DENOISING DIFFUSION PROBABILISTIC MODELS

Harshit Agarwal | Jan 31st 2025

Kalinga Institute of Industrial Technology | MLSA

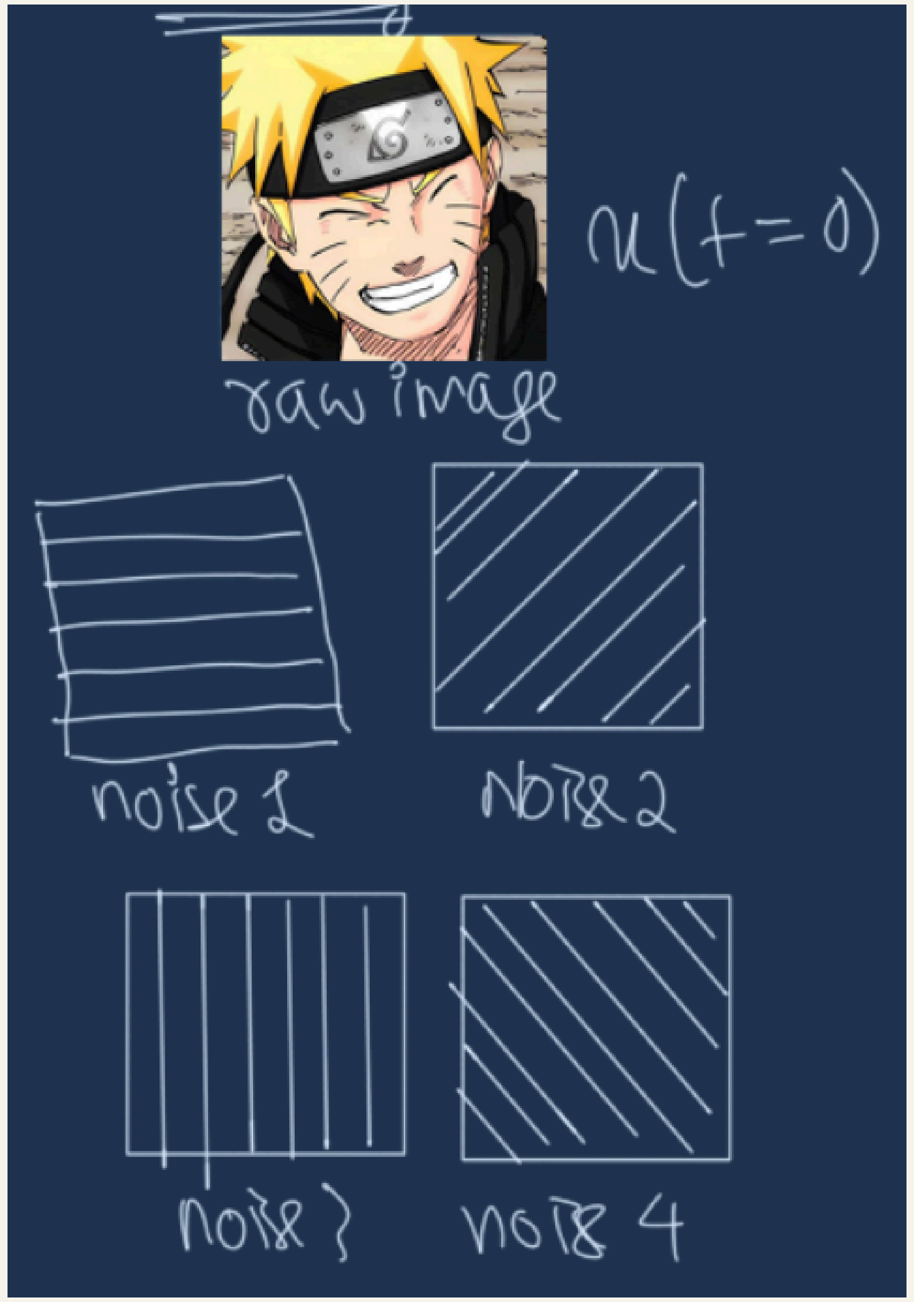
ABSTRACT

High quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics.
This talk covers the mathematical intuitions of denoising, diffusion and how it translates into the code implementation.

