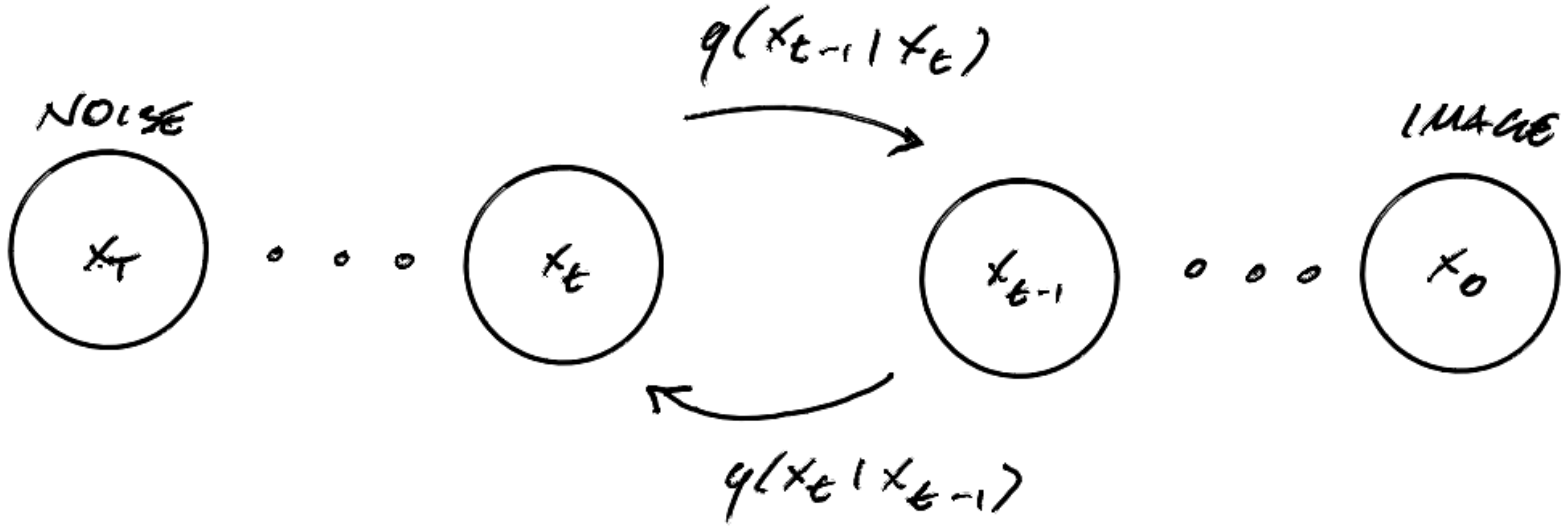
Cat 🐱 Generator

with Diffusion Model


$q(x_{t-1} | x_t)$
NOISE
IMAGE
x_T
x_t
x_{t-1}
x_0
$q(x_t | x_{t-1})$

recursive definition:

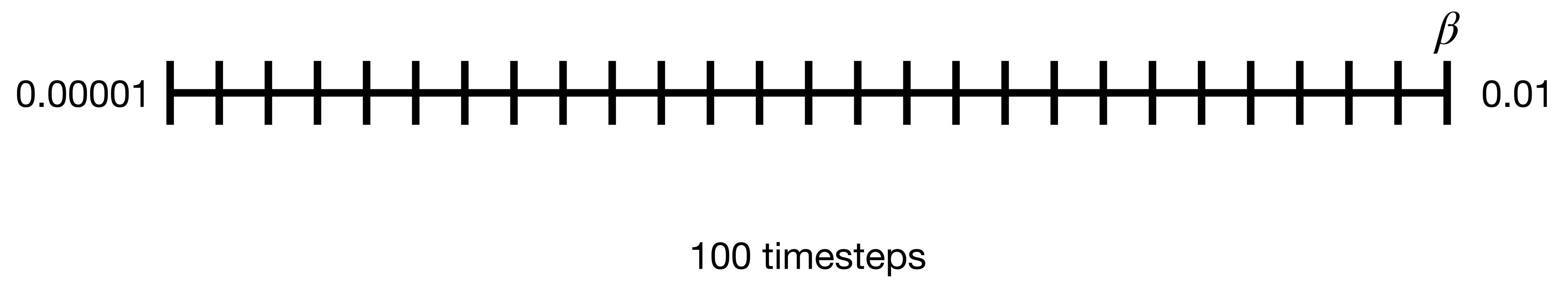
$$x_t = \sqrt{1 - \beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot z_{t-1}$$

