution. [12].

Formally, consider a sample from the true distribution $x_0 \sim q(x_0)$. The forward process is defined as:

$$q(x_t \mid x_{t-1}) = N(\sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t \cdot \mathbf{I}) \tag{1}$$

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{2}$$

where $\beta_1, \dots, \beta_T \in (0, 1)$ can either be hyperparameters or trained to represent the variance of the normal distributions.

If we re-parameterize β_t as $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{i=0}^{t} a_i$, we can show that $q(x_t|x_0)$ converges to a standard gaussian distribution as the number of steps t goes to infinity.

$$q(x_t \mid x_0) = N(\sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t \cdot \mathbf{I})) \xrightarrow{t\to\infty} N(0, \mathbf{I}) \quad (3)$$

As for the backward process, it can be defined as:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t) \quad (4)$$

$$p_\theta(x_{t-1} \mid x_t) = N(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \quad (5)$$

2.2. FLAVR

FLAVR [7] is a flow-agnostic approach. It is based on a U-Net style architecture, which first down-samples the video using 3D space-time convolutional blocks and then upsamples them back to the original input space. Their final prediction layer is implemented using a convolution layer that projects 3D feature maps into k frame predictions. This allows the model to perform single-shot multiple-frame predictions. Results also showed that FLAVR outperform other PSNR-based methods and is applicable to other downstream tasks, such as action recognition and optical flow estimation. However, this method is based on 3D CNNs while we would like explore the use of diffusion models due to its superiority in image synthetic tasks.

2.3. FREGAN

FREGAN [8] approaches VFI as an image synthetic task. Instead of using diffusion models, they used GANs with Huber loss. Results showed that they perform the best in terms of Structural Similarity Index (SSIM) and slightly worse in terms of Peak Signal-to-Noise Ratio (PSNR) than other state-of-the-art methods. We would like to further their work with diffusion models.

2.4. LDMVFI

Unlike FREGAN [8], LDMVFI [2] uses diffusion models instead of GANs. This work was published in March 2023 and it claims itself to be the first work using diffusion models for the VFI task. It uses a VQ-FIGAN for to encode images into latent embeddings and feed them into a latent diffusion model. The VQ-FIGAN uses feature pyramids from adjacent frames for cross-attention during the decoding process. Our approach would be similar to their works, except that we use VQGAN instead of a VQ-FIGAN.

3. Problem Formulation

Let x_t be the video frame at time step t. Our approach takes two input frames x_{t-k}, x_{t+k} to synthesize the intermediate frames $\{x_{t-k+1}, ..., x_t, ..., x_{t+k-1}\}$. Formally, let G be our proposed model and L be the objective function. Our goal is to compute $\{x_{t-\hat{k}+1}, ..., \hat{x_t}, ..., x_{t+\hat{k}-1}\}$ that minimizes the objective loss L.

$$\hat{x_z} = G(x_{t-k}, x_{t+k}) \text{ for } z \in (t-k, t+k) \cap \mathbb{N} \quad (6)$$

$$\hat{x_z}^* = \text{argmin}_{\hat{x_z}} \{ \sum_{z=t-k+1}^{t+k-1} L(x_z, \hat{x_z}) \} \quad (7)$$

However, due to the model complexity and resource limitations, we will only interpolate 1 intermediate frame.

4. Technical Methods

Our proposed method can be seperated into two parts. The first part consists of a VQGAN [3] that performs the encoding and decoding task. The encoder learns a transformation from the image space to a latent space while the decoder performs the opposite. The second part is our latent diffusion model. We fix the encoder and decoder learnt in the first step, and fit an image through the encoder to object a latent vector. This latent vector is then passed through the diffusion model for frame interpolation. Finally, the synthesized latent vector is decoded back to the image space.

4.1. VQGAN

The original implementation of VQGAN [3] is to synthesize high-quality images. It is based on VQVAE [11], which uses a discrete latent space instead of a continuous one. The authors of VQVAE [11] have also shown that vector quantization can alleviate "posterior collapse", a common issue of VAEs. Unlike VQVAE [11], VQGAN [3] uses perceptual loss (LPIPS) [6] and adds a discriminator to guide the image reconstruction process. It also consists of a transformer that learns plausible sequences from the latent codebook. However, we will focus on the autoencoder and will not use the transformer for this task.