Find code on github.com/aharshit123456/ddpm

OVERVIEW

INTRODUCTION

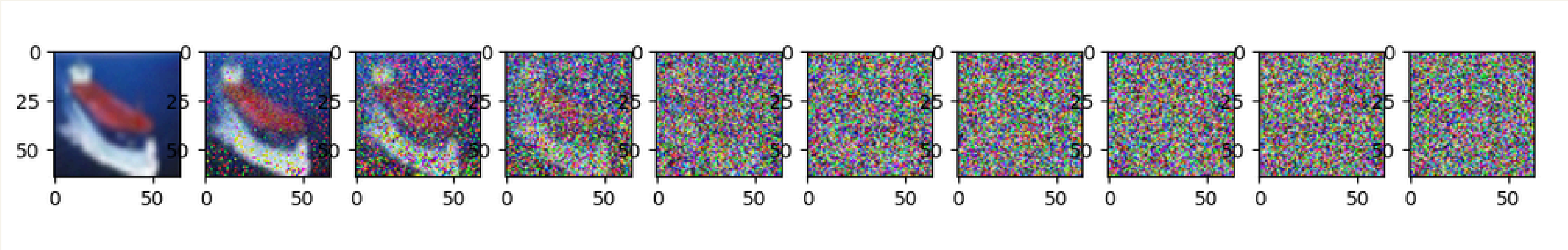


DENOISING

r/howToLearnToCreate~~Hen~~AnImage

Step 1: throw junk on a photo

Step 2: learn how to remove the junk from the photo



lets play a game ? goto **learndiffusion.vercel.app**

PROBLEM AND SOLUTION

HOW TO THROW JUNK AT AN IMAGE EFFICIENTLY ????

The problem with noise augmentation is to generate effective and balanced noise.

HOW TO REMOVE THIS JUNK??

The next big problem is to learn and remember denoising process efficiently.

NOISE

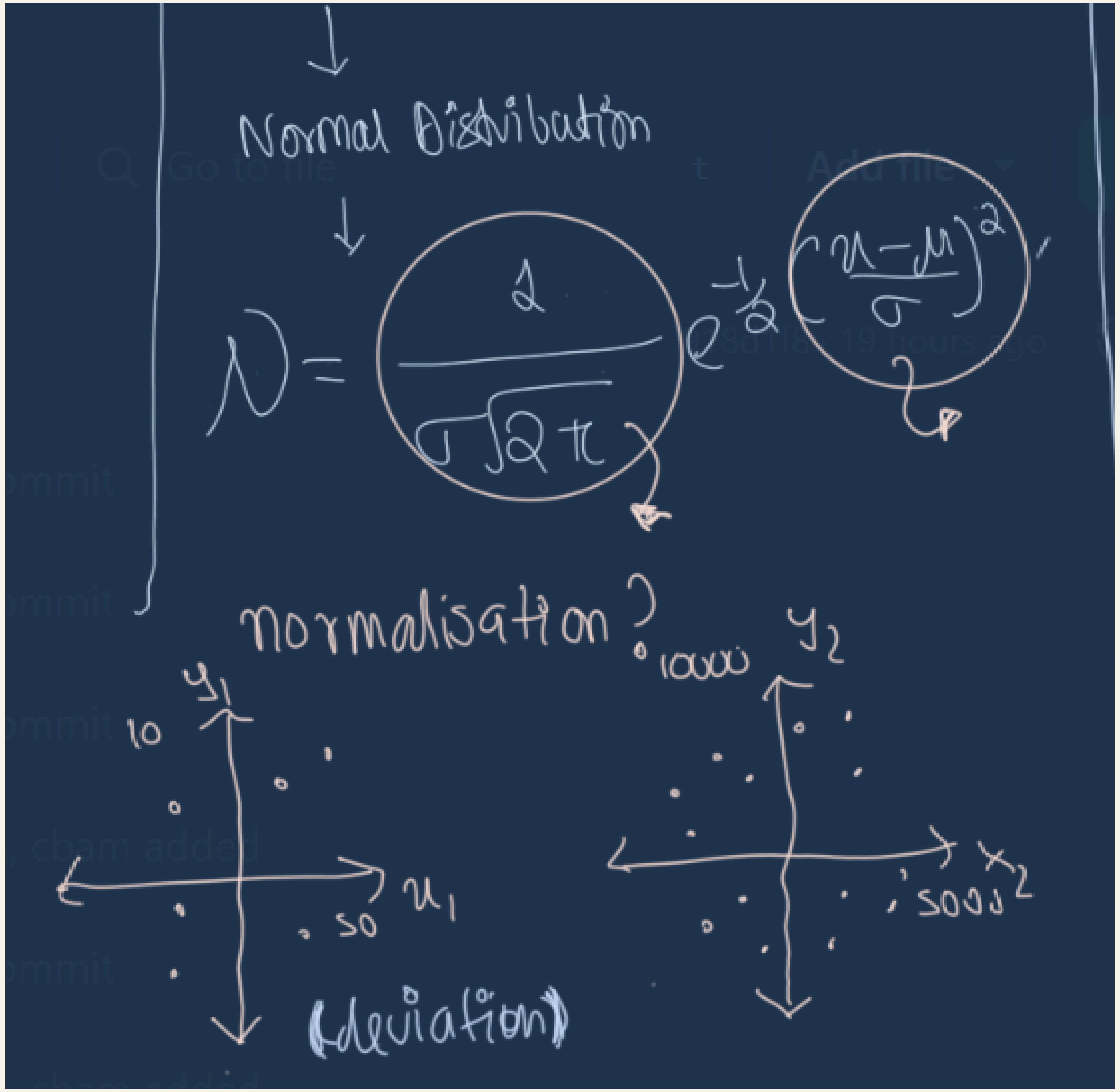
the best way to create noise is to use normal distribution and diffusion models.