Where $z_{t-1} \sim \mathcal{N}(0, I)$

Variance schedule governs how noise is added over time.

this linear function could be rappresented by analogy distribution point of view:

$$x_t \sim \mathcal{N}(\sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t)$$



let define

$\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

Let the recursion be:

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t}\, \epsilon_{t-1}$$

Unrolling the recursion:

$$\begin{aligned}
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\,\epsilon_{t-2}\right) + \sqrt{1-\alpha_t}\,\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\sqrt{\alpha_t}\,\epsilon_{t-2} + \sqrt{1-\alpha_t}\,\epsilon_{t-1} \\
&= \dots
\end{aligned}$$

Eventually, we reach:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\,\hat{\epsilon}$$

so we can rewrite

$$q(\boldsymbol{x}_t | \boldsymbol{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, 1 - \bar{\alpha}_t)$$

this formulation conduct us to a direct sampling:

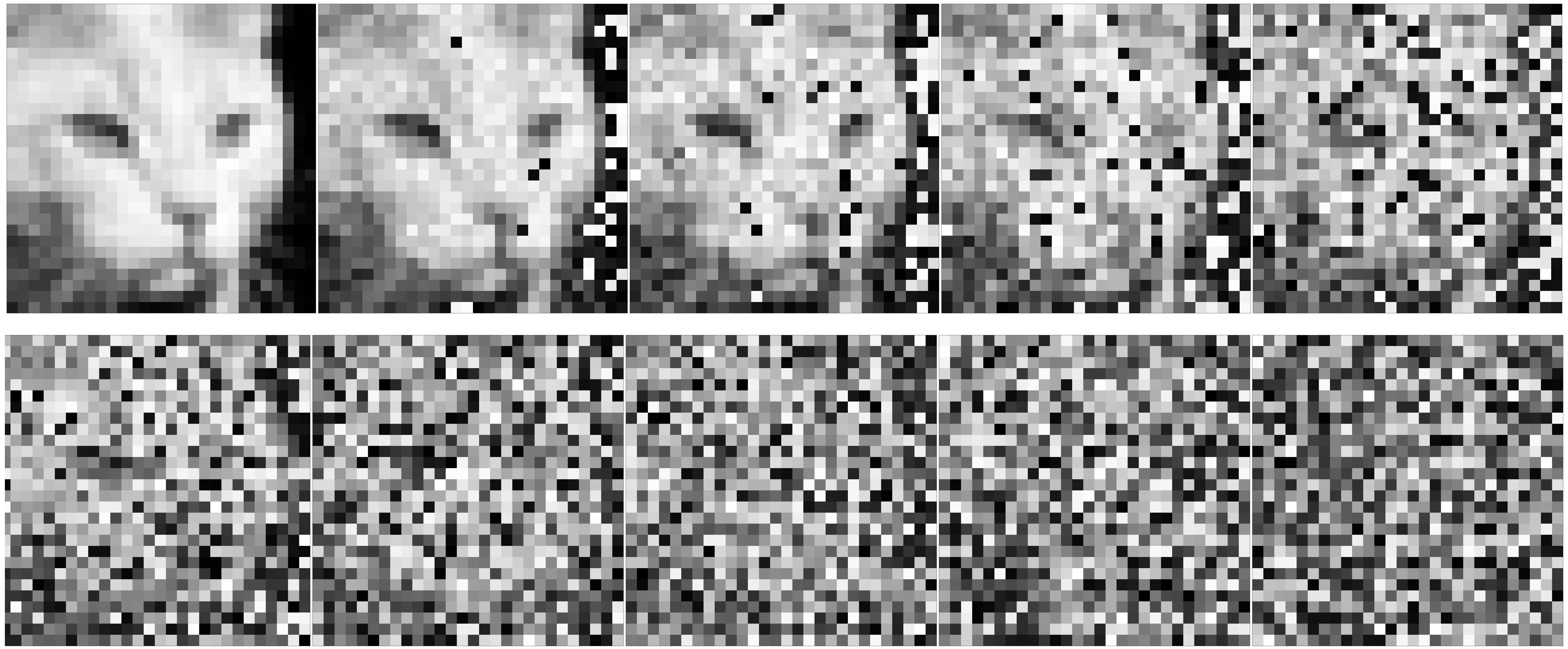
$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot z$$