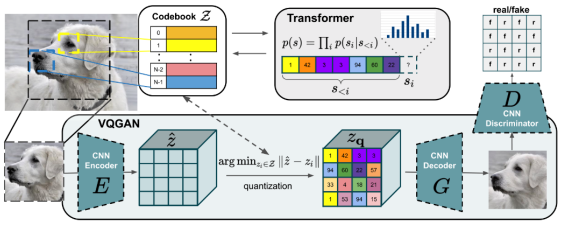


Figure 1. Model architecture of VQGAN

The authors have provided pre-trained weights on different latent and codebook size. We further finetuned the weights on our dataset using loss functions described in the original implementation of VQGAN. [3]. Simply speaking,

the full objective for our optimization model is,

$$Q^* = \text{argmin}_{E,G,Z} \max_D \mathbb{E}_{x\sim p(x)}[L_{VQ}(E,Q,Z) + \lambda L_{GAN}(\{E,G,Z\},D)], \tag{8}$$

where L_{VQ} is the vector quantization loss based on perceptual quality, L_{GAN} is the discriminator loss, and λ is an adoptive weight factor based on gradients [3].

4.2. Latent Diffusion

Latent Diffusion Models (LDM) [10] adds an encoder and a decoder on top of diffusion models. This converts the original high-dimensional input space into a much lower-dimensional latent space, significantly reducing model complexity and increasing inference speed.

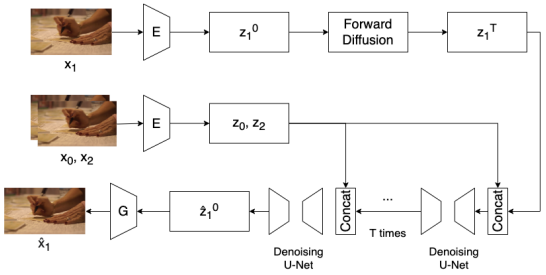


Figure 2. Model architecture of Conditional Latent Diffusion Model

After training the VQGAN, we fix the weights of the encoder E and decoder G. Figure 2 shows the model architecture of our conditional latent diffusion model. Frames 0, 1, and 2 (x_0, x_1, x_2) are fed through the encoder E to obtain their latent representations z_0, z_1, z_2. We denote z_1^i as the latent at step i of the forward diffusion. The forward diffusion process converts the input latent $z_1 = z_1^0$ gradually into noise z_1^T in T steps. During backward diffusion, we concatenate z_0, z_1^T, and z_2 and feed it into a denoising U-Net. The latent embeddings z_0 and z_2 serve as conditions to avoid the denoising process in generating arbitrary images. Finally, we obtain $\hat{z_1^0}$ from z_1^T, z_0 and z_2 and convert it back to the input space as $\hat{x_1}$ using the decoder G.

At inference stage, instead of forwarding z_1^0 into z_1^T, we sample $\hat{z}_1^T$ from $N(0,1)$ for backward diffusion. The complete algorithm for training and inference is shown in 1 and 2 [2] [4].

4.2.1 Denoising U-Net ϵ_θ

Figure 3 shows the U-Net model structure. It is formed by ResNet and attention blocks with small convolutional layers and pooling layers. The latent code of the previous and the next frames are concatenated with the sampled noisy latent code as the input of the U-Net, with time embeddings inserted at the ResNet blocks. The U-Net predicts the noise value added to form the noisy latent code at the next step and can be used to perform the reverse process.

Algorithm 1 Training

Input: Dataset $D = \{x_{i0}, x_{i1}, x_{i2}\}_{i=1}^N$ of 3 adjacent frames, Diffusion Steps T, and Variance Schedule $\{\beta_t\}_{t=1}^T$

Load: Pre-trained encoder E from VQGAN in Step 1

Initialize: Denoising U-Net ϵ_θ

While not converged **do**
Sample $(x_0, x_1, x_2) \sim D$
Encode $z_0 = E(x_0)$, $z_1 = E(x_1)$, $z_2 = E(x_2)$
Sample $t \sim \text{Unif}(1, T)$
Sample $\epsilon \sim N(0, I)$
Sample z_1^t based on ϵ, t, β_t
Take a gradient descent step on

$$\nabla_\theta \, ||\epsilon - \epsilon_\theta(z_0, z_1^t, z_2)||_2$$

Algorithm 2 Inference

Input: Dataset $D = \{x_{i0}, x_{i2}\}_{i=1}^N$ of 2 neighbour frames, Diffusion Steps T, and Variance Schedule $\{\beta_t\}_{t=1}^T$