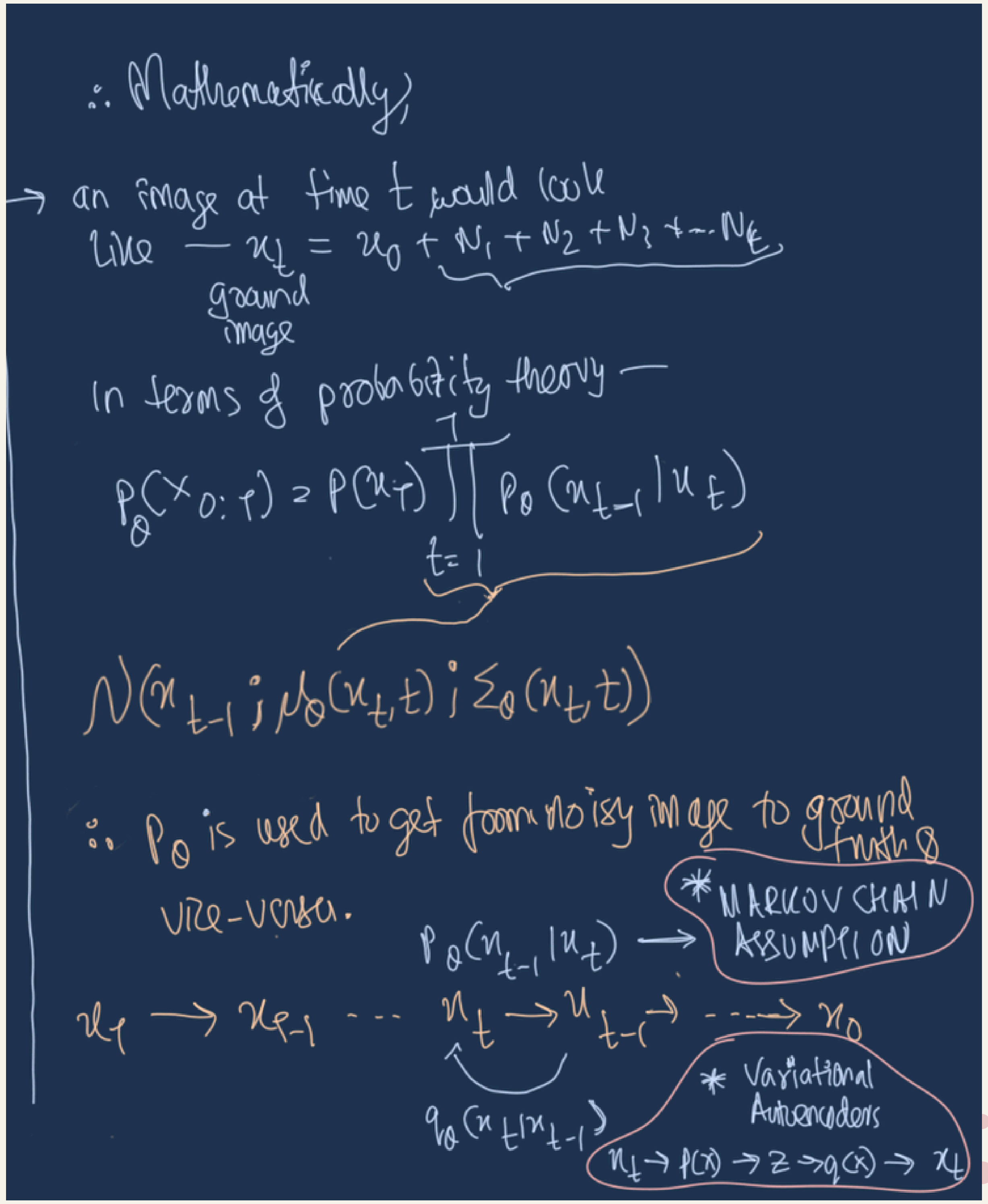
DENOISE

make neural networks that take in noisy image and train them to output image with lesser noise.

