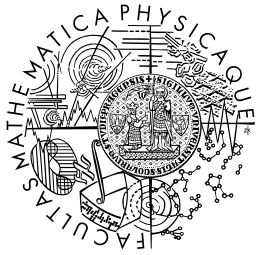

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Sviatoslav Gladkykh

Denoising Diffusion Models for Dynamic Sky Image Generation

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: doc. RNDr. Elena Šikudová, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2024

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I extend my gratitude to my supervisor, Elena Šikudová, for her invaluable guidance and detailed feedback throughout this work. Additionally, I would like to acknowledge Martin Mirbauer for his technical assistance and the Computer graphics group at Charles University for generously providing computational resources and access to the dataset essential for conducting this research.

Title: Denoising Diffusion Models for Dynamic Sky Image Generation

Author: Sviatoslav Gladkykh

Department: Department of Software and Computer Science Education

Supervisor: doc. RNDr. Elena Šikudová, Ph.D., Department of Software and Computer Science Education

Abstract: Recent advancements in the domain of generative AI demonstrate the exceptional performance of denoising diffusion models (DDMs), surpassing alternative models in image generation tasks. This work will be specifically oriented towards the in-depth investigation of the denoising diffusion models and their variants, including conditional, video, and random-mask video diffusion models. Utilizing historical data collected by people from Computer Graphics Group (CGG), containing 360° sky images, a series of experiments will be conducted to generate high-quality, optically faithful sky images. These experiments will not only involve the generation of static images but will also extend to generating and predicting sequences of images that can be compiled into a video format.

Keywords: Diffusion models Image generation Image prediction Video generation Video prediction

Contents

Introduction

Diffusion models are versatile and applicable across a wide range of tasks, including unconditional image generation (where images are generated from random noise), image-to-image translation (where the application receives an image and outputs another image, such as in super-resolution tasks), text-to-image synthesis (where the application receives text as input and generates an image based on the provided text), and many other tasks. This work will concentrate on unconditional image generation and image-to-image translation tasks. Additionally, these tasks will be extended to video scenarios, where a sequence will be considered instead of a single image. The outputs of my work can be useful across diverse fields, including weather forecasting, where accurate visualizations can increase prediction accuracy. Additionally, applications extend to renewable energy planning, agriculture, and environmental monitoring.

In the first part of the research, our focus will be on generating high-fidelity and visually realistic images of the sky utilizing denoising diffusion probabilistic models (DDPM) [1] and denoising diffusion implicit models (DDIM) [2]. The objective is to produce images indistinguishable from real ones to the human eye. Following this, we will modify the DDPM and DDIM models. These modifications will enable them to condition additional data, specifically the sky image. In this case, the model aims to try to predict the "next frame", showing how objects in the sky will evolve.

The second part will concentrate on generating entire videos as the outputs of the models. To achieve this, substantial modifications will be made to the architectures of the models to support additional time dimension. This will result in video diffusion models (VDM) [3] designed to unconditionally generate sequences of images that can be compiled into a video. Our aim remains to produce videos that are as realistic as possible. Next, we will switch to the random-mask video diffusion models (RaMViD) [4], which extend VDMs by introducing a novel conditioning technique during training. This technique allows the selection of a "mask" of frames we want to condition, offering significant capabilities for video prediction, infilling, and upsampling. However, our primary focus will be on the video prediction task, allowing us to observe the dynamic changes in the sky over time.

Chapter 1 introduces fundamental concepts of deep learning and denoising diffusion models, enabling readers with knowledge in the domains of computer science and mathematics to fully comprehend this work. In Chapter 2, several similar studies are presented with a brief description of their goals, approaches, datasets, and results. Details regarding the dataset used in this research are elaborated in Chapter 3. Experiments concerning static and conditional image generation are outlined in Chapter 4 and Chapter 5, respectively. The work on video generation is presented in Chapter 6, whereas the study regarding conditional video generation is addressed in Chapter 7. The final chapter, Chapter 8 provides a summary of our work and proposes further research and improvement ideas.

1. Background

1.1 Deep Neural Networks

This section introduces the fundamental concepts of deep neural networks, essential for understanding the material presented in this work. The information is presented concisely, with references provided to more detailed resources in case deeper explanations are needed. We are assuming knowledge in computer science and mathematics, while expertise in machine learning is not mandatory, could be beneficial. Readers with experience in deep learning may skip this chapter or skim through it briefly.

1.1.1 Perceptron

The perceptron [5], invented by Frank Rosenblatt in 1958, represents the foundational and simplest structure of modern neural networks. It serves as the algorithm for supervised learning of binary classifiers and consists of 3 main components: input vector $\boldsymbol{x} \in \mathbb{R}^n$, weight vector $\boldsymbol{w} \in \mathbb{R}^n$ and bias term $b \in \mathbb{R}$. The following expression computes the output value: $y = f(\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{w}_i + b)$, where f represents the activation function, which determines the output produced by perceptron. The perceptron schema is shown in Figure 1.1. In the binary classification context, where we categorize input data into one of two possible discrete classes, the activation function is defined as:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

referred to as the step function. Further investigation into activation functions can be found in the Subsection 1.1.3.

Training perceptron is a process of iterative adjustment of weights and bias to classify input data correctly. To simplify the notation of the training algorithm, we denote the input vector as $\boldsymbol{x} = (x_1, x_2, \ldots, x_n, 1)^\top$ and the weight vector as $\boldsymbol{w} = (w_1, w_2, \ldots, w_n, b)^\top$. This setup adds dimension, which plays a role of the bias term. Consequently, the output value can be computed as: $y = f(\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{w}_i) = f(\boldsymbol{x}^\top \boldsymbol{w})$.

Algorithm 1 Perceptron training algorithm

Input: Linearly separable dataset $(\boldsymbol{X} \in \mathbb{R}^{N \times D}, \boldsymbol{t} \in \{-1, 1\}^N)$.
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$
$\boldsymbol{w} \leftarrow 0$
while there exists a misclassified input vector **do**
  for $i = 1, \ldots n$ **do**
    $y \leftarrow \boldsymbol{x_i}^\top \boldsymbol{w}$
    if $t_i y \leq 0$ **then** ▷ Input vector classified incorrectly
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x_i}$ ▷ Update step of the weights and bias
    end if
  end for
end while

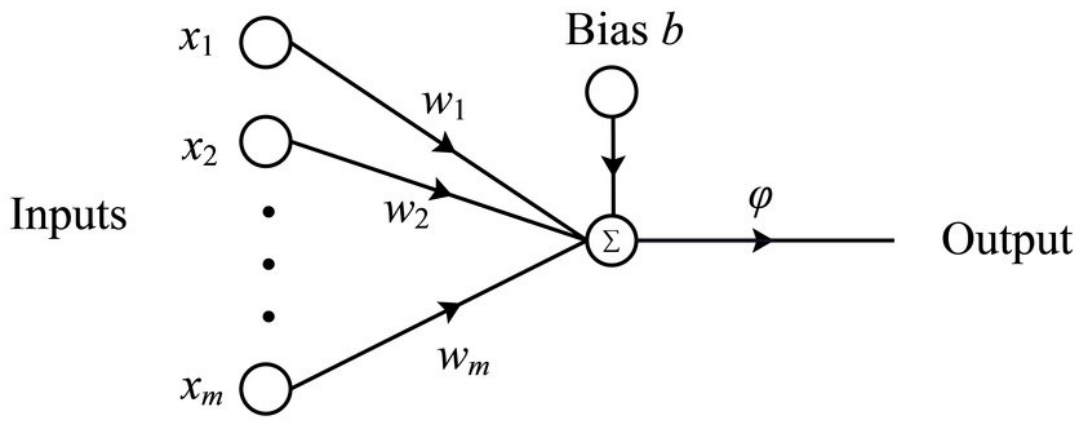

Figure 1.1: The perceptron schema [6]. It comprises inputs $x_1, x_2, \ldots, x_m$, corresponding weights $w_1, w_2, \ldots, w_m$, a bias term b and an activation function φ to generate output. Arrows directed toward the summation node denote the values to be summed, with the final arrow indicating the application of the step function to the sum.

1.1.2 Multilayer Perceptron

In 1986, David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams expanded upon the foundational concept of the perceptron by integrating hidden layers into the neural network and defining a novel optimization algorithm called backpropagation. The advancement, termed the multilayer perceptron [7], represents a modern iteration of feedforward artificial neural networks.

In the context of a single hidden layer configuration, the model comprises the following elements: input vector $\boldsymbol{x} \in \mathbb{R}^n$, weights matrix $\boldsymbol{W}^{(h)} \in \mathbb{R}^{n \times k}$ and bias vector $\boldsymbol{b}^{(h)} \in \mathbb{R}^k$ between the input and hidden layers, weights matrix $\boldsymbol{W}^{(y)} \in \mathbb{R}^{k \times m}$ and bias vector $\boldsymbol{b}^{(y)} \in \mathbb{R}^m$ between the hidden and output layers as well as two activation functions $f^{(h)}$, $f^{(y)}$ for hidden and output layers, respectively. Outputs of the model can be computed in two steps: $\boldsymbol{h} = f^{(h)}(\boldsymbol{x}^\top \boldsymbol{W}^{(h)} + \boldsymbol{b}^{(h)})$, $\boldsymbol{h} \in \mathbb{R}^k$ and then $\boldsymbol{y} = f^{(y)}(\boldsymbol{h}^\top \boldsymbol{W}^{(y)} + \boldsymbol{b}^{(y)})$, $\boldsymbol{y} \in \mathbb{R}^m$. The schema of a similar network is shown in Figure 1.2. The next two subsections provide information regarding the selection of the activation functions and details concerning the training procedure Subsection 1.1.4.

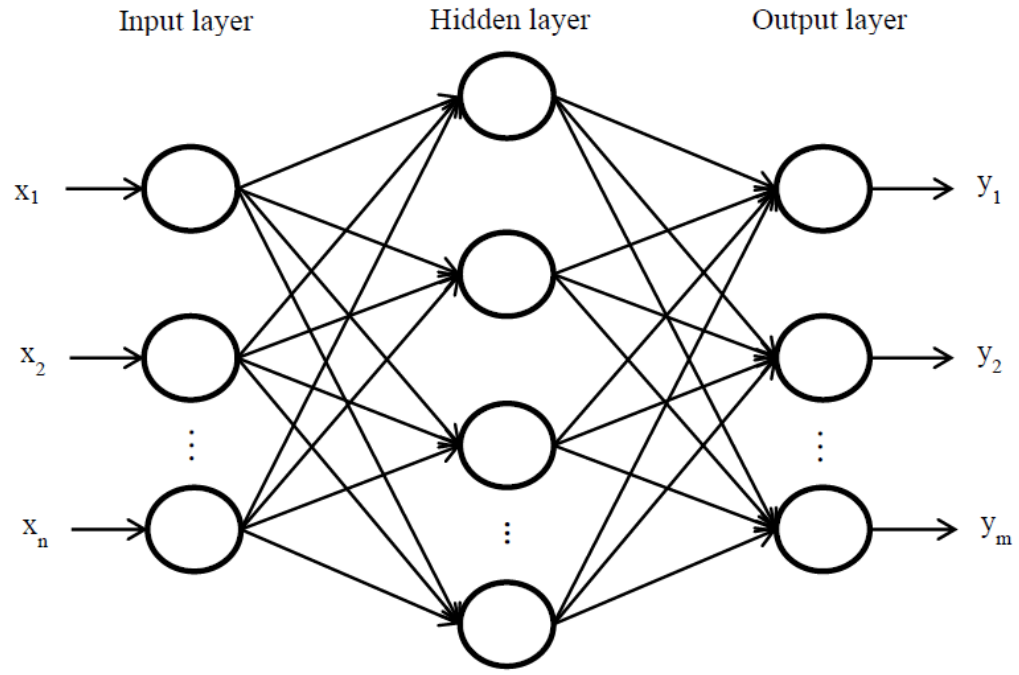

Figure 1.2: The multilayer perceptron schema [8]. Neural network in the figure consists of the input layer with values $x_1, \ldots, x_n$, a hidden layer, and an output layer with values $y_1, \ldots, y_m$.