Load: Pre-trained encoder E and decoder G from VQGAN in Step 1, Pre-trained Denoising U-Net $\epsilon\theta$ during training

Sample $\hat{z}_1^t \sim N(0, I)$
Encode $z_0 = E(x_0)$, $z_2 = E(x_2)$

For $t = T, \ldots, 1$ **do**
Predict noise $\hat{\epsilon} = \epsilon(z_0, z_1^t, z_2)$
Compute mean μ_θ and covariance Σ_θ from $\hat{\epsilon}$ and β_t
Sample $\hat{z}_1^{t-1}$ from $N(\mu_\theta, \Sigma_\theta)$

Return $\hat{x_1} = G(z_0, \hat{z_1^0}, z_2)$

5. Experimental Setup

5.1. Dataset

We will use Vimeo-90k for our task. Specifically, we will use the triplet dataset in Vimeo-90k, which consists of 3-frame sequences from 15k selected videos with fixed resolutions of 448 x 256. [13]. We use the train-test split provided by the authors, with 51313 3-frame sequences for training and 3782 for testing. Although Vimeo-90k also provides a dataset with 7-frame sequences, we did not use that due to resource limitations.

Further, we center-cropped the images into dimensions of 256 x 256 as it would be easier to handle square images

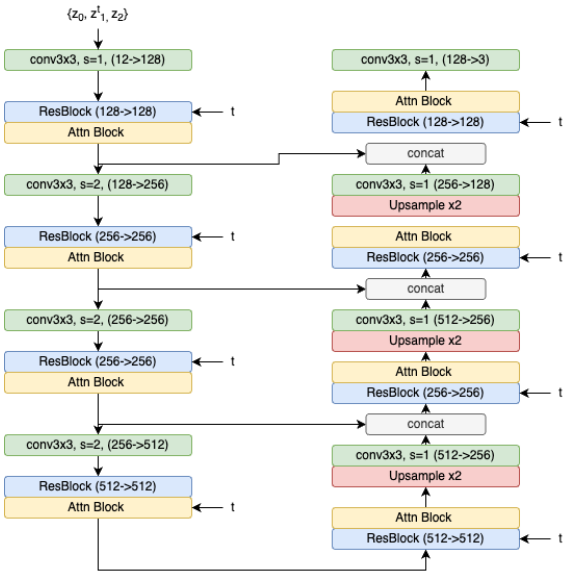


Figure 3. Model structure of the Denoising U-Net. In each block, the $(\cdot \to \cdot)$ indicates the input and output channels of the block.

and we do not want to distort the images through resizing. Apart from that, no augmentation has been performed due to the nature of this task. It is important to note that, when applying crops or flips to the frames, it has to be consistent across all frames in the same sequence.

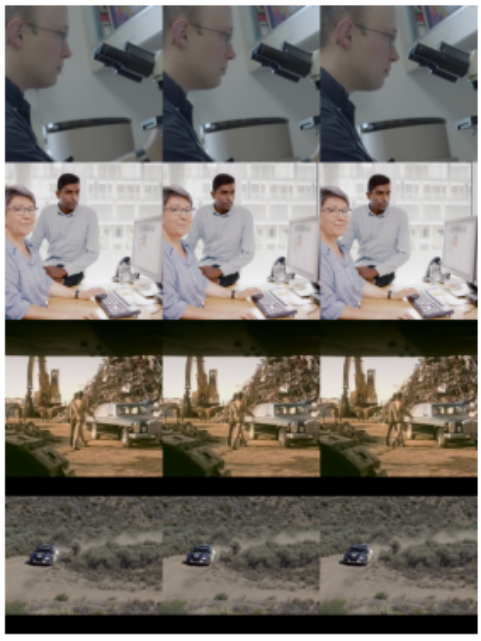


Figure 4. Sample frames from the VIMEO-90k dataset, from left to right are frames 0, 1, and 2

5.2. Configurations

For VQGAN, we have tested different downsampling factors (f) 8 and 16, and codebook size ($|Z|$) 256 and 16384. We further finetuned the pre-trained weights with 10 epochs and batch size 2 or 4 on our training dataset with a learning rate of 1e-7. NVIDIA RTX 2080Ti, 3080, and 3090 are used depending on availability from HKUST UG servers.