KIIT DU | Harshit Agarwal

NORMAL DISTRIBUTION





NORMAL DISTRIBUTION

For given data x and it's latent variable z, the joint probability distribution $p(x, z)$ is given as,

$$p(x, z) = p(x) \cdot p(z|x) \tag{1}$$

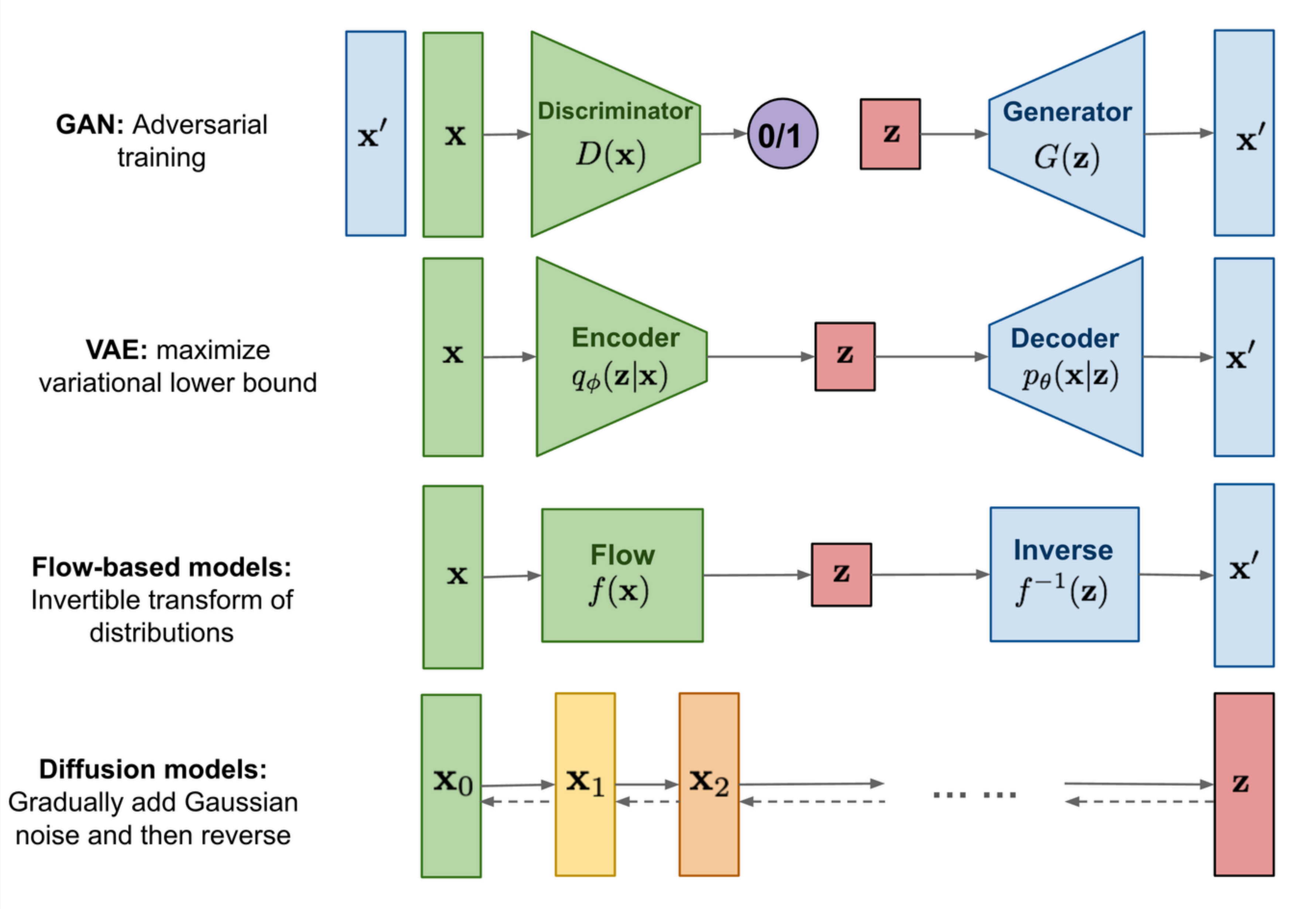
The marginalized latent z provides the full probability of seeing the data,