This type of neural network is commonly called fully connected because it consists of fully connected layers. Every input neuron (node) is linked to every output in a fully connected layer through assigned weights. Having more than one hidden layer in the multilayer perceptron is typically unnecessary. This is attributed to the Universal approximation theorem [9], which states that a feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function to arbitrary accuracy, given the appropriate activation function and a sufficient number of neurons.

1.1.3 Activation functions

The selection of the activation function depends on the nature of the problem and desired properties of the model.

Among the most prevalent choices for the hidden layer are the Hyperbolic Tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, the Rectified Linear Unit $\mathrm{ReLU}(x) = max(0, x)$, and various modifications such as Leaky ReLU, Swish, Parametric ReLU, and Exponential Linear Unit. Although it is technically possible to omit an activation function in the hidden layer, doing so is not practical as it disallows the multilayer perceptron to capture nonlinear relationships in the data [10].

The activation function used in the output layer determines the output format generated by the neural network. For regression tasks, it is customary not to use an explicit activation function, while for binary classification, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is often chosen. In multiclass classification scenarios the softmax function is commonly preferred $\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

1.1.4 Training Neural Networks

To train a neural network, it is essential to define a loss function first. The loss function is a measure of how well the network models the training data. During the training process, the objective is to minimize this loss between predicted values and real targets. A common principle to design loss functions is the maximum likelihood principle. For regression tasks, the loss is a mean squared error (MSE) [11], defined as:

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)})^2$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^N$ represents a vector of predicted values, and $\boldsymbol{y} \in \mathbb{R}^N$ denotes a vector of real targets. For binary classification tasks, the loss function is commonly referred to as binary cross-entropy or log loss [11], given by:

$$L(\boldsymbol{p}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} -(\boldsymbol{y}^{(i)} \cdot \log \boldsymbol{p}^{(i)} + (1 - \boldsymbol{y}^{(i)}) \cdot \log (1 - \boldsymbol{p}^{(i)}))$$

where $\boldsymbol{p} \in \mathbb{R}^N$ represents a vector of probabilities belonging to the first class as predicted by the model, $\boldsymbol{y} \in \mathbb{R}^N$ is a vector of actual targets. In multiclass classification, a common choice for the loss function is categorical cross-entropy [11], defined as:

$$L(\boldsymbol{P}, \boldsymbol{Y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} \boldsymbol{Y}_{ij} \cdot \log \boldsymbol{P}_{ij}$$

where $\boldsymbol{Y} \in \mathbb{R}^{n \times m}$ represents a one-hot matrix of targets where $\boldsymbol{Y}_{ij}$ is 1 if sample i has target value j, and 0 otherwise. Additionally, $\boldsymbol{P} \in \mathbb{R}^{n \times m}$ is a matrix where $\boldsymbol{P}_{ij}$ represents a probability of sample i belonging to class j outputted by the model.

The basic optimization algorithm for training neural networks and various other models is gradient descent [11]. Assuming the objective is to minimize a loss function $\arg\min_\theta L(\hat{\boldsymbol{y}}, \boldsymbol{y})$, where $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ represents a model parametrized by $\boldsymbol{\theta}$, the parameter update step during training is expressed as follows:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y})$$

where α is a learning rate, which specifies the magnitude of each step we perform in every iteration of the gradient descent. Several variants of the gradient descent exist [11]:

- Standard gradient descent: all training data is used to compute the gradient.
- Stochastic gradient descent (SGD): gradient is estimated by the single sample in the training data.
- Minibatch SGD: gradient is estimated by m random independent samples in the training data, where m is a batch size. This approach is widely favored due to its balanced trade-off between the standard and stochastic gradient descent methods.

Gradient descent converges to the unique optimum (global minimum) under the condition that the loss function is convex, continuous, and the sequence of learning rates α_i satisfies the following properties: $\sum_i \alpha_i = \infty$, $\sum_i \alpha_i^2 < \infty$.

The training procedure procedure involves two main phases: forward propagation and backward propagation (backpropagation). Forward propagation is the computation of the outputs of the neural network given inputs. The computational formulas previously described in the Subsection 1.1.2. Subsequently, the loss function value is computed based on the predictions and actual targets. In contrast, backward propagation calculates the gradient of the loss function with respect to the weights and biases of the model. This gradient is then propagated through the network by applying the chain rule. Following the computation, the parameters are adjusted in the opposite direction of the gradient to minimize the loss function. Both forward and backward propagation are iteratively performed for multiple epochs during training, where one epoch denotes a single pass through the entire dataset.

The update step of the parameters is typically done using an optimization algorithm. Apart from the well-known SGD, there exist several smarter algorithms:

- SGD with momentum [12]: introduces a concept of inertia to the parameters update by considering not only the current gradient but also the accumulated past gradients.

$$\begin{aligned}\boldsymbol{g} &\leftarrow \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y})\\ \boldsymbol{v} &\leftarrow \beta\boldsymbol{v} - \alpha\boldsymbol{g}\\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v}\end{aligned}$$

  An adaptation of this method called SGD with Nesterov momentum [13] considers the gradient evaluated at the "look-ahead" position, determined by adding the momentum-scaled previous update vector to the current parameters.

- Adagrad [14]: adaptive gradient algorithm, allowing larger updates for infrequent parameters and smaller updates for frequent ones.

$$\begin{aligned}\boldsymbol{g} &\leftarrow \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y})\\ \boldsymbol{r} &\leftarrow \boldsymbol{r} - \alpha\boldsymbol{g}^2\\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \varepsilon}\boldsymbol{v}\end{aligned}$$

- RMSProp [15]: root mean square propagation, which computes and uses a decaying average of squared gradients

$$\begin{aligned}\boldsymbol{g} &\leftarrow \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y})\\ \boldsymbol{r} &\leftarrow \beta\boldsymbol{r} + (1-\beta)\boldsymbol{g}^2\\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \varepsilon}\boldsymbol{v}\end{aligned}$$

- Adam [16]: adaptive moment estimation can be looked at as a combination of RMSProp and SGD with momentum for estimating the first and second moments of the gradient.

$$
\begin{aligned}
&\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) \\
&t \leftarrow t + 1 \\
&\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g} \\
&\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g}^2 \\
&\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \beta_1^t}, \hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \beta_2^t} \\
&\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\hat{\boldsymbol{r}}} + \varepsilon} \hat{\boldsymbol{s}}
\end{aligned}
$$

  An adaptation of this method, AdamW [17], directly incorporates weight decay into the optimization process.

Using different optimizers instead of the usual SGD has various advantages, including faster convergence, better handling of sparse gradients, robustness to saddle points, and others. Among the listed algorithms Adam and AdamW are the most popular due to their adaptive learning rate mechanisms and momentum-based updates. They are widely used and demonstrate good performance across various tasks and datasets.

1.1.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) [18] represent a class of deep neural networks typically applied to computer vision tasks, including image classification, semantic segmentation, object detection, and more. A key innovation of CNNs is their ability to successfully capture the spatial and temporal dependencies within an image by applying relevant filters. These filters detect low-level patterns, such as edges or textures in early layers, gradually progressing to learn more abstract and complex features in deeper layers.

Convolution Operation

The cornerstone of CNNs' exceptional performance in computer vision tasks is attributed to the convolution operation. This operation involves sliding a filter, also known as the kernel, over the input data, such as the image, and computing the element-wise multiplication between the filter and the overlapping region of the input data. Mathematically, we employ cross-correlation instead of the convolution operation, where cross-correlation can be interpreted as convolution with a mirrored kernel. This process is formally expressed as follows:

$$
(\boldsymbol{K} \star \boldsymbol{I})_{i,j,o} = \sum_{m,n,c} \boldsymbol{I}_{i+m,j+n,c} \boldsymbol{K}_{m,n,c,o}
$$

where $\boldsymbol{K}$ denotes the kernel, typically represented as a small matrix for sliding convolution over the input data, and $\boldsymbol{I}$ represents the input image. It is customary to specify the desired output dimensionality of the convolutional layer. This is accomplished by defining the number of output channels o for every pixel.

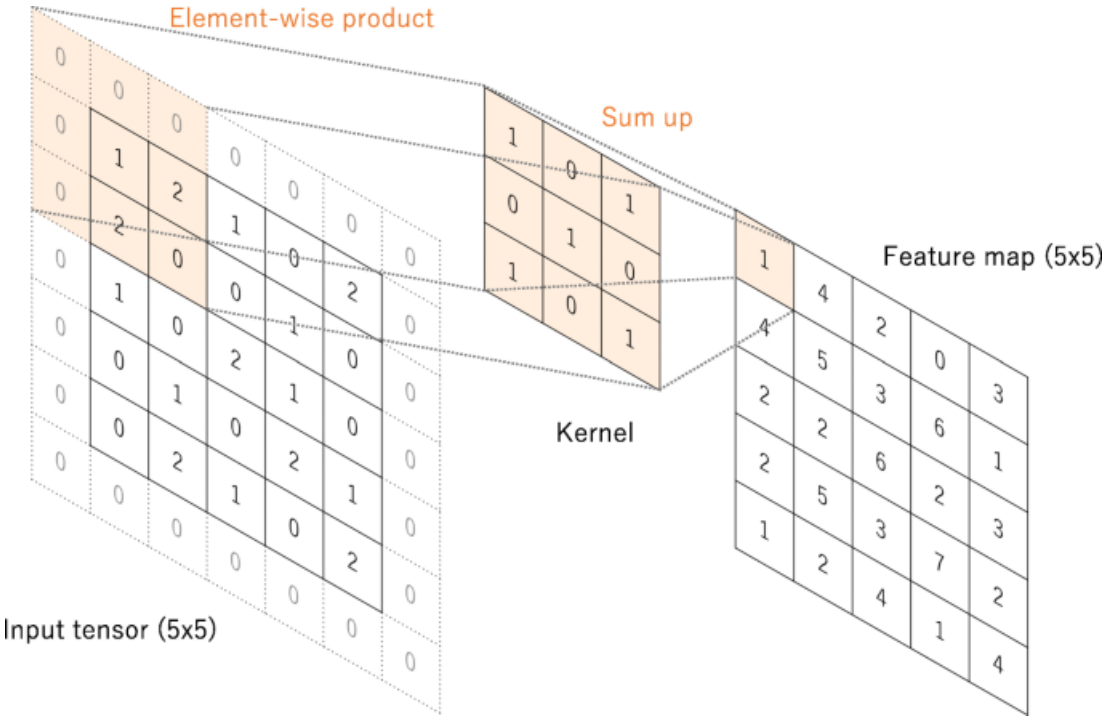

Figure 1.3: An example of convolution operation [19] with input image 5×5, kernel size 3×3, a stride of 1 and padding of 1 as well. The padding of 1 ensures that the output feature map size remains the same as the input image size for a 3×3 kernel.

The convolutional layer can be finely tuned by adjusting several parameters, including the number of output channels, kernel size, strides (number of pixels the kernel shifts over the image), padding around the image, activation function, and some other less significant parameters. A demonstration of the convolution operation with these parameters is shown in Figure 1.3. Single convolutional layer often needs more strength for the most common tasks. Therefore, practitioners often combine convolutional layers in diverse configurations, designing various convolutional neural network architectures tailored to different tasks.

Pooling

Polling [18] is an operation similar to convolution, but instead of computing the dot product between the input image and kernel, a fixed operation is performed. Typically, two types of pooling operations (layers) are used:

- Max pooling: this can be viewed as a sliding window of a certain size that extracts the maximum value from each window.
- Average pooling: similarly, this operation involves sliding a window over the input and calculating the average value within each window.

Pooling layers are employed to effectively and explicitly reduce the spatial dimensions of the feature maps. Their purpose can be intuitively explained as combining the information from several small regions into fewer but larger regions, thereby facilitating learning about larger portions of the initial image.