so finally we have no more dependence on t

so **we can compute x_t directly, skipping all the steps**

Dataset

Resize
Grayscale



Backward Process

Why can't we compute $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ analytically?

Because this distribution depends on the **true data distribution** $p(\mathbf{x}_0)$, which we **don’t have access to**.

Instead, we **approximate** it using a neural network.

Key Idea: Conditioning on $\mathbf{x}_0$

To simplify, we consider the **posterior** of the forward process:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

This **can** be computed analytically, and it turns out to be a **Gaussian**:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}\right)$$

We use this to define our training objective.

The mean and variance used in this approximation are:

Variance:

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$$

This is a **precomputed constant** for each timestep.
It's derived from the forward process and doesn't depend on $\mathbf{x}$, so we don't need the network to predict it.

Mean:

$$\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\tilde{z}_t\right)$$

Here, $\tilde{z}_t$ is the **noise** that was added at time step t
During training, the model learns to **predict this noise**.

What does the model learn?

Instead of learning $\tilde{\mu}_t$ directly, we train the model to **predict the noise** $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, and plug that into the equation for $\tilde{\mu}_t$.

This gives us an approximate reverse step:

$$\mathbf{x}_{t-1} \sim \mathcal{N}\left(\tilde{\mu}_t(\mathbf{x}_t, \boldsymbol{\epsilon}_\theta), \tilde{\beta}_t \mathbf{I}\right)$$