$$p(x) = \int p(x, z)\ dz \tag{2}$$

And from Bayes' rule,

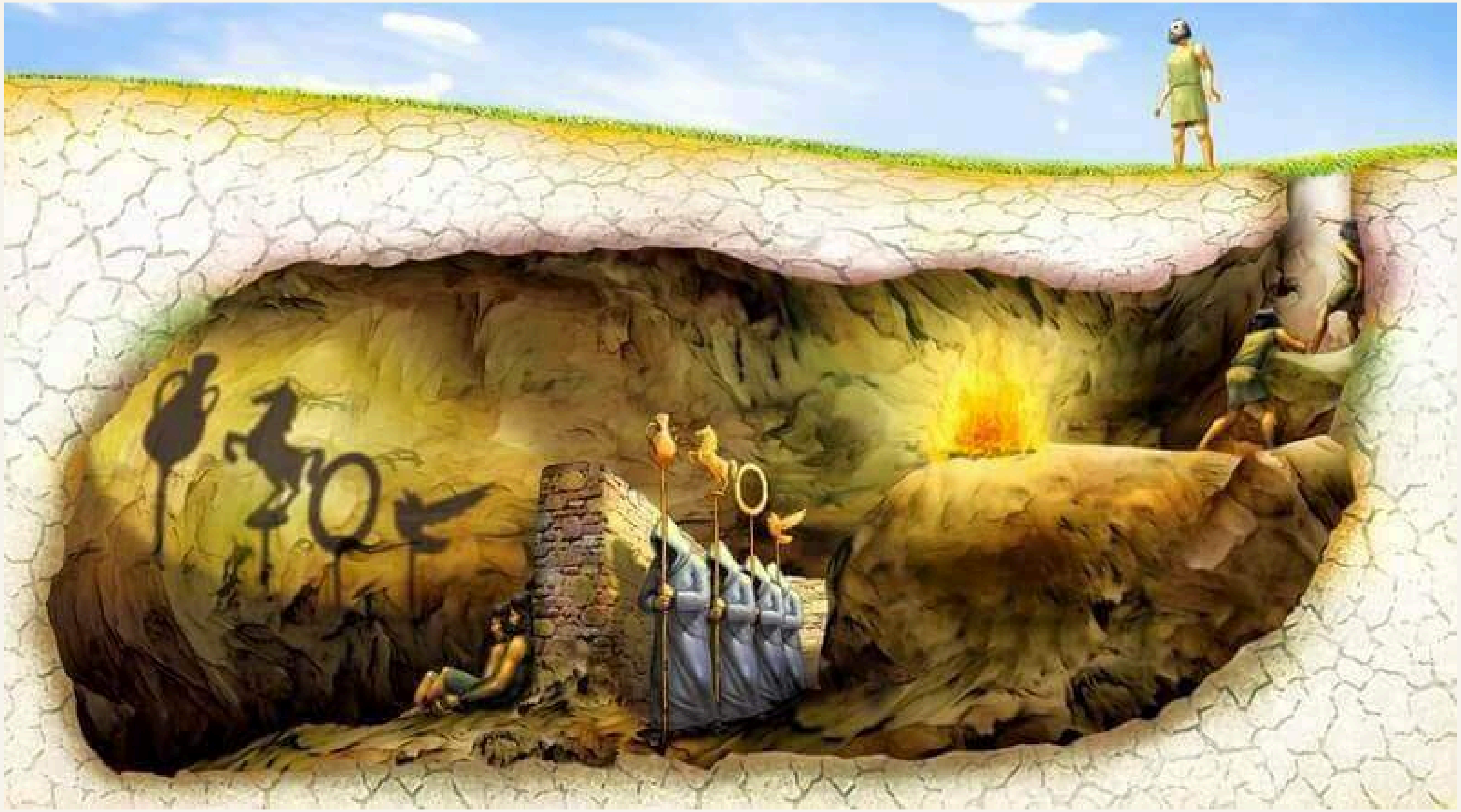
$$p(z|x) = \frac{p(x|z) \cdot p(z)}{p(x)} \tag{3}$$

where, $p(z|x)$ is the *posterior*, $p(x|z)$ is the *likelihood*, $p(z)$ is the *prior* and $p(x)$ is the *evidence*.