For Diffusion, we have tested with different latent code shapes and model structures. Two latent code shapes, $(256, 16, 16)$ and $(4, 32, 32)$, are formed by image $(3, 256, 256)$ through pre-trained VQGAN with downsampling factors 8 and 16, respectively. The number of U-Net layers is different for two latent shapes, for $(4, 32, 32)$ one, the U-Net structures follow Figure 3 while for $(256, 16, 16)$, the U-Net is one layer less compared with the other. The number of training time steps is 1000, and cosine learning rate schedule with 500 steps warmup is used.

Both models are trained with batch size 4 on our training dataset with 80 epochs with a learning rate of 1e-5 with AdamW optimizer. It took around 3 days to train a model to 80 epochs on 7x 3090 GPUs.

6. Experiments

6.1. Finetuning VQGAN

Since the interpolation process requires compacting the image space into the latent space, it is important to examine the effects of the encoders. The reconstruction losses are lossy and will be an upper bound of our interpolation task. If the encoders do not produce decent quality, the interpolated frames would be undesirable as well. Figure 5 illustrates some samples reconstructed by VQGAN, a clearer image is available at https://github.com/haydenych/comp5214-project/blob/main/samples/vqgan.png. The leftmost image is the original image, while the rest are reconstructed ones, with configurations shown in table 1 from top to bottom.

From the images, we see that the reconstruction results are desirable but with loss in some detailed features. In particular, it is common to see distorted facial features after reconstruction. In general, a smaller downsampling factor (f) and a larger codebook size ($|Z|$) produce better results. For the images reconstructed on fine-tuned weights, we observe some defects or noise produced at the edge of the images. We suspect that our implementation of the loss functions mentioned by the authors of [3] is incorrect, but we failed to fix it due to its complexity.

We further compute the FID scores on the reconstructed images. The table 1 shows the results. c refers to the number of channels at the latent space, and ft refers to fine-tuned or not. If they are fine-tuned, the configurations follow from section 5.2.

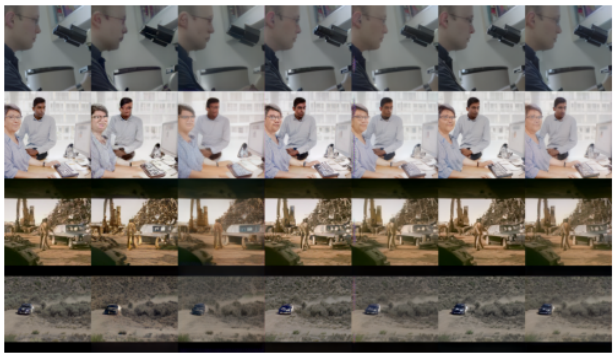


Figure 5. Sample images for the VQGAN reconstruction

f	$\|Z\|$	c	ft	FID on train	FID on val
16	16384	256	N	11.17	19.99
16	16384	256	Y	10.69	19.03
8	256	3	N	4.93	11.46
8	256	3	Y	9.02	15.33
8	16384	3	N	3.10	8.99
8	16384	3	Y	5.21	10.30

Table 1. FID scores on reconstructed images

We can observe that the FID scores of downsampling 16 times are much higher compared with downsampling 8 times, which shows loss and error is increased with the downsampling ratio. Then by comparing the FID scores before and after fine-tuning, the FID scores on the training and validation set increase after fine-tuning. This shows that fine-tuning does not improve the model, moreover, seems to overfit some images due to the small batch size training problem. However, since we do not have hardware that supports higher batch sizes, we cannot ensure whether pre-training with higher batch sizes can produce better results.

Since there is no improvement from fine-tuning, and there are defects in the reconstructed images on fine-tuned weights, we decide to use pre-trained weights on the subsequent interpolation task.

6.2. Video Frame Interpolation

We have tested the model with different downsampling sizes and different U-Net structures. Figure 7 shows the results of the two experiments.

First, we use the f16-16384 VQGAN auto-encoder and 3 layers for the denoising U-Net, the regenerated images are all black after 50 epochs. We observe that in Figure 6, the images in the middle column are all black. The interpolated latent code is mapped to black color in the auto-encoder. This implies that there is a huge difference between the interpolated latent code and the ground-truth latent code.