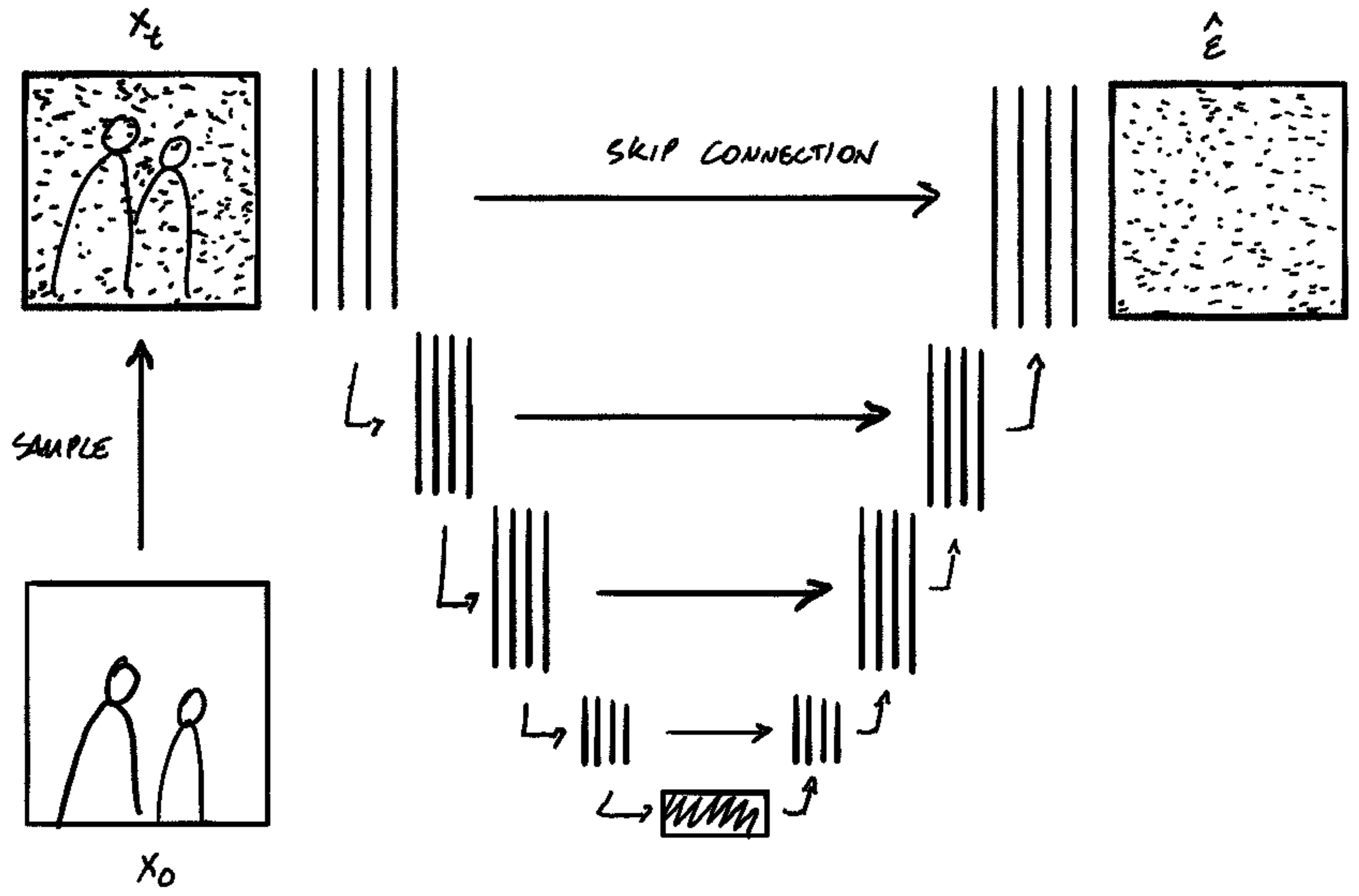
UNet

We approximate $\tilde{z}_t$ with a NN.

The reason why we are gonna use Unet is they excels at capturing **local and global context** useful to denoise images in diffusion models.

Unet are often **timestep-aware**, enabling the model to adapt its denoising strategy based on how much noise is present.

Unet **skip connections** allow the decoder to access fine-grained spatial information from earlier layers.



Loss

define a simple loss norm-1: $\|z_{t-1} - \tilde{z}_t\|_1 = \|z_{t-1} - NN(\boldsymbol{x}_t, t)\|_1$

z_{t-1} is the noise used to compute $\boldsymbol{x}_t$ in the forward process

Remember that to compute $\boldsymbol{x}_t$ we dropped the time index on z:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{z}$$

Simply, we can sample z, compute $\boldsymbol{x}_t$, and recover z with the NN

Training

The model is trained to denoise images corrupted by varying levels of noise.

At each step, a random timestep is chosen, and the image is noised accordingly.

The model sees the noisy image and the timestep, and learns to predict the noise added.

By minimizing the difference between predicted and true noise, the model learns how to reverse the diffusion (noising) process, and thus generate clean images from noise.

Sampling

We said that

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}\left(\tilde{\mu}_t, \tilde{\beta}_t\right)$$