Batch Normalization

When feeding data into a neural network, it is a standard practice to normalize the data to have zero mean and unit variance. This practice ensures that all input features have the same scale and, as a result, helps the optimizer in updating the parameters more smoothly. This is done in the following way:

$$\boldsymbol{x}_i = \frac{\boldsymbol{x}_i - \mu}{\sigma}$$

Here, μ represents the mean value of the feature, and σ denotes the standard deviation. In situations where computing these parameters over the whole dataset is impractical due to high dimensionality or a large number of sample samples, batch normalization [20] is employed. It estimates μ and σ on the batch currently being processed by the network, and applies a scaling coefficient and an offset to recover the lost degrees of freedom:

$$BN(\boldsymbol{x}) = \hat{\boldsymbol{\gamma}} \odot \frac{\boldsymbol{x} - \hat{\boldsymbol{\mu}}}{\hat{\boldsymbol{\sigma}}} + \hat{\boldsymbol{\beta}}$$

Here, $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ represent the sample mean and sample standard deviation across the batch, while $\hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\beta}}$ are trainable parameters. Batch normalization is typically applied after the linear transformation but before the non-linear activation function. Normalization is performed for each dimension (channel) separately.

Residual Connections

Residual connections [21], also known as skip connections, are fundamental components in various convolutional neural network architectures. They were designed to address the "vanishing gradient" problem, which occurs in very deep neural networks where gradients diminish significantly as they propagate backward through many layers during training. Residual connections mitigate this issue by introducing a direct connection from earlier layers to later ones, bypassing one or more intermediate layers. Mathematically, this connection is represented by the sum of an earlier layer's output and the current layer's output. An illustration of such a connection is depicted in Figure 1.4.

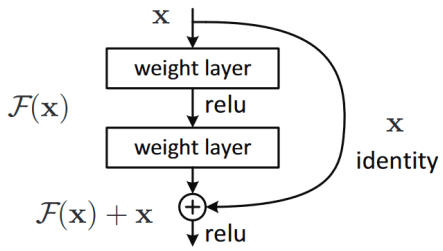

Figure 1.4: The residual connection schema [22]. Input X is summed with the output of two subsequent layers. Note that the activation is applied after the sum operation.

U-Net

U-Net [23] is the convolutional neural network architecture primarily designed for semantic segmentation. Over time, it has found various applications, particularly in denoising diffusion models. The name "U-Net" comes from its U-shaped architecture, which comprises a contracting path (encoder) followed by an expansive path (decoder).

The encoder is composed of several convolutional layers followed by max-pooling layers, which downsample the feature maps, thereby reducing the spatial dimensions while increasing the receptive field. Conversely, the decoder contains up-convolutional layers (transposed convolutions) and skip connections.

In U-Net, skip connections are achieved by concatenating feature maps rather than summing them. Nonetheless, the objective remains similar: to preserve spatial information lost in the contracting path and to mitigate the "vanishing gradient" problem. The illustration of the architecture is shown in Figure 1.5

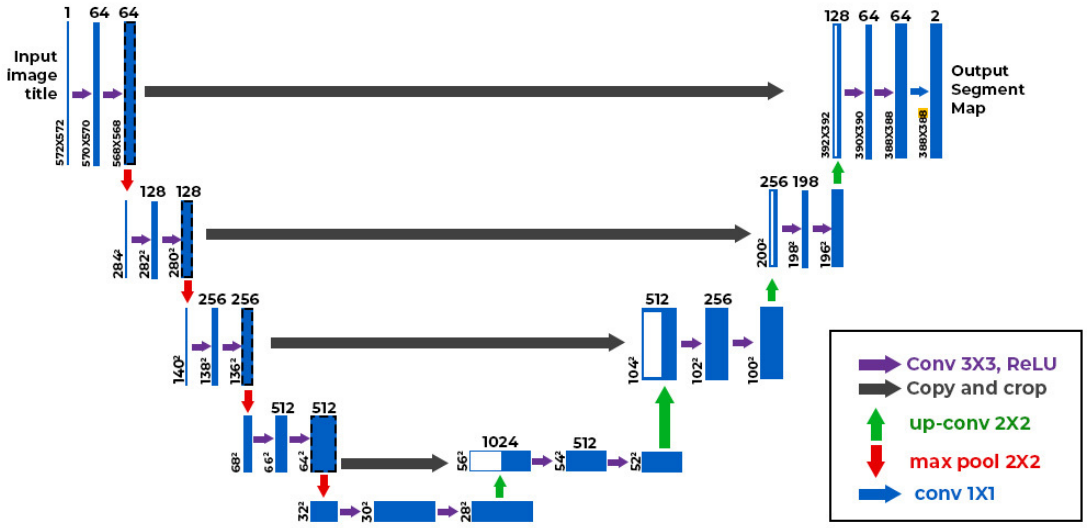

Figure 1.5: The U-Net architecture figure [24]. It consists of 4 encoder (downsampling) blocks, a middle part (bottleneck) and four decoder (upsampling) blocks with skip connections to the corresponding downsampling blocks. The output size differs from the input size because no padding was utilized in the convolutional layers.

1.1.6 Self-attention

Self-attention [25] is a mechanism primarily employed in natural language processing. It was introduced in the transformer architecture for sequence-to-sequence tasks, enabling models to capture relationships between the elements of the given sequence, such as words.

The concept behind self-attention is that each input element of the sequence undergoes three transformations, which can be intuitively interpreted as follows:

- Query $\boldsymbol{Q}$ denotes the data the element seeks among the others.
- Key $\boldsymbol{K}$ represents the information that the element can offer to the others.

- Value $\boldsymbol{V}$ is the actual information or meaning associated with each element of the sequence.

Assuming $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, $\boldsymbol{Q} \in \mathbb{R}^{n \times d_k}$, $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$ the self-attention can be computed as follows:

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

Where $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$, $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$ and $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, $\boldsymbol{W}^V$ are trainable weight matrices.

In simpler terms, the process involves combining queries with keys $\boldsymbol{Q}\boldsymbol{K}^\top$ to produce a matrix of "relevances" for each query and key. Dividing by $\sqrt{d_k}$ ensures that the variance does not grow as the dimensionality increases. Then, the softmax function is applied to obtain a probability distribution. Finally, a weighted combination of values is obtained using the corresponding probabilities.

Self-attention has found applications in computer vision tasks, particularly in scenarios where capturing global context is crucial. In the context of images, self-attention considers the grid of features as a sequence of feature vectors, and trainable weight matrices are implemented as 1×1 convolutions applied to each vector, allowing pixels to attend to each other. The visualization of such self-attention is shown in Figure 1.6.

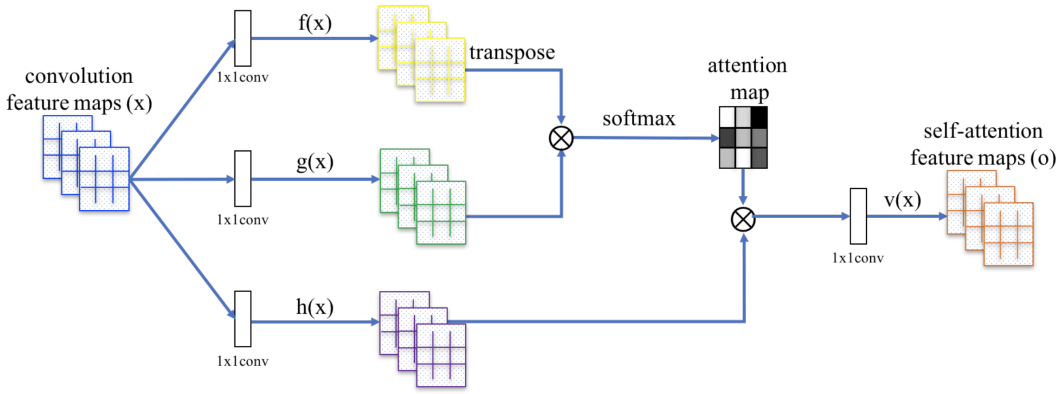

Figure 1.6: The visualization of self-attention mechanism applied to convolution feature maps [26]. The $\otimes$ denotes matrix multiplication. The softmax operation is performed on each row.

1.1.7 Positional Embeddings

Positional embedding is another technique derived from natural language processing. Transformers need to gain the ability to understand the order of elements in the input sequence because they process these elements in parallel. Therefore, positional embeddings were introduced to provide the transformer with positional information.

Various methods exist to represent positional information. However, the most common approach involves using sinusoidal functions [25]. The positional embeddings are calculated based on the position of the token within the sequence and

its dimension within the embedding vector, expressed as follows:

$$PE_{pos,2i} = sin(\frac{pos}{10000^{\frac{2i}{d}}}) \quad \text{and} \quad PE_{pos,2i+1} = cos(\frac{pos}{10000^{\frac{2i}{d}}})$$

Here, $PE_{pos,2i}$ denotes the positional embedding for the i-th dimension of the token at position pos, and d is the dimensionality of the model's input embeddings.

1.2 Denoising Diffusion Models

Denoising diffusion models [1] belongs to a class of generative models and are designed to describe a probabilistic process for generating high-quality images from noisy inputs. They operate by gradually introducing Gaussian noise to the original data and subsequently learning to remove it until a clear image is obtained.

1.2.1 Forward Diffusion Process

Given a data point $\boldsymbol{x}_0$ from real data distribution $q(\boldsymbol{x})$, we define a T-step forward diffusion process that gradually adds Gaussian noise to the data according to some variance schedule $\beta_1, \ldots, \beta_T$:

$$\begin{aligned} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) &= \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}) \\ &= \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon} \\ q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) &= \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \end{aligned}$$

As more noise is gradually added to the original image $\boldsymbol{x}_0$, the image converges closer to pure Gaussian noise. The schema of the forward diffusion process is shown in Figure 1.7.

Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, then using the reparametrization trick we can show that:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\boldsymbol{\epsilon}}_{\mathbf{0}}$$

for standard normal random variable $\bar{\boldsymbol{\epsilon}}_{\mathbf{0}}$. In other words, this reparametrization demonstrates that $\boldsymbol{x}_t \sim \boldsymbol{q}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I})$, enabling us to sample $\boldsymbol{x}_t$ at any arbitrary time step from this distribution.

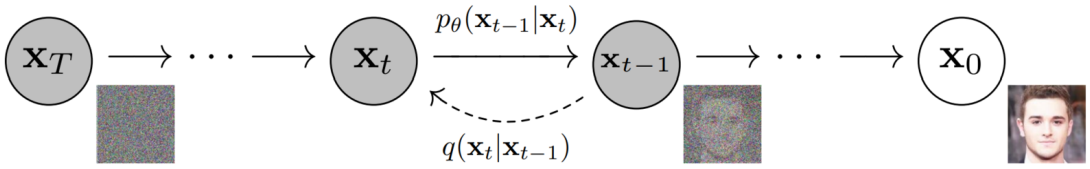

Figure 1.7: A schema illustrating the forward $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ and reverse $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ diffusion processes with T steps [1]. Here, x_0 represents the original image, and x_T is the initial image after T steps of adding noise.

1.2.2 Noise Schedule

The previous section mentioned the variance schedule $\beta_1, \ldots, \beta_T$, also known as the noise schedule. Initially, the linearly increasing sequence from 0.0001 to 0.04 was utilized. However, this approach encountered issues as the second part of the corresponding $\bar{\alpha}_t$ sequence contained minimal values. Therefore, during the diffusion process, the second half was dominated by random noise, decreasing its effectiveness. To mitigate this problem, the cosine schedule [27] was introduced:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \; f(t) = \cos\left(\frac{t/t+s}{1+s} \cdot \frac{\pi}{2}\right)^2, \; s = 0.008$$

1.2.3 Reverse diffusion process

The reverse diffusion process is essentially the inverse of the forward process. To approximate the reverse of $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ we have to learn a model $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$. When β_t is relatively small, the reverse closely resembles a Gaussian distribution. Hence, we can represent p_θ as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \sigma_t^2 \boldsymbol{I})$$

where $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. This formulation involves taking the current image, computing a denoised image using $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$, and reintroducing some noise $\sigma_t^2 \boldsymbol{I}$. The schema of the reverse diffusion process is shown in Figure 1.7.

With this setup, the entire reverse process can start with $\boldsymbol{x}_T$ and gradually remove noise until a fully denoised image is reached. Mathematically, this can be expressed as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$$

However, instead of predicting the denoised image $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ directly, we aim to predict the noise added to the image $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$. With this reparametrization, the denoised image is computed as follows:

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \right)$$

where $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ represents the model's prediction of the noise.