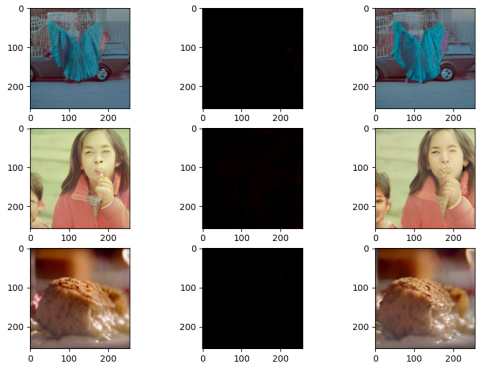
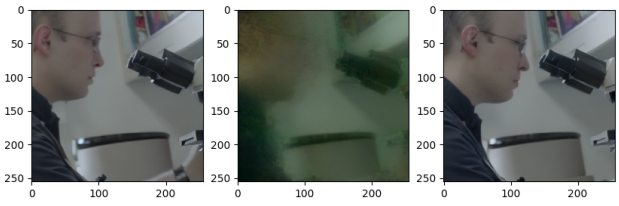
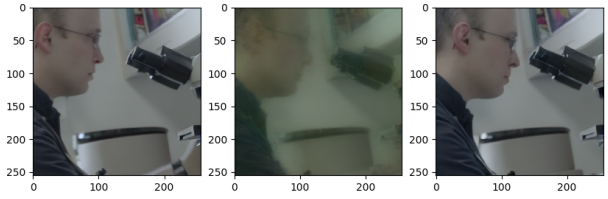


Figure 6. Results of using the f16-16384 VQGAN, middle is the interpolated image



(a) Results at epoch 1



(b) Results at epoch 80

Figure 7. Results of using f8-16384 VQGAN at different epochs, the middle one is the interpolated result

We then tried using the f8-16384 VQGAN autoencoder and 4 layers for the denoising U-Net, the regenerated images met our expectations. Figure 7a is the interpolated image at the first epoch, while Figure 7b is the interpolated image at epoch 80. We see an improvement in the training process, as the image gets sharper and cleaner after a few epochs. Figure 8 shows some additional results of our model. Compared with the images in the neighbor frames, we observe slight changes in motion and position, which meets our expectations. However, we can observe a pale-green layer on all interpolated images, possibly due to the random noise when synthesizing the latent code of the intermediate frame. Since the denoising U-Net we used is quite shallow, and the steps for gradient update is relatively small, these results are acceptable.

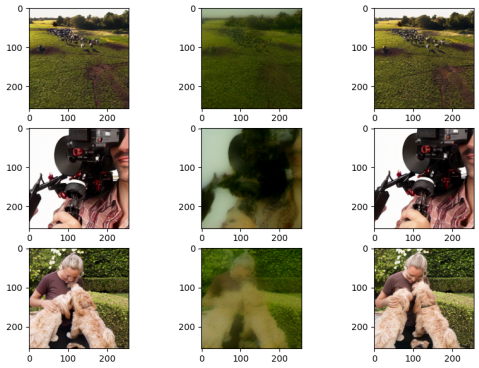


Figure 8. Results of using the f8-16384 VQGAN, the middle one is the interpolated image

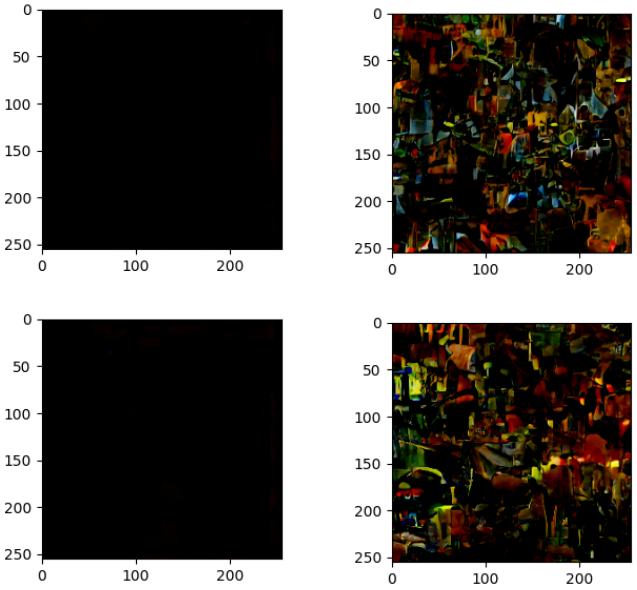
The interpolation time for 4 frames is around 20 seconds, which is limited by the GPU memory size. If more GPU memory is given, more frames can be interpolated together.