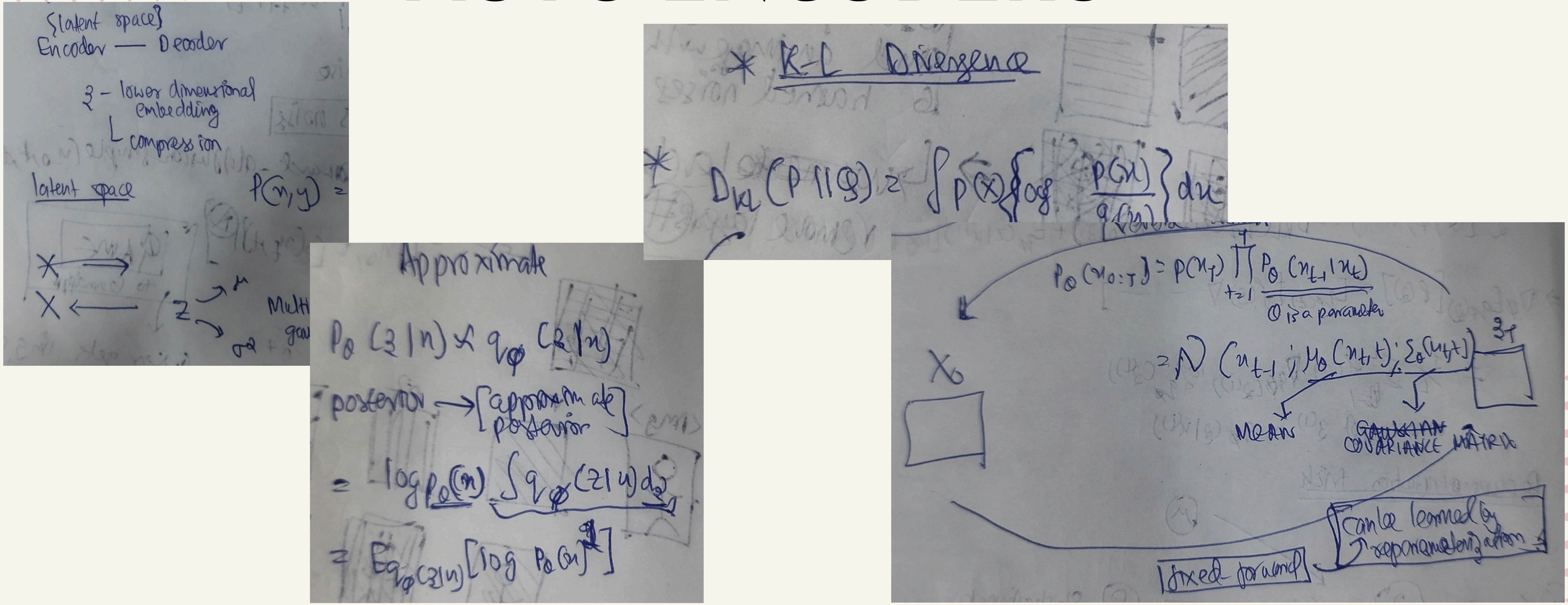


VARIATIONAL AUTOENCODERS AND UNET

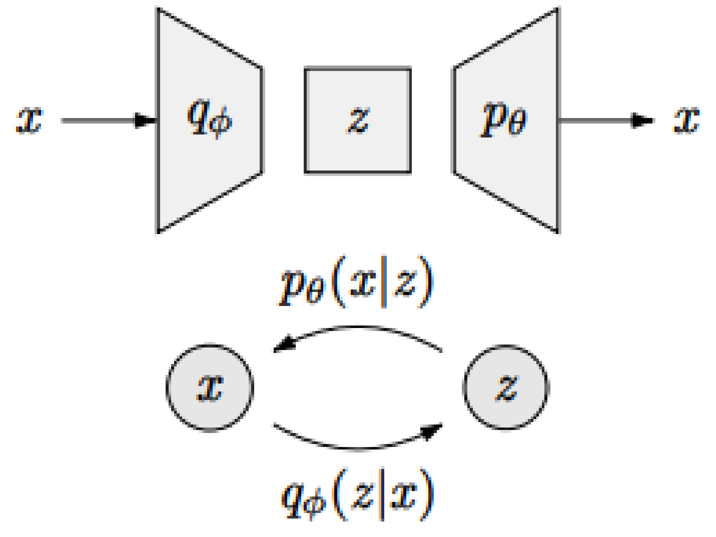
PLATO'S ALLEGORY



AUTO ENCODERS



AUTO ENCODERS



$$
\begin{aligned}
\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)\cdot p(z)}{q_\phi(z|x)}\right] && (10)\\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(z)}{q_\phi(z|x)}\right] && (11)\\
&= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z|x) \,||\, p(z))}_{\text{prior matching term}} && (12)
\end{aligned}
$$

Same thing as earlier but more readable, I guess ?