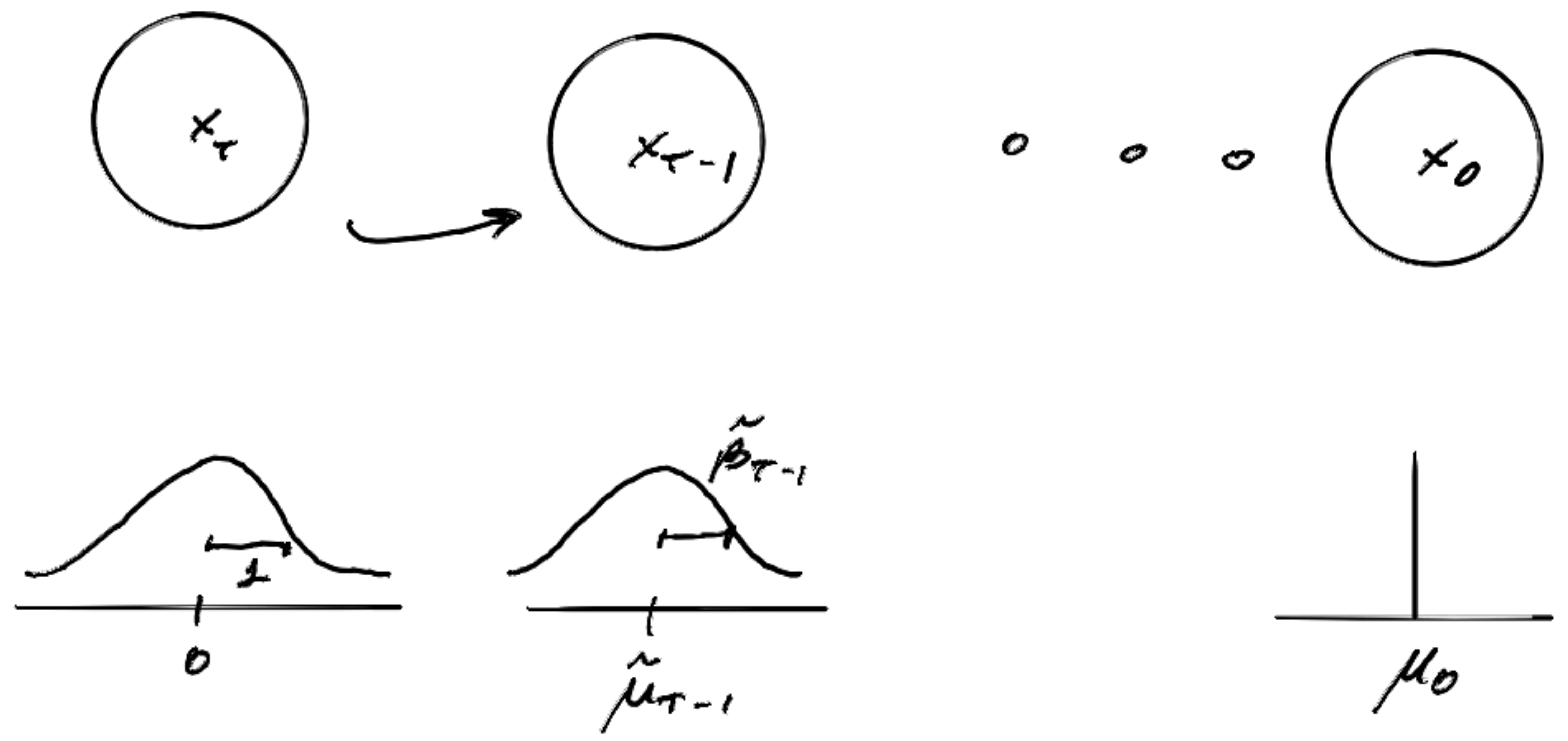
where

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

and

$$\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\tilde{\boldsymbol{z}}_t\right)$$

we would define a way to reconstruct original image from noise



Sampling algorithm

$\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ start from a noise random image