1.2.4 Loss function

The loss for training diffusion models can be derived by minimizing negative log-likelihood and applying Jensen's Inequality. Before reparametrization, the loss for the model is defined as:

$$\begin{aligned}
&- \mathbb{E}_{q(\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) \right] \\
&\leq \mathbb{E}_{q(\boldsymbol{x}_{0:T})} \Bigg[D_{KL}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) || p_{\boldsymbol{\theta}}(\boldsymbol{x}_T)) \\
&\quad + \sum_{t=2}^{T} D_{KL}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) || p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x})) \\
&\quad - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \Bigg]
\end{aligned}$$

Here, $D_{KL}(q(\boldsymbol{x}_T|\boldsymbol{x}_0)||p_{\boldsymbol{\theta}}(\boldsymbol{x}_T))$ is constant and can be ignored, $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)$ can be used to generate discrete $\boldsymbol{x}_0$ from continuous $\boldsymbol{x}_1$ (for now, can be ignored as well) and $\sum_{t=2}^{T} D_{KL}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)||p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}))$ is a KL divergence between two Gaussians, therefore, can be computed explicitly as:

$$L_t = \mathbb{E} \left[\frac{1}{2||\sigma_t \boldsymbol{I}||^2} ||\tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t, \boldsymbol{x}_0) - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2 \right]$$

After the reparametrization, the loss function slightly changes to:

$$L_t = \mathbb{E}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha})||\sigma_t \boldsymbol{I}||^2}||\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)||^2\right]$$

It was found that training without the weighting term performs better, so the final loss is:

$$L_t = \mathbb{E}_{t\in\{1..T\},\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}\left[||\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)||^2\right]$$

1.2.5 Training and sampling

During the training of the diffusion model, we randomly sample an image x_0 from the training set, along with a timestamp t and random noise $\boldsymbol{\epsilon}$. Next, noise is added to the image, and the model predicts the added noise. Then, the loss between the actual and predicted noise is computed, and a gradient step is taken in the direction that minimizes the loss function. This process is repeated iteratively.

Algorithm 2 Training

repeat
  $x_0 \sim q(x_0)$
  $t \sim Uniform(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  Take gradient descent step on
    $\nabla_\theta||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)||^2$
until converged

The sampling process starts from a random noise x_T and subsequently undergoes T iterations of denoising. Within each iteration, the noise added to the image is estimated. This estimated noise is then subtracted from the image with appropriate scaling. Furthermore, additional noise is introduced based on the variance determined for that iteration.

Algorithm 3 Sampling

$x_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
for $t = T, \dots 1$ **do**
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ if $t > 0$, else $\boldsymbol{z} = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(x_t, t)\right) + \sigma_t \boldsymbol{z}$
end for
return x_0

2. Related Work

In this chapter, we explore various studies that parallel the focus of our work. Through this analysis, we explore various methodologies researchers implement, identifying promising model configurations and data preprocessing techniques. Subsequently, in later chapters, we will compare our findings with their benchmarks, thereby assessing our outcomes.

In the SkyGAN [28] research, the problem of generating photorealistic environment maps for rendering virtual scenes was solved. Although the researchers aimed to generate images with predefined sun position and cloud coverage ratio, the same generative adversarial network (GAN) could have been employed for unconditional sky image generation. Furthermore, the dataset utilized in this research is identical to the one we will use but with fewer samples. In their study, the dataset comprised 39,000 HDR fisheye photographs of clouds. The quality of the generative model was evaluated using the Fréchet inception distance (FID), achieving a minimum score of 14.6.

Another similar research [29] aimed to forecast sky images. In this study, participants used the generative adversarial network (GAN) with modifications to allow the model to predict future frames of a sky image sequence. Two approaches to image prediction were explored: one where the model receives four frames and outputs the next one, and another model receives eight sequential images as input and produces the next eight frames. The training dataset consisted of 15,000 photos captured using a digital camera equipped with a fisheye lens mounted in front of the camera's entrance optics. A cloud coverage metric was proposed to assess the quality of generated sky images. Remarkably, the average deviation of the cloud coverage estimation is around 10% at the end of the generated sequence is extremely close to the ground truth.

A distinct study [30] was conducted to improve solar irradiance forecasting by utilizing score-based diffusion models. These models were used to sample high-resolution images to estimate uncertainty in forecasts. The research employed two models: the first model generated 64×64 samples, while the second performed super-resolution to produce 128×128 images. Both diffusion models were conditioned on the same coarse-resolution atmospheric variables output by numerical simulations. The researchers utilized three main data sources for training and evaluating the experiments: GOES satellite data, GFS forecasts, ERA5 reanalysis data. They resulted in a dataset of 51,830 training data and 22,807 test data. The results of the study were successful: the produced images were realistic, the samples were diverse, and the mean of the samples served as a good point estimate for numerical weather prediction.

3. Dataset

For training models, we utilized a dataset provided by the Computer Graphics Group at Charles University, Faculty of Mathematics and Physics. This dataset contains 18,840 HDR skydome photos with 1024×1024 resolution. The photos were captured using a full-frame camera sensor with an 8 mm circular fisheye lens aimed upward toward the sky. The data was collected from various locations in Central Europe and coastal California, offering a diverse range of sky conditions, from clear skies to completely clouded ones, from early morning to late night. Notably, the dataset is organized sequentially, with photos taken at a frequency of one image every 30 seconds, preserving the temporal sequence within the dataset.

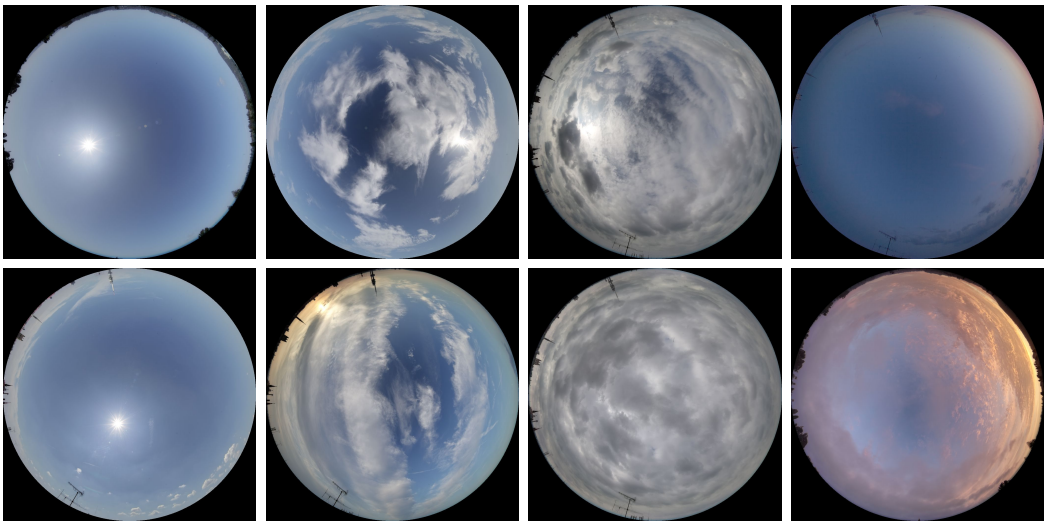

Figure 3.1: Examples of photos from the dataset with clear, partially clouded, heavily clouded, and evening sky conditions.

4. Unconditional Image Generation

This chapter introduces two denoising diffusion models designed for unconditional image generation, along with their corresponding neural network architectures. Additionally, we discuss our experimental setups, present the experiments conducted, and provide the results obtained on the dataset mentioned in Chapter 3.

4.1 Denoising Diffusion Probabilistic Models

The theoretical background regarding diffusion models described in Section 1.2 is the foundation used in denoising diffusion probabilistic models [1]. To train the model and sample images, defining a neural network capable of performing the required task is necessary.

The DDPMs model the noise prediction $\boldsymbol{\epsilon_\theta}(\boldsymbol{x}_t, t)$ using the U-Net architecture. This architecture employs pre-activated residual blocks at each resolution of the model. A residual block comprises batch normalization, activation, and 3×3 convolution, repeated twice, with the residual connection overlaying them. This architecture is highly favored for diffusion models due to numerous advantages, such as the encoder-decoder structure, multi-resolution feature maps, skip connections, etc.

To inform the model about the current time step, transformer sinusoidal embeddings will be added within the middle of every residual block. This addition occurs after the first convolution but before the subsequent batch normalization.

To capture global information and enhance the consistency of the resulting image, convolutional self-attention blocks can be added on several low-resolution levels immediately following the residual blocks. They are applied at lower levels due to this operation's relatively high computational cost.

4.2 Denoising Diffusion Implicit Models

Denoising diffusion implicit models [2] were introduced to accelerate the sampling procedure, addressing the primary weakness of the DDPMs. The fundamental concept is to redefine the forward process from probabilistic to deterministic. The new one can be described as follows:

$$q_0(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) = q_0(\boldsymbol{x}_T|\boldsymbol{x}_0) \prod_{t=2}^{T} q_0(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$$

where

$$q_0(\boldsymbol{x}_T|\boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_T}\boldsymbol{x}_0, (1-\bar{\alpha}_T)\boldsymbol{I}\right)$$

and for all $t > 1$,

$$q_0(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\left(\frac{\boldsymbol{x}_t - \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0}{\sqrt{1-\bar{\alpha}_t}}\right), \boldsymbol{0}\right)$$

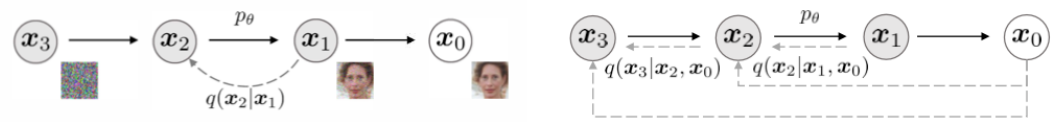

Figure 4.1: Graphical models for diffusion (left) and non-Markovian (right) inference models [2].

It can be proved by induction that $\boldsymbol{q}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I})$, which indicates that the "marginals" are the same as in DDPMs. Therefore, the training algorithm and the loss function remain unchanged.

The actual forward process can be expressed using the Bayes theorem:

$$q_0(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0) = \frac{q_0(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)q_0(\boldsymbol{x}_t|\boldsymbol{x}_0)}{q_0(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}$$

The reverse process can be expressed using the definition of $q_0(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ in the following way:

$$\begin{aligned}\boldsymbol{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x_t}, \boldsymbol{t}) \\ &= \sqrt{\bar{\alpha}_{t-1}}\left(\frac{\boldsymbol{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x_t}, \boldsymbol{t})}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_t}\bar{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x_t}, \boldsymbol{t})\end{aligned}$$

Finally, $\boldsymbol{q}_0$ has a crucial property that allows performing several steps at once:

$$q_0(\boldsymbol{x}_{t'}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t'}}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_{t'}}\left(\frac{\boldsymbol{x}_t - \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0}{\sqrt{1-\bar{\alpha}_t}}\right), \mathbf{0}\right)$$

4.3 Results

We trained multiple models with varying hyperparameters across different image resolutions. However, due to computational constraints and slow sampling, we tuned and evaluated models primarily with 128×128 image size.

The entire dataset, consisting of $18,840$ images, was used for training without any significant preprocessing steps apart from resizing and normalization.

Our final diffusion model has the following architecture and parameters:

- Number of steps in the diffusion process was set to $T = 1000$ for the probabilistic model and $T = 50$ for the implicit model. Increasing linear variance schedule from $\beta_1 = 10^{-4}$ $\beta_T = 2{*}10^{-2}$ was employed for the forward process.

- For the reverse process, we utilized a slightly modified 2D U-Net model with 6 downsampling and upsampling blocks with the number of output channels set to $128, 128, 256, 256, 512$, and 512, respectively. Each U-Net block comprised 2 ResNet layers. Transformer sinusoidal embeddings for timestamp representation were added in the middle of every residual block. Convolutional self-attention was applied only in the second last downsampling block and second upsampling block.

- AdamW optimizer mentioned in 1.1.4 was chosen with the learning rate $\alpha = 10^{-4}$. Additionally, we employed a cosine learning rate schedule (annealing) with 500 warm-up steps (where the learning rate gradually increases from zero to its base value).

- The size of the batch was set to 16. We trained for 1000 epochs (approximately 1.2M steps). However, after 350k steps (300 epochs), there was no noticeable improvement in visual image quality and metrics.