LOSS FUNCTION

LOG LIKELIHOOD

$$\log p_\theta(x_0) = p_\theta(x_0) \int q(z|x)\,dz$$

$$= \int q_\phi(z|x) \log p(x)\,dx \qquad \left\{ \because p(x) \to \frac{p(x,z)}{q(z|x)} \right\}$$

$$= E_{q_\phi(z|x)}\left[\log p(x)\right]$$

$$= E_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right] + E_{q_\phi(z|x)}\left[\log \frac{q(z|x)}{p(z|x)}\right]$$

$$= \underline{\underline{VLB}} + D_{KL}\left[q_\phi(z|x) \,||\, p(z|x)\right]$$

$$\therefore \log p_\theta(x_0) \geq E_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right]$$

$(z = x_{1:T})$ if referring to paper

$$\text{Loss} = \left\| \text{GROUND TRUTH NOISE} - \text{PREDICTED NOISE} \right\|$$

applying the forward process posterior formula (7):

$$L_{t-1} - C = \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t(\mathbf{x}_0,\epsilon), \frac{1}{\sqrt{\bar\alpha_t}}(\mathbf{x}_t(\mathbf{x}_0,\epsilon) - \sqrt{1-\bar\alpha_t}\epsilon)\right) - \boldsymbol{\mu}_\theta(\mathbf{x}_t(\mathbf{x}_0,\epsilon),t)\right\|^2\right] \tag{9}$$

$$= \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t(\mathbf{x}_0,\epsilon) - \frac{\beta_t}{\sqrt{1-\bar\alpha_t}}\epsilon\right) - \boldsymbol{\mu}_\theta(\mathbf{x}_t(\mathbf{x}_0,\epsilon),t)\right\|^2\right] \tag{10}$$

$\tilde\mu_t(x_t, x_0)$ is defined as $\dfrac{\sqrt{\bar\alpha_{t-1}}\,\beta_t}{1-\bar\alpha_t} x_0 + \dfrac{\sqrt{\alpha_t}(1-\bar\alpha_{t-1})}{1-\bar\alpha_t} x_t$ $\Big\}$ $\tilde\beta_t = \dfrac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\beta_t$

$$E_q\left[\frac{1}{2\sigma_t^2}\left\|\tilde\mu_t(x_t, x_0) - \mu_\theta(x_t,t)\right\|^2\right] + C$$

LOSS FUNCTION

SIMPLIFIED LOSS FN

image preserved at timestept according to variance schedule

$$L(\theta) = E_{t,x_0,\epsilon}\left\| \epsilon - \underbrace{\epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t\right)}_{\text{predicted noise}} \right\|^2$$