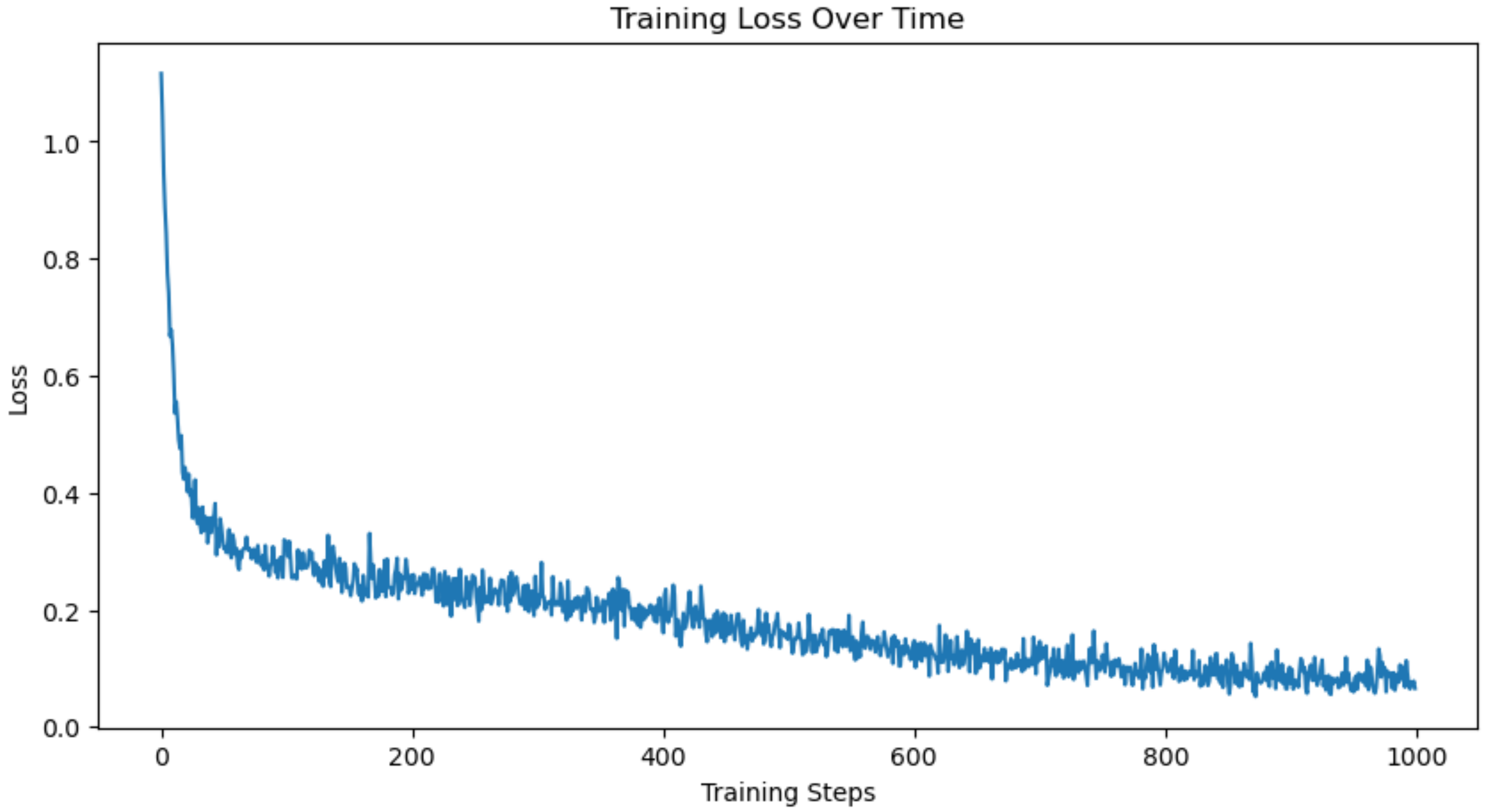
for $t = T-1,\ldots,0$ **do** (from noise to image timestep)

1) $z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ if $t > 1$ else $z = 0$ (generate noise)

2) $\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ NN(\boldsymbol{x}_t, t)\right) + \tilde{\beta}_t z$ (based on Unet trained move to the next denoised image)

return $\boldsymbol{x}_0 = \mu_0$ the final image recostructed

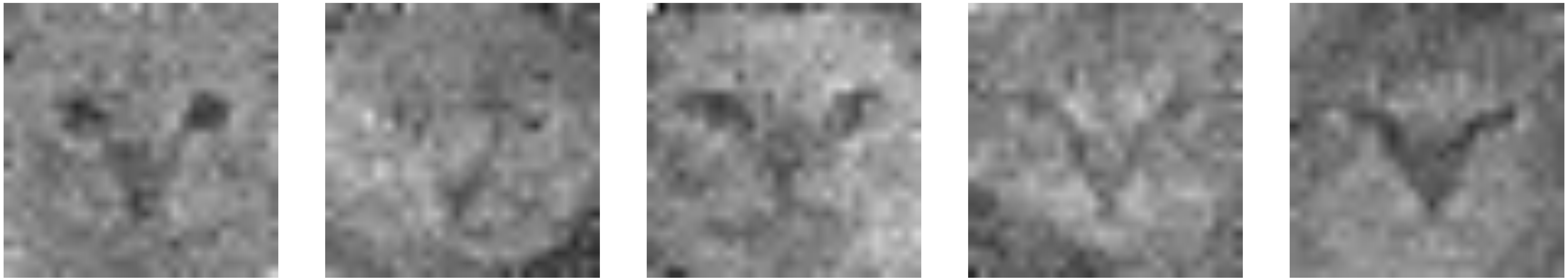
Loss performance



Results

(real 28x28 image)