We further chose 200 images randomly to calculate the FID scores and obtained 48.1, which is high due to huge difference in color and sharpness. In comparison to SOTA results, this is undesirable but we expect to achieve better results if more time, training time, and computational resources are given.

Overall, our proposed model structure obtained preliminary success with certain configurations. With using f8-16384 VQGAN autoencoder and 4 layers of denoising U-Net, interpolation among two image frames can be achieved.

6.3. Effects of latent size on diffusion

We have used two different down-sampling auto-encoder sizes for the experiment, which form a $(256, 16, 16)$ and a $(4, 32, 32)$ latent shape. The one with a latent shape of $(256, 16, 16)$ was unsuccessful with our model structure. However, the latter ones works well. We then explore the effects of latent size by randomly sampling noise from $N(0, 1)$ and fitting it into the decoder. Figure 9a shows the f16-16384 Decoder produced fully black images while Figure 9b shows the f8-16384 Decoder generated colorful noisy images. This suggests the possibility that many parts of the latent space of the f16-16384 auto-encoder are mapped to black images, and its latent space is not fully utilized. Since the size $(256, 16, 16)$ is much larger, it requires more time for the learning process. This creates a larger difficulty for the denoising U-Net to learn its latent representations. Therefore, we believe using an auto-encoder that maps to a smaller latent space may perform better on our proposed model structure.



(a) Results of using f16-16384 VQGAN

(b) Results of using f8-16384 VQGAN

Figure 9. Results of using different autoEncoder

7. Discussions

We have proposed a model structure that combines VQ-GAN and DDPM for video frame interpolation. We tested on different model structures and obtained preliminary results with f8-16384 VQGAN pre-trained weight and 4 layers of denoising U-Net. This showed the potential of our model.

The generated image of our proposed model still requires huge improvement, given the interpolated results. For example, we may use a larger U-Net and a better-fine-tuned auto-encoder. However, this requires more computational resources and time to train the model. When training our Latent Diffusion Model, we referred to the training environment in the original paper of Latent Diffusion Models [10] and saw that they trained their models from scratch on A100 machines with a batch size of 64 and 2M gradient steps, while we only had a batch size of 4 and roughly 500k gradient steps. Further, the interpolation time of the proposed method is longer than we expected, which limits the usage of our method. Given 30 to 60 frames per second, our proposed method fails to perform real-time video frame interpolation with an inference time of roughly 5 seconds per frame. However, it can still be used in video preprocessing.

The future direction of our model would be enlarging the image resolution and the depth of the machine learning model. Moreover, different loss functions could be investigated to improve the similarity of the latent code and generated latent code.

8. Conclusion

In conclusion, we proposed a potential method for video frame interpolation, which achieves preliminary results under limited computational power. The proposed model combines the VQGAN and DDPM, which learns the latent code of the interpolated frames by latent diffusion. With the huge success seen from stable diffusion in generative tasks, we see a huge potential for using them in video frame interpolation.

References

[1] Duolikun Danier, Fan Zhang, and David Bull. A subjective quality study for video frame interpolation, 2022. 1

[2] Duolikun Danier, Fan Zhang, and David Bull. Ldmvfi: Video frame interpolation with latent diffusion models, 2023. 1, 2, 3

[3] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2021. 2, 3, 4

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. 1, 3

[5] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation, 2021. 1

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016. 2

[7] Tarun Kalluri, Deepak Pathak, Manmohan Chandraker, and Du Tran. Flavr: Flow-agnostic video representations for fast frame interpolation, 2022. 2

[8] Rishik Mishra, Neeraj Gupta, and Nitya Shukla. Fregan : an application of generative adversarial networks in enhancing the frame rate of videos, 2021. 2

[9] Fitsum Reda and Janne Kontkanen. Large motion frame interpolation, 2022. 1

[10] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 3, 6

[11] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018. 2

[12] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. 1

[13] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision (IJCV)*, 127(8):1106–1125, 2019. 3