For model evaluation, we utilized a Fréchet inception distance (FID) metric [31], a widely used measure for assessing the quality of generated images. FID compares the distribution of generated images with the distribution of natural images. The comparison is done by computing the means and standard deviations of feature maps extracted from Inception v3 [32] model trained on ImageNet dataset[33].

To compute Fréchet inception distance, we generated 10k images after 300 epochs. The resulting FID was 5.43, which significantly outperforms the SkyGAN [28] model minimum score of 14.6. This represents a 2.69-fold reduction in FID score. The chart illustrating FID scores against the number of generated images is presented in Figure 4.2, while Examples of images generated by our network can be seen in Figure 4.3.

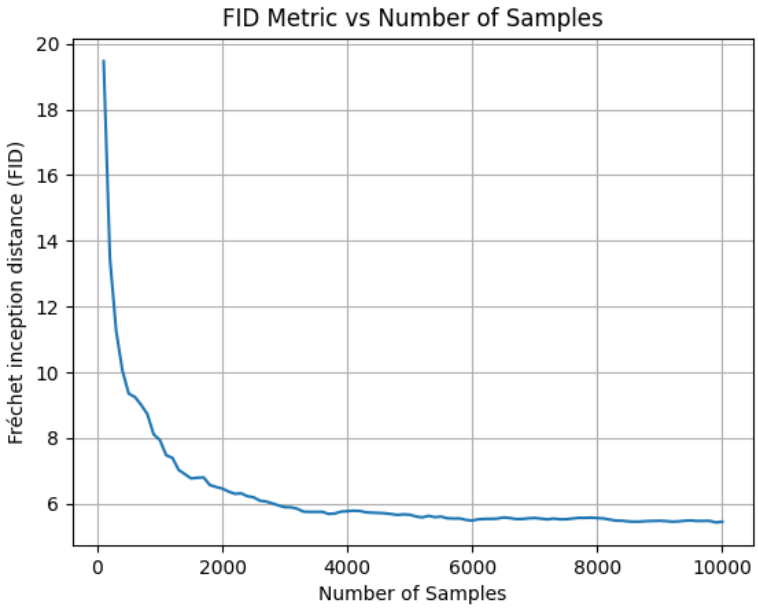

Figure 4.2: FID scores versus the number of sampled images used for computation

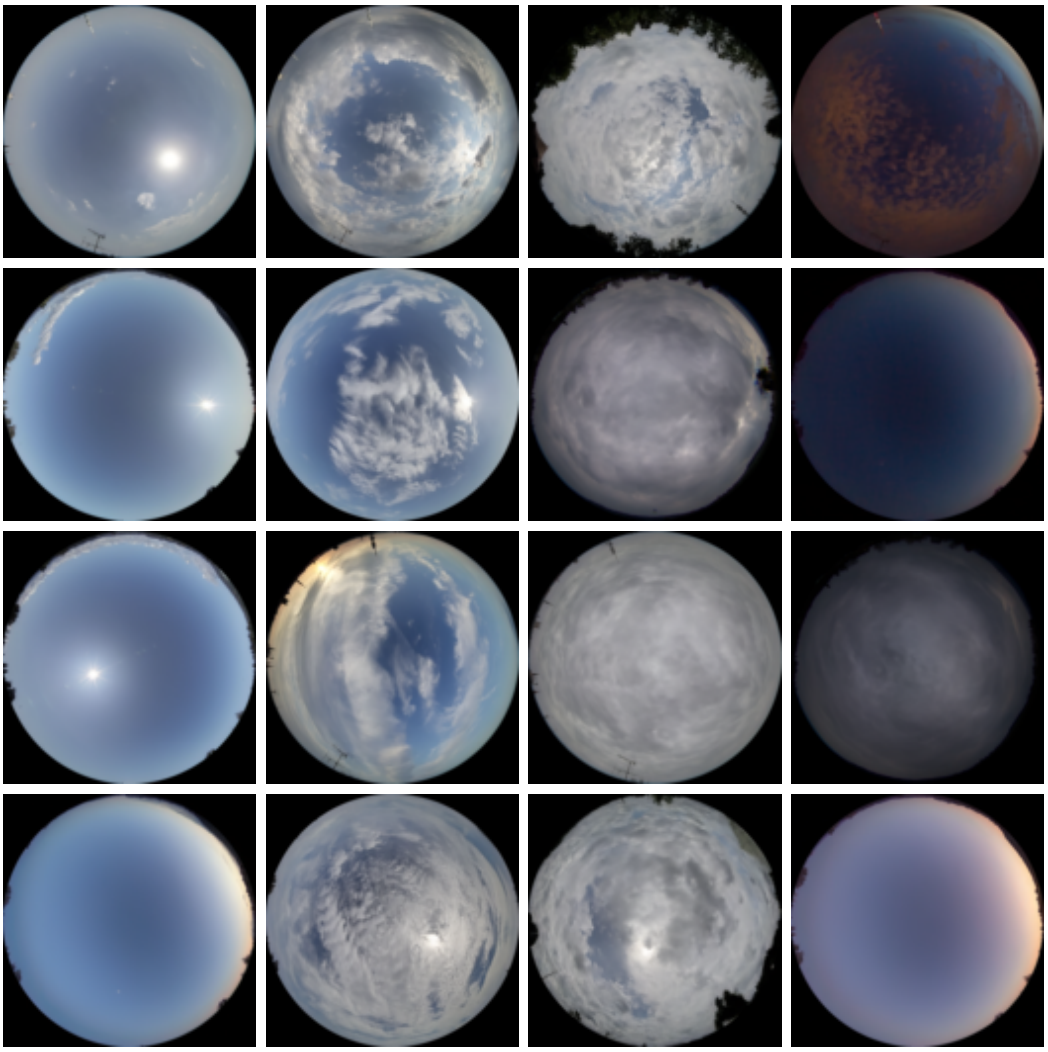

Figure 4.3: Samples of unconditionally generated sky images. Each column contains four images with clear, partially clouded, heavily clouded, and evening sky conditions respectively.

5. Conditional Image Generation

This chapter delves into the denoising diffusion approaches for conditional image generation, describing various conditioning techniques. Following an exploration of different conditioning methodologies, we proceed to present the outcomes of our experiments conducted with these models in the "next frame" prediction task.

5.1 Conditioning in Denoising Diffusion Models

There are scenarios where it is advantageous for the generative model to be conditional. For instance, in Section 4.1, we have described how the diffusion models are conditioned on the current timestamp. In addition to timestamp conditioning, using other data may be helpful. Mathematically, this involves modeling the distribution $p(\boldsymbol{x}|\boldsymbol{y})$, where $\boldsymbol{x}$ is a target image and $\boldsymbol{y}$ is a condition image.

Conditioning on images is a common practice and will also be utilized in our work. It finds utility in various scenarios, including super-resolution (the model is conditioned on a low-resolution image and generates a high-resolution one), inpainting (filling in missing or damaged parts of an image), colorization (converting grayscale images to color while preserving semantic content), and numerous other image-to-image tasks. Conditioning can be done in many ways, such as concatenating noisy images with the conditioning image, employing more complex feature fusion techniques, incorporating attention mechanisms, and others. These methods can be applied alone or in combination, depending on the architecture and specific requirements of the denoising diffusion model. However, the concatenation will be used in our experiments for simplicity and effectiveness.

Utilizing text conditioning is another widely used approach, particularly in text-to-image tasks, where the goal is to generate an image based on textual descriptions or captions. This form of conditioning consists of two stages: first, encoding the text using some pre-trained encoder such as BERT or GPT, and then employing an additional attention layer, typically located after self-attention layers. However, the attention mechanism operates between the image and text representations in this case.

5.2 Dataset Preparation

It was crucial to ensure that the images were organized sequentially to prepare the dataset for the "next frame" prediction task. Given that the time interval between two consecutive images is 30 seconds and assuming there are no significant changes in the sky during this period, the following algorithm was employed for verification:

1. For each non-empty directory containing images, collect the names of the image files.

2. Sort the names of the image files based on the numerical substring (e.g., "IMG_2830.jpg" $\rightarrow$ 2830).

3. Determine the most common difference in numbers between two subsequent images.

4. Split the sequences at the indices where the difference is greater than the most common difference.

5. Calculate the structural similarity (SSIM) between consecutive images in every sequence.

6. Analyze for anomalies by plotting the metric values or automatically detecting extreme deviations from average values.

After implementing and executing the algorithm as mentioned earlier, we have confirmed that the images in the dataset indeed maintain a sequential order based on the folder and numerical substring within the filename. From the examples of four SSIM plots of randomly selected sequences shown in Figure 5.1, it is evident that either the images exhibit almost 100% similarity consistently, or the SSIM value gradually changes in correspondence with weather variations. Sharp and significant drops in SSIM values were not observed within the sequences.

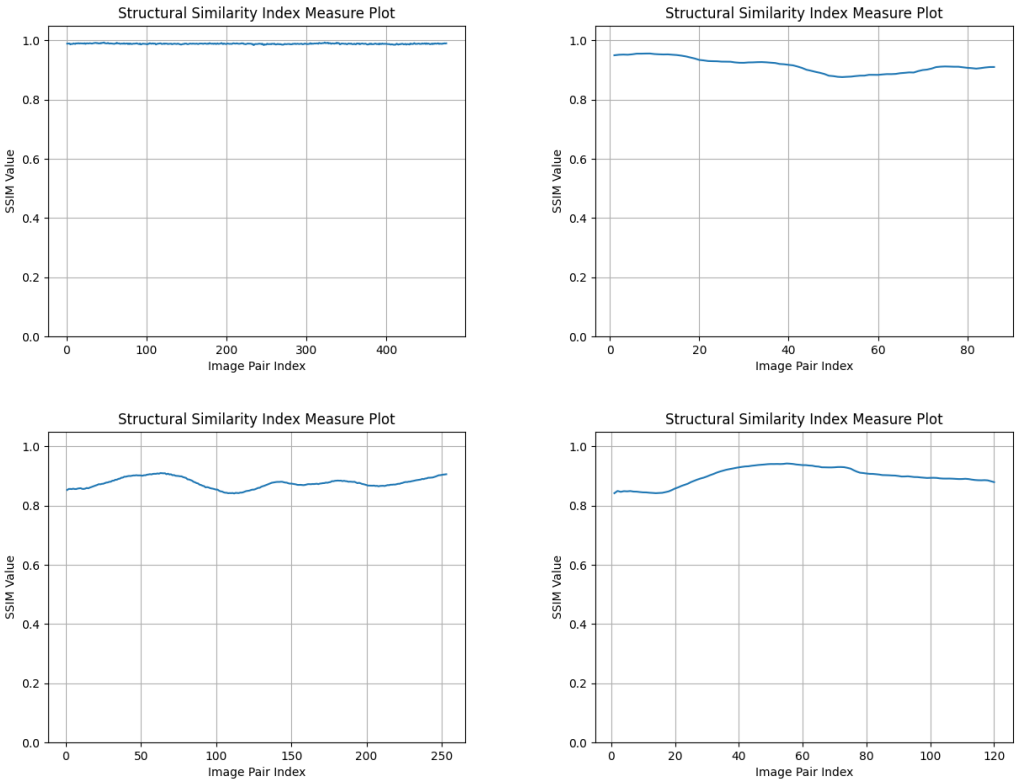

Figure 5.1: Four plots of the structural similarity index measure (SSIM) between two consecutive images within randomly selected dataset sequences.

To improve the flexibility of our dataset, we introduced two parameters:

- Offset between images - this parameter allows specifying the time interval between frames. For example, setting the offset to 1 corresponds to a time interval of 30 seconds, while an offset of 2 corresponds to 60 seconds, and so on.
- Length of the image sequence - this parameter allows the creation of sequences of images of the desired length by controlling the number of images included in each sequence.

5.3 Results

For model training, we utilized the dataset with an offset between images set to 1. This decision was made because a larger offset led to more significant changes in the sky between two consecutive frames, making it more challenging for the model to learn patterns. The image sequence length was set to 5, comprising 4 conditioning images and one we aim to predict. The performance was much worse while we experimented with conditioning on 1 and 2 previous frames. Moreover, 4 frames proved a successful choice in the sky forecasting work [29] with GANs. Additionally, we kept 100 dataset instances untouched during training to evaluate performance on them separately. In the end, the resulting dataset comprised $18,424$ samples.