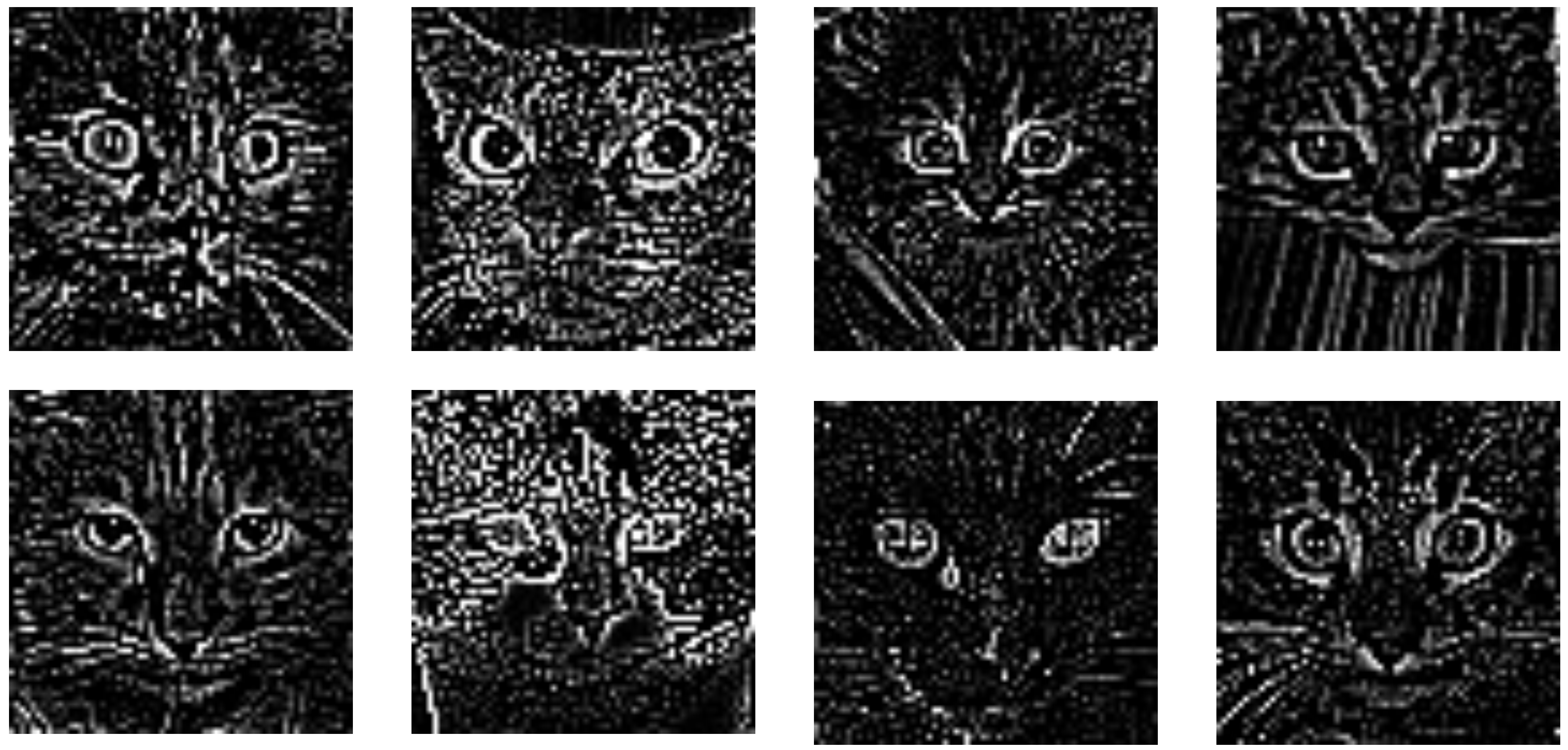
zoom on results



muzzle and eyes have been captured

A little trial

Trying to use an edge detector kernel on original dataset



Results even worse 👌