Regarding the model, we decided to stick with the most successful one for unconditional image generation, as described in detail in Section 4.3. However, we applied the following changes to support conditioning frames:

- Increased the number of model's channels from 3 to $3 \cdot (\text{sequence length} - 1)$
- Conditioning was achieved by concatenating images. Therefore, we concatenated conditioning frames to the current noisy image during each training or sampling step.

To assess the quality of predicted images, we will employ the following measures:

- Structural similarity index measure (SSIM): quantifies the perceptual difference between two similar images.
- Peak signal-to-noise ratio (PSNR): measures the ratio between the maximum possible power of an image and the power of corrupting noise that affects the quality of its representation.
- Mean squared error (MSE) - the actual loss between two images, as described in Subsection 1.1.4.
- Cloud coverage [29] - quantifies the ratio of sky pixels, which are clouds, out of the total sky pixels. To maintain consistency with the original work, we adopted identical thresholds for cloud selection, given the high similarity between our datasets.

Since our task involves predicting sequences rather than individual frames, a direct comparison between predicted "next frame" images and actual ones may not be as informative. Instead, our focus lies on more long-term predictions. Therefore, we sequentially generated a sequence of 20 frames, where the first 4 images were given, and evaluated metrics on these.

From the results depicted in Figure 5.2, we can see that all measures deteriorate as we generate frames. Specifically, the last 20th generated frame exhibits an average SSIM of 0.74, PSNR of 20.9, and MSE of 0.15. However, these metrics remain relatively reasonable, considering they are 16 frames beyond conditioning. Moreover, the average deviation of the cloud coverage estimation on the 8th generated image is around 5.5%, considerably lower than the 10% deviation reported using GAN models [29]. Even our 20th image has an average deviation of approximately 8.1%. Examples of original and generated sequences are shown in Figure 5.3.

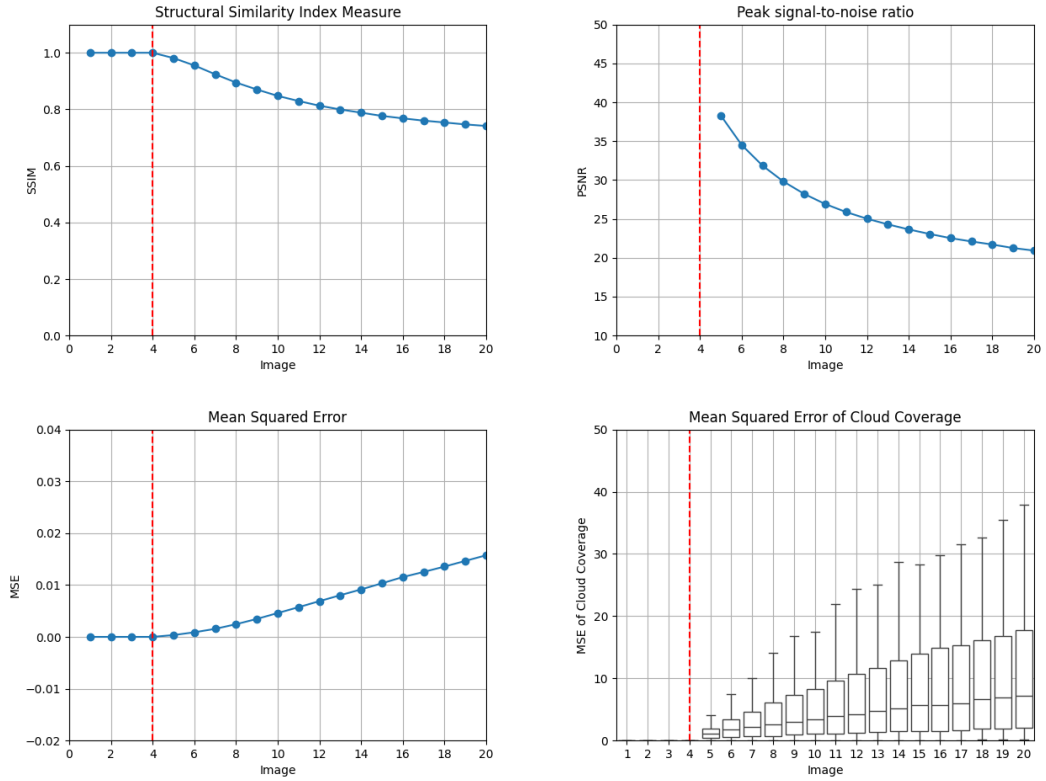

Figure 5.2: Four charts with SSIM, PSNR, MSE, and cloud coverage MSE measures are shown. The vertical axis corresponds to the score, while the horizontal axis corresponds to the generated frame by our model. The red dotted line shows where the conditioning frames end.

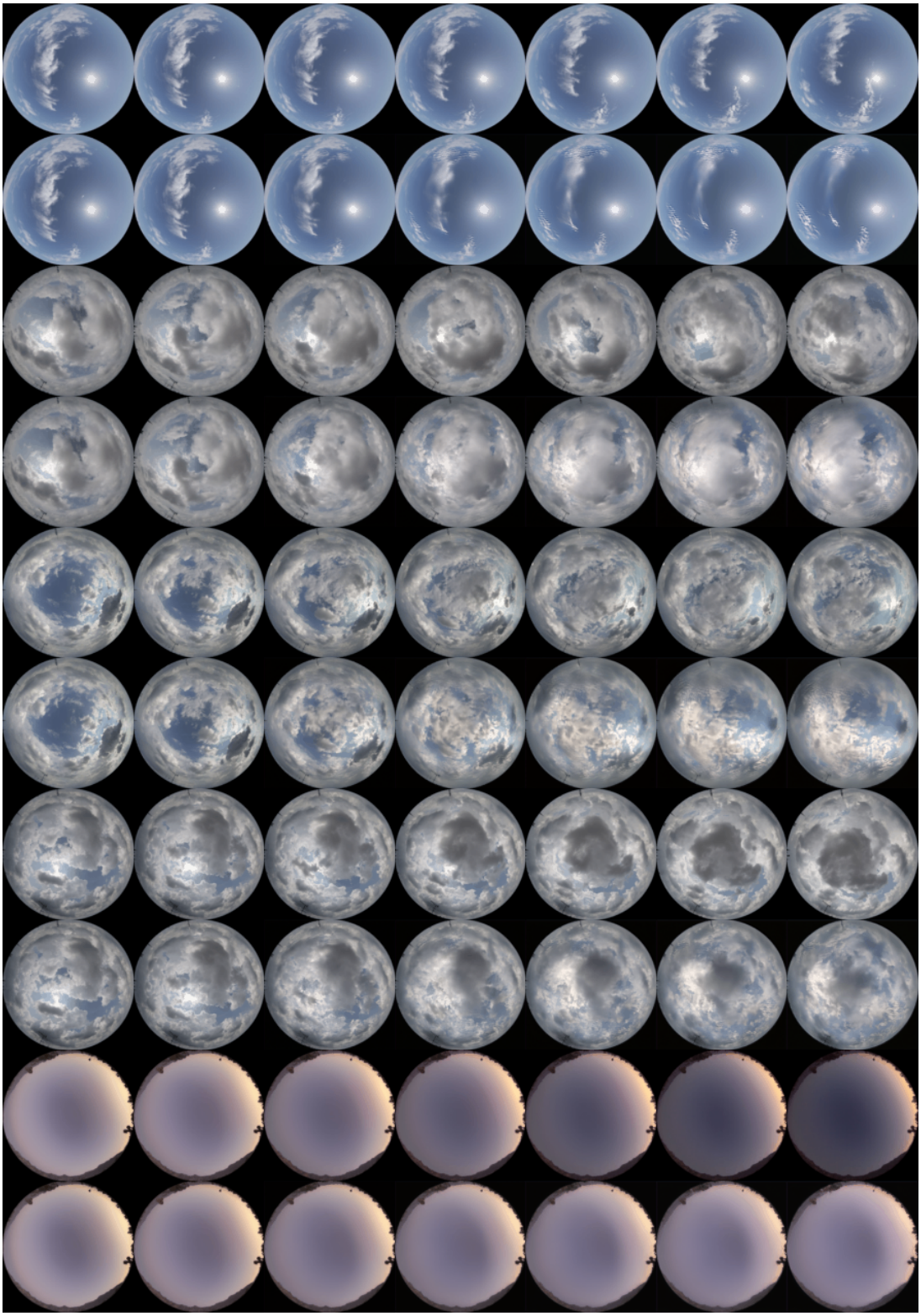

Figure 5.3: 10 rows of images, each comprising 7 frames extracted from a 20-frame image sequence: Specifically, frames 1, 4, 7, 10, 13, 16, and 19. Odd rows depict frames extracted from the original videos. Even rows represent the frames extracted from predicted videos. Note that the first two images remain consistent across each row, as they serve as conditioning images for our model.

6. Video Generation

This chapter introduces video diffusion models as a framework for unconditional video generation. These models naturally extend standard image diffusion architecture for unconditional video generation. Furthermore, the chapter presents the results obtained from generating short videos of the sky and evaluates the model's performance.

6.1 Video Diffusion Model

The video diffusion models [3] leverage the standard denoising diffusion framework described in Section 1.2. However, it adopts the 3D extension of the U-Net [34] architecture shown in Figure 6.1 with the following modifications:

- Transition from 2D Convolution to space-only 3D Convolution (e.g. 3×3 kernel becomes $1 \times 3 \times 3$)
- The attention in each spatial block remains as attention over space. Additionally, a temporal attention block is performed after each spatial attention block.
- In each temporal block, relative position embeddings [35] are used to discern the sequential order of frames.

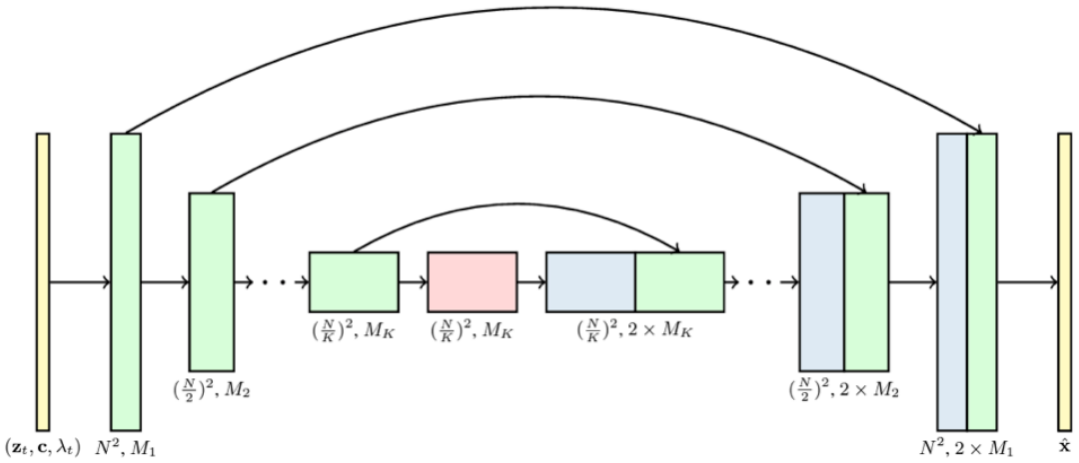

Figure 6.1: Schema of 3D U-Net [3] architecture for video diffusion models. Each block represents a 4D tensor processed in a space-time factorized manner. Here K is the number of upsampling/downsampling blocks, and M_i is a channel multiplier for each block.

A technique of reconstruction-guided sampling was described in video diffusion mode ls paper [3] to improve conditional generation. This approach enables the generation of videos with any length, given that the model generates a limited number of frames. Instead of separately sampling from unconditional and conditional models, sampling is conducted from a single diffusion model that has been jointly trained.

6.2 Dataset Preparation

The dataset preparation for the video diffusion models closely resembles the approach used for conditional image generation in Section 5.2. However, in this case, it is essential to avoid including samples with repeating images in the dataset. For instance, a sequence generated from samples 1-20 and another from samples 2-21 repeat frames 2-20. Eliminating such sequences is essential as they do not significantly enhance the model's performance but consume additional memory and training time.

To incorporate this functionality into the dataset, we introduce a new parameter that allows control of the number of images allowed to be repeated in the next sequence within the dataset.

6.3 Results

We decided to choose a number of frames in a video $N = 20$ with the resolution 64×64. The offset between images was set to 1, implying that each subsequent frame represents a 30-second interval, resulting in a 10-minute video. Furthermore, we adjusted a dataset parameter as mentioned in Section 6.2 to permit the repetition of 10 frames from one image sequence to the next. These parameter choices yielded a dataset comprising 1707 samples, which were exclusively utilized for training.

We trained a video diffusion model with the following configuration:

- The number of diffusion steps was set to $T = 1000$, with a cosine noise schedule as described in Subsection 1.2.2. We clipped β_t values to be no larger than 0.999 to prevent singularities near the end of the diffusion process.

- For predicting noise added to the image, we utilized a 3D U-Net architecture. It comprised 4 downsampling and upsampling U-Net blocks with the number of channels set to $64, 128, 256$, and 512, respectively. Each block consisted of 2 ResNet blocks, spatial and temporal attention. Additionally, spatial and temporal attention blocks were employed between downsampling and upsampling.

- We employed Adam optimizer described in Subsection 1.1.4, using a learning rate $\alpha = 10^{-4}$. Additionally, we utilized an exponential moving average (EMA) with a factor of $\alpha = 0.995$, which gives more weight to recent data points while still considering older data points. EMA helps maintain a smoothed estimate of model parameters, prevent overfitting, and improve the model's generalization performance.

- During training, the batch size was set to 12. We trained the model for 683 epochs (approximately 100k steps). While additional training steps could be beneficial, we stopped at this number due to the extremely time-consuming nature of training. Further training will be considered as part of future work.

For evaluating the trained video model, we have the option of using Fréchet inception distance (FID) as in Section 4.3. However, in this case, we will proceed with Fréchet video distance (FVD) [36], specifically designed to evaluate the quality of generated videos. The key advantage of FVD over FID is that FVD considers both spatial and temporal aspects, whereas FID focuses only on spatial features. Fréchet video distance feature maps are extracted from Two-Stream Inflated 3D ConvNet (I3D) [37] pretrained on Kinetics-400 dataset [38].

In order to compute the Fréchet video distance, we generated the same number of videos as the dataset's size, which is 1707 videos. Subsequently, a Fréchet video distance score of 154.7 was attained. The graph illustrating the fluctuations of FVD values with respect to the increment in the number of image sequences employed for assessment is presented in Figure 6.2. Furthermore, examples of videos produced by our model are shown in Figure 6.3.

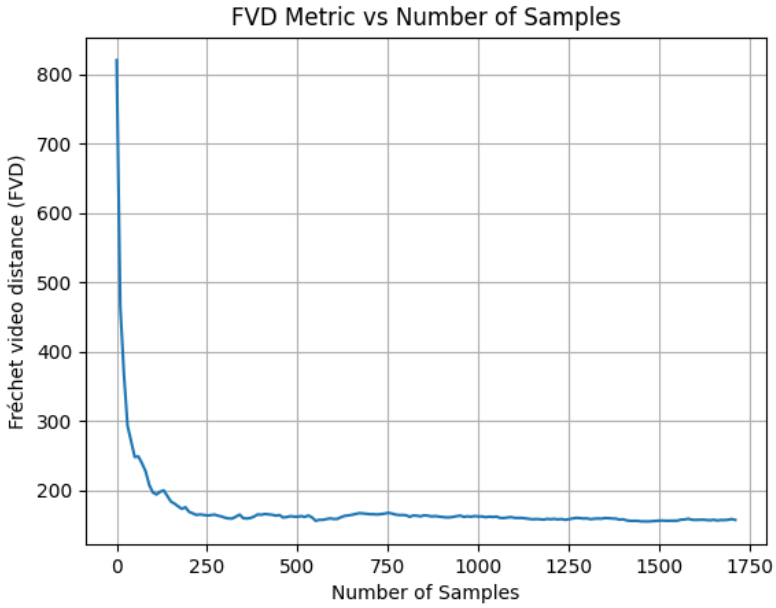

Figure 6.2: FVD scores versus the number of sampled videos used for computation

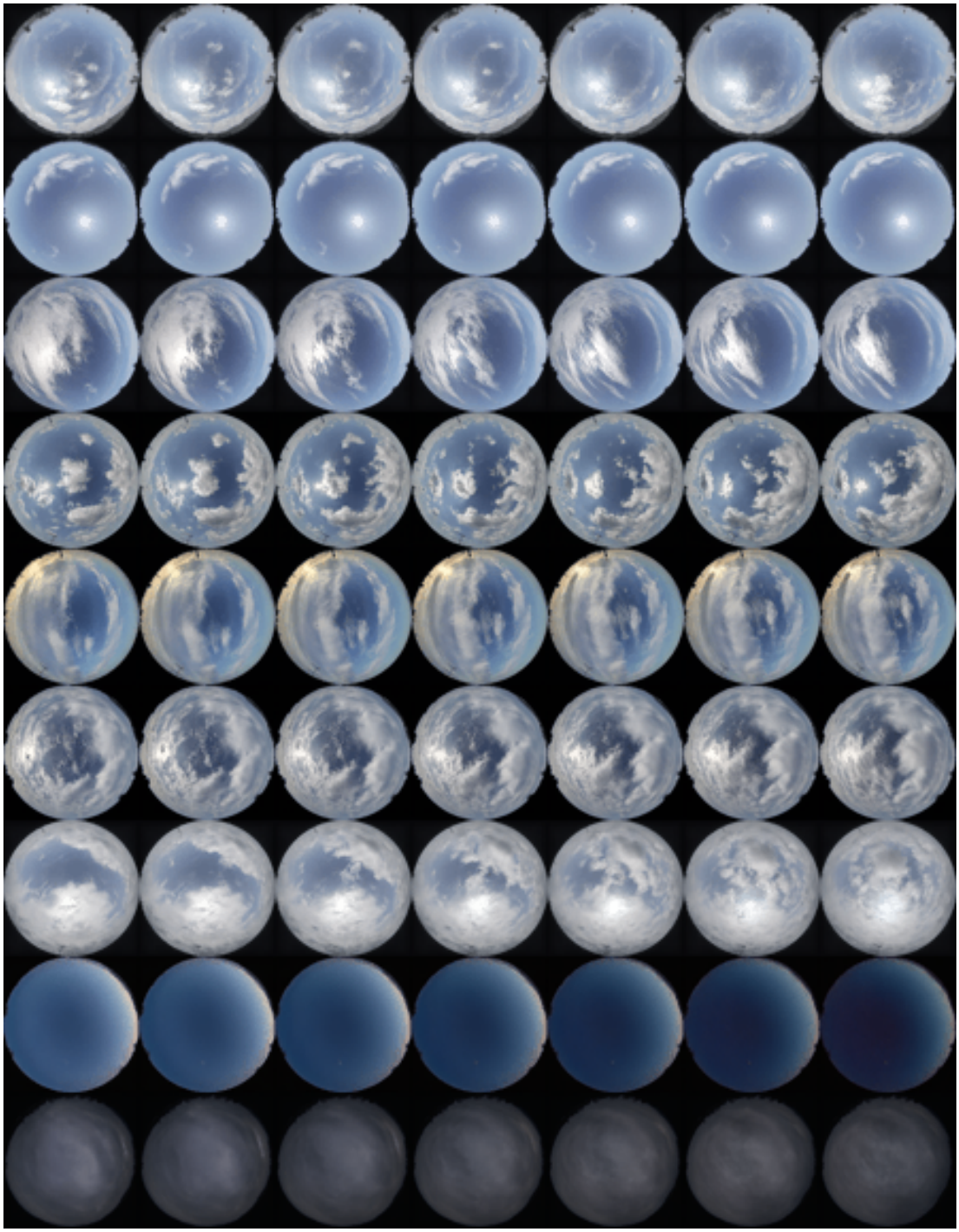

Figure 6.3: 9 samples of videos generated by the Video Diffusion Model. Each row showcases 7 frames from a 20-frame video: frames 1, 4, 7, 10, 13, 16, and 19.

7. Conditional Video Generation

This chapter delves into an alternative video generation technique capable of predicting, infilling, and upsampling videos from a single model. This approach employs a straightforward conditioning scheme, allowing training of the model conditionally and unconditionally simultaneously. Furthermore, we present our findings regarding video prediction tasks and assess the performance of the trained models.

7.1 Random-Mask Video Diffusion Models

The random-mask video diffusion models [4] are relatively close to the video diffusion models from the architectural point of view. However, they diverge in their conditioning technique. The central concept involves assigning a mask to each video, determining which frames are used as conditioning and which are to be predicted. During training, the model is conditioned on a randomly selected mask, leading to along slight alterations in the forward and reverse processes. In each training step, a mask size is determined, followed by a random selection of a subset of frames corresponding to that size. Subsequently, the model undergoes an update step conditioned on the chosen mask. This approach also allows mixed training, where with a probability p_U, an empty mask is applied. The pseudocode for RaMViD from the original paper [4] is demonstrated below:

Algorithm 4 RaMViD

Initialize model $\sim s_\theta$
T = Number of diffusion steps
K = Max number of frames to condition on
L = Length of the video
while not converged **do**
  $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{0, \dots, T\})$
  $b \sim \text{Bernoulli}(p_U)$
  if b **then**
    $\mathcal{C} = \varnothing$
  else
    $k \sim \text{Uniform}(\{1, \dots, K\})$
    $\mathcal{C} \sim \text{Uniform}\left(\{S \subseteq \{0, \dots, L-1\} \,:\, |S| = k\}\right)$
  end if
  $\mathcal{U} = \{0, \dots, L-1\} \setminus \mathcal{C}$
  $\mathbf{x}_t^{\mathcal{U}} \sim \mathcal{N}\left(\mathbf{x}_t^{\mathcal{U}}; \mathbf{x}_0^{\mathcal{U}}, (\sigma^2(t) - \sigma^2(0))\,\mathbf{I}\right)$
  $\mathbf{x}_t = \mathbf{x}_t^{\mathcal{U}} \oplus \mathbf{x}_0^{\mathcal{C}}$
  Gradient step on: $\nabla_\theta \mathbb{E}_{\mathbf{x}_0}\left\{\mathbb{E}_{\mathbf{x}_t^{\mathcal{U}}|\mathbf{x}_0}\left[\left\|s_\theta(\mathbf{x}_t, t)^{\mathcal{U}} - \nabla_{\mathbf{x}_t^{\mathcal{U}}} \log p(\mathbf{x}_t^{\mathcal{U}} \mid \mathbf{x}_0)\right\|_2^2\right]\right\}$
end while

7.2 Results

One of the advantages of random-mask video diffusion models is that we do not require specific data preparation steps to train the model conditionally. As mentioned earlier, we train the model both conditionally and unconditionally simultaneously. Therefore, we used the same dataset for training the video diffusion model in Chapter 6. Additionally, we excluded 100 training instances to perform the evaluation.

- The number of the diffusion steps was set to $T = 1000$, following the standard practice. However, we employed the linear variance schedule from $\beta_1 = 10^{-4}$ to $\beta_T = 2 * 10^{-2}$, as the authors of the original paper found that it works better than cosine when training model conditionally.
- For modeling the added noise, we utilized a 3D U-Net with the kernel size of $3 \times 3 \times 3$. The model comprised 4 downsampling and 4 upsampling U-Net blocks with the number of channels set to 128, 256, 384, and 512, respectively. Each U-Net block consisted of 2 ResNet layers. Spatial self-attention was applied in the last two downsampling steps and the first two upsampling blocks.
- We employed the AdamW optimizer, as described in Subsection 1.1.4, with a learning rate of $\alpha = 10^{-5}$. The exponential moving average (EMA) of the weights was used for sampling, with a rate of $\alpha = 0.9999$.
- The batch size was set to 12, and we trained the model for $120,000$ steps, approximately equivalent to 840 epochs. The unconditional rate was set to $p_U = 0.5$ (probability of conditioning on the mask), and the maximum number of frames to mask during training was set to $K = 4$.

For performance evaluation, we employed identical metrics to those utilized for assessing the conditional diffusion model in Section 5.3. The corresponding charts are shown in Figure 7.1. Notably, on the last 20th frame, the average structural similarity index measure registered 0.88, the peak signal-to-noise ratio measured 20.4, and the mean squared error attained 0.17. The Mean Squared Error of cloud coverage exhibited a deviation of 5.4% and 7.18% on the 8th and 16th generated images, respectively. A few generated videos together with original ones are demonstrated in Figure 7.2.

In general, the metrics closely align with our findings with conditional diffusion models. However, the structural similarity metric significantly increased, and the MSE of cloud coverage notably improved. Moreover, convergence was enhanced across all metrics as the number of generated frames increased. This may be beneficial in the generation of longer sequences of images.

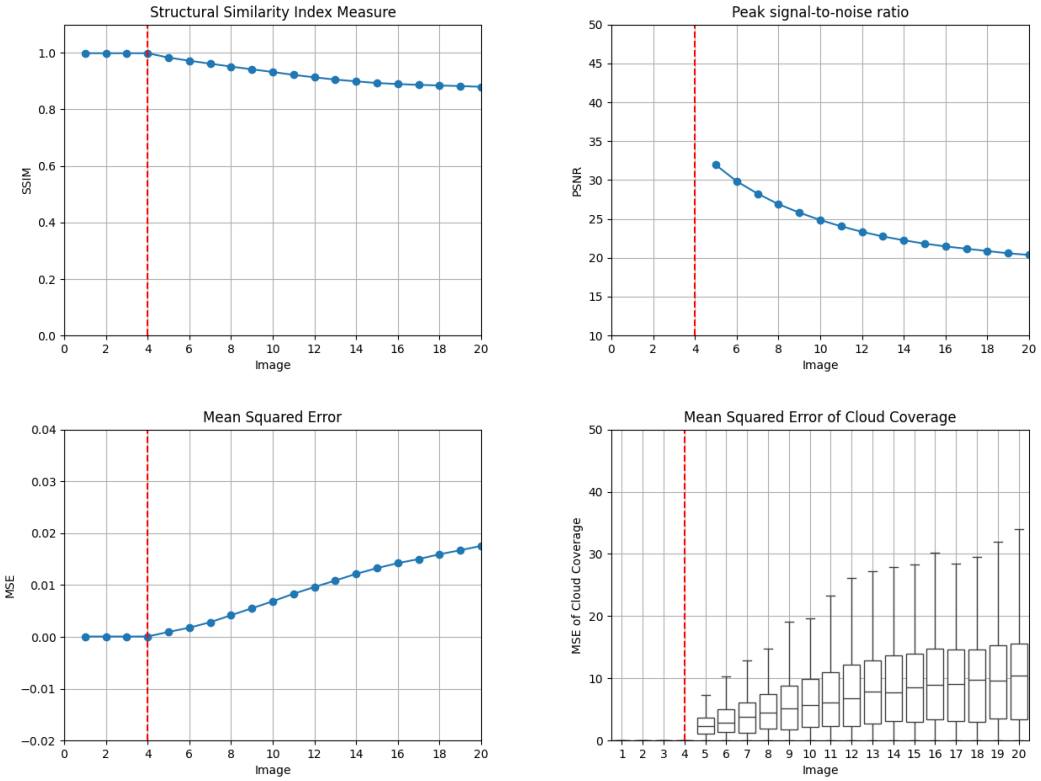

Figure 7.1: Four charts with SSIM, PSNR, MSE, and cloud coverage MSE measures are shown. The vertical axis corresponds to the score, while the horizontal axis corresponds to the generated frame by our model. The red dotted line shows where the conditioning frames end.

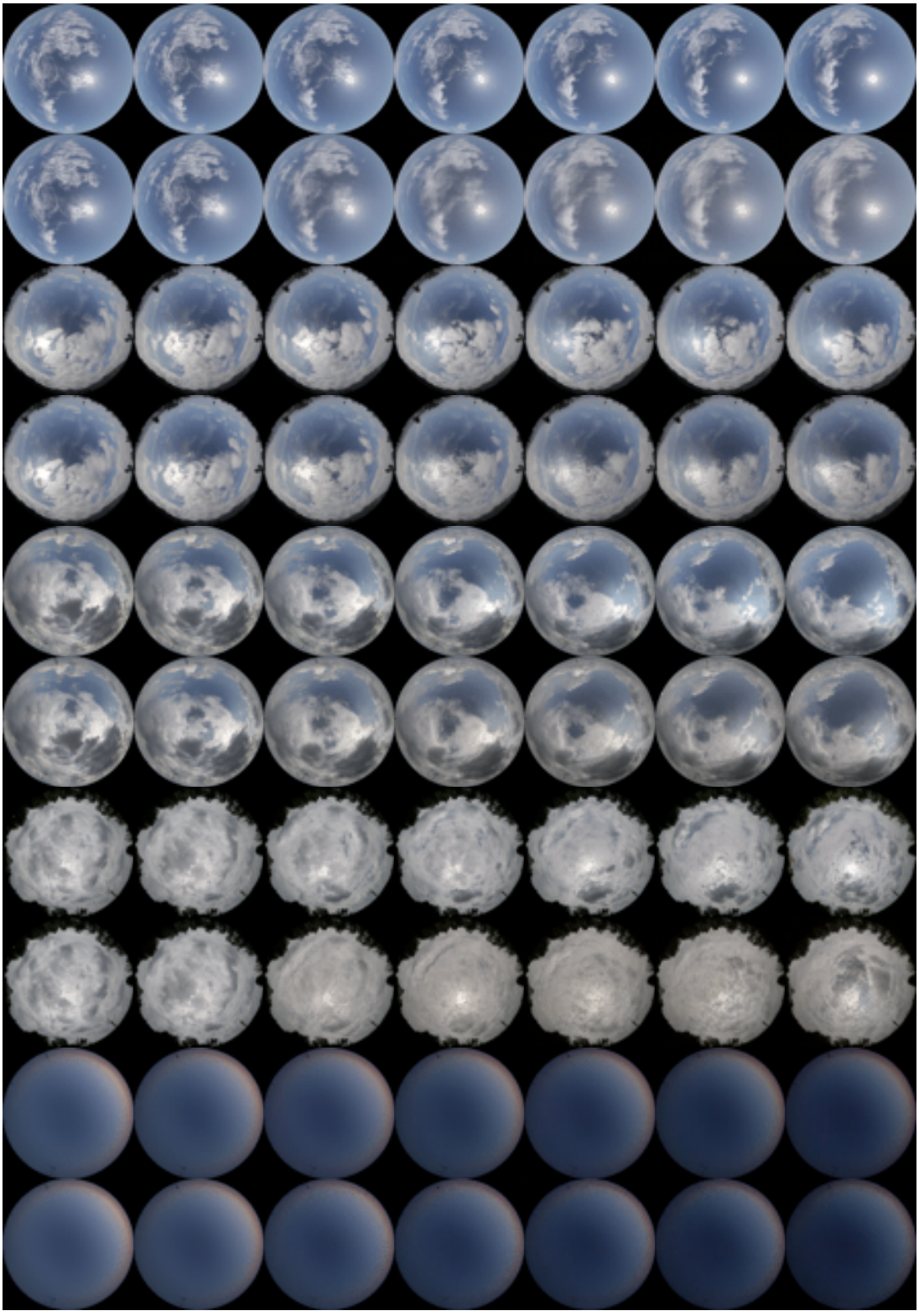

Figure 7.2: 10 rows of images, each comprising 7 frames extracted from a 20-frame image sequence: Specifically, frames 1, 4, 7, 10, 13, 16, and 19. Odd rows depict frames extracted from the original videos. Even rows represent the frames extracted from predicted videos. Note that first two images remain consistent across each row, as they serve as conditioning images for our model.

8. Conclusion

In this work, we conducted a comprehensive investigation into the denoising diffusion models, exploring both the theoretical side and practical applications. Our experiments consisted of various tasks, including the unconditional generation of images and videos, as well as visual prediction of sky conditions over time.

In the domain of unconditional image generation, our models exhibited superior performance compared to generative adversarial network utilized in SkyGAN work [28] by a factor of 2.69, proving the capability of diffusion models to produce high-quality and visually realistic images.

During our exploration of video diffusion models, we observed that certain modifications to the usual diffusion architecture enabled the generation of visually appealing videos that accurately capture the movements of objects in the sky. The metrics we obtained in this experiment are satisfactory and are comparable to those reported in similar research studies on different datasets.

Transitioning to conditional diffusion models, our goal was to train a model capable of forecasting the evolving appearance of the sky over time. We decided to asses, not the prediction of the "next" frame but longer-term predictions by sequentially generating longer sequences. Ultimately, we noticed a degradation in prediction accuracy as we generated increasingly longer sequences. However, despite the degradation, our metrics still indicated reasonable values, close to the ground truth. Furthermore, our models outperformed the GAN [29] trained for the same task.

Finally, we introduced random-mask video diffusion models, which closely resemble video diffusion models but with a novel conditioning technique enabling the selection of the frames for conditioning. This approach demonstrated slightly superior results to the standard conditional diffusion model, once again surpassing the GAN [29].

Future work

There are several things we could improve in our work in the future. Adjusting parameters and architecture to train models for higher image resolutions would be beneficial for the unconditional model. Alternatively, training another super-resolution model, such as upscaling from 128×128 to 512×512, could enhance the quality of outputs.

Experimenting with the number of frames we condition on and exploring classifier-free guidance could lead to advancements in the context of conditional models. Additionally, our video models (VDM and RaMViD) may not have reached their full potential due to GPU limitations, so further training could help improve visual quality and prediction accuracy.

As subsequent research, modifying our existing models or exploring different approaches to condition additional environmental information, such as time, location, wind direction, and others, is also promising. Such modification could allow our models to make highly accurate predictions over extended periods but require larger datasets.

Bibliography

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

[2] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[3] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models, 2022.

[4] Tobias Höppe, Arash Mehrjou, Stefan Bauer, Didrik Nielsen, and Andrea Dittadi. Diffusion models for video prediction and infilling, 2022.

[5] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[6] A quantum model for multilayer perceptron. Scientific Figure on ResearchGate.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[8] Advances in quantum deep learning: An overview. Scientific Figure on ResearchGate.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function, 1989.

[10] Kevin Gurney. *An Introduction to Neural Networks*. CRC Press, 1997.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[12] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[13] Y. Nesterov. A method of solving a convex programming problem with convergence rate o(1/k**2), 1983.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[15] T. Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[18] Yann Lecun and Y. Bengio. Convolutional networks for images, speech, and time-series. 01 1995.

[19] Learning shape features and abstractions in 3d convolutional neural networks for detecting alzheimer's disease. Scientific Figure on ResearchGate.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[22] Shearlets as feature extractor for semantic edge detection: The model-based and data-driven realm. Scientific Figure on ResearchGate.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[24] Resnet50-boosted unet for improved liver segmentation accuracy. Scientific Figure on ResearchGate.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[26] Generative adversarial networks in computer vision: A survey and taxonomy. Scientific Figure on ResearchGate.

[27] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[28] Martin Mirbauer, Tobias Rittig, Tomáš Iser, Jaroslav Křivánek, and Elena Šikudová. Skygan: Realistic cloud imagery for image-based lighting. *Computer Graphics Forum*, n/a(n/a):e14990, 2023.

[29] George Andrianakos, Dimitrios Tsourounis, Spiros Oikonomou, Dimitrios Kastaniotis, George Economou, and Andreas Kazantzidis. Sky image forecasting with generative adversarial networks for cloud coverage prediction. pages 1–7, 07 2019.

[30] Yusuke Hatanaka, Yannik Glaser, Geoff Galgon, Giuseppe Torri, and Peter Sadowski. Diffusion models for high-resolution solar forecasts, 2023.

[31] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[32] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[34] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.

[35] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018.

[36] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges, 2019.

[37] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018.

[38] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.

List of Figures

A. Attachments

A.1 Code

The Python code utilized for data manipulation, training, sampling, and evaluation of the models presented in this work.

A.2 README

README file containing installation and execution instructions, project structure overview, and a short guideline for loading trained models and dataset examples for sampling.