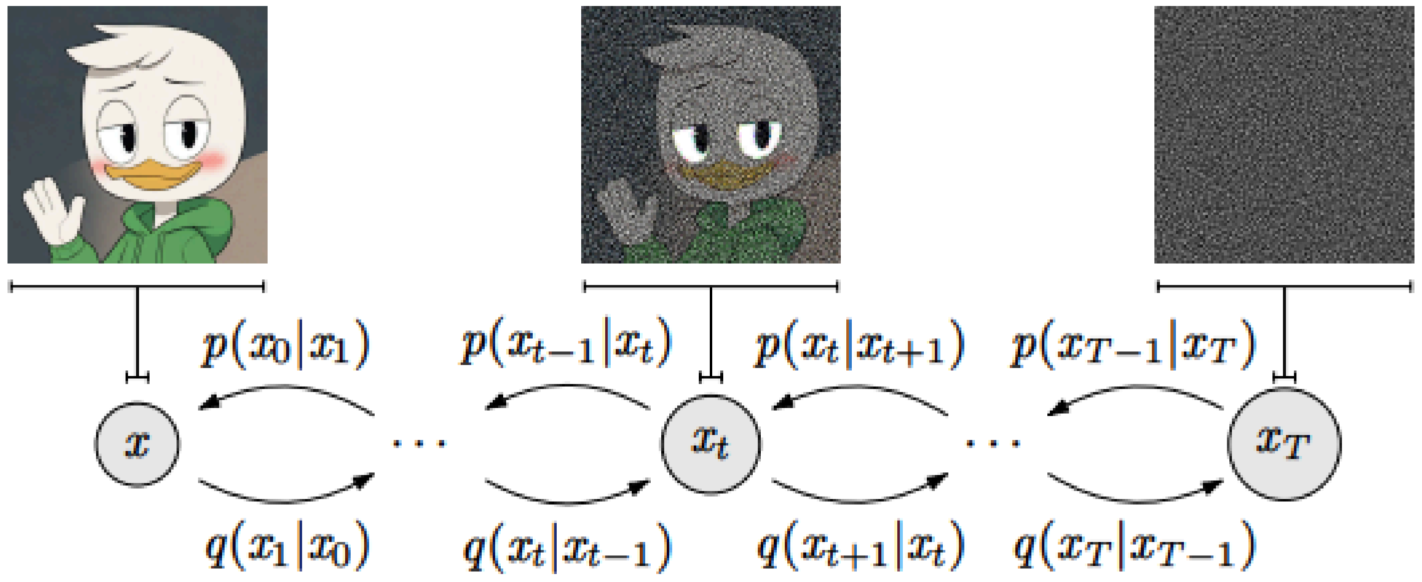
ground truth noise

event not preserved according to variance schedule at t

KIIT DU | Harshit Agarwal

VARIATIONAL DIFFUSION

KIIT DU | Harshit Agarwal



Restriction I lets me abuse the notation and write $z = x$ and hence the posterior is,

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x|x_{t-1}) \tag{19}$$

Restriction II lets me encoder as, $q(x_t|x_{t-1} = N(x_t; \sqrt{\alpha_t}\, x_{t-1}, (1-\alpha_t)I)$ And the decoder as (Restriction III),

$$p(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x) \text{ and } p(x_T) = N(x_T; 0, I) \tag{20}$$

IMPLEMENTATION

- ## Noise Scheduler

  We have precomputed values of alpha and beta to predict posterior variances and thus create noise distributions.

- ## Timestep Embedding

  We create embeddings of timesteps into the denoising neural network in-order to make the model understand the extent of noise at a given timestep.

- ## Autoencoder for denoising

  VAE - UNET used for learning the reverse (denoising) process efficiently and have a rich pool of latent variables.

- ## Sampling

  We curate a sampler method that can show the model's noise prediction and denoising process.

NOISE SCHEDULER

```
def forward_diffusion_sample(x_0, t, device="cpu"):
    noise = torch.randn_like(x_0)
    sqrt_alphas_cumulative_products_t = get_index_from_list(sqrt_alphas_cumulative_products, t, x_0.shape)
    sqrt_one_minus_alphas_cumulative_products_t = get_index_from_list(
        sqrt_one_minus_alphas_cumulative_products, t, x_0.shape
    )
    ## formulae for image augged looks like sqrt(pi(alpha_t)) * x_t-1 * sqrt(pi(1-alpha_t)) * noise~N(0,1)
    return sqrt_alphas_cumulative_products_t.to(device) * x_0.to(device) \
        + sqrt_one_minus_alphas_cumulative_products_t.to(device) * noise.to(device), noise.to(device)
```

```
### SOO MMANNYY PRECOMPUTEDD VALUESS TO TRACKKKKSS
betas = torch.linspace(1e-4, 0.02, T)
alphas = 1. - betas
alphas_cumulative_products = torch.cumprod(alphas, axis=0)
alphas_cumulative_products_prev = F.pad(alphas_cumulative_products[:-1], (1, 0), value=1.0)
sqrt_recip_alphas = torch.sqrt(1.0 / alphas)
sqrt_alphas_cumulative_products = torch.sqrt(alphas_cumulative_products)
sqrt_one_minus_alphas_cumulative_products = torch.sqrt(1. - alphas_cumulative_products)
posterior_variance = betas * (1. - alphas_cumulative_products_prev) / (1. - alphas_cumulative_products)
```

```
def get_loss(self, x_0, t):
    x_noisy, noise = forward_diffusion_sample(x_0, t, self.device)
    noise_pred = self(x_noisy, t)
    return F.l1_loss(noise, noise_pred)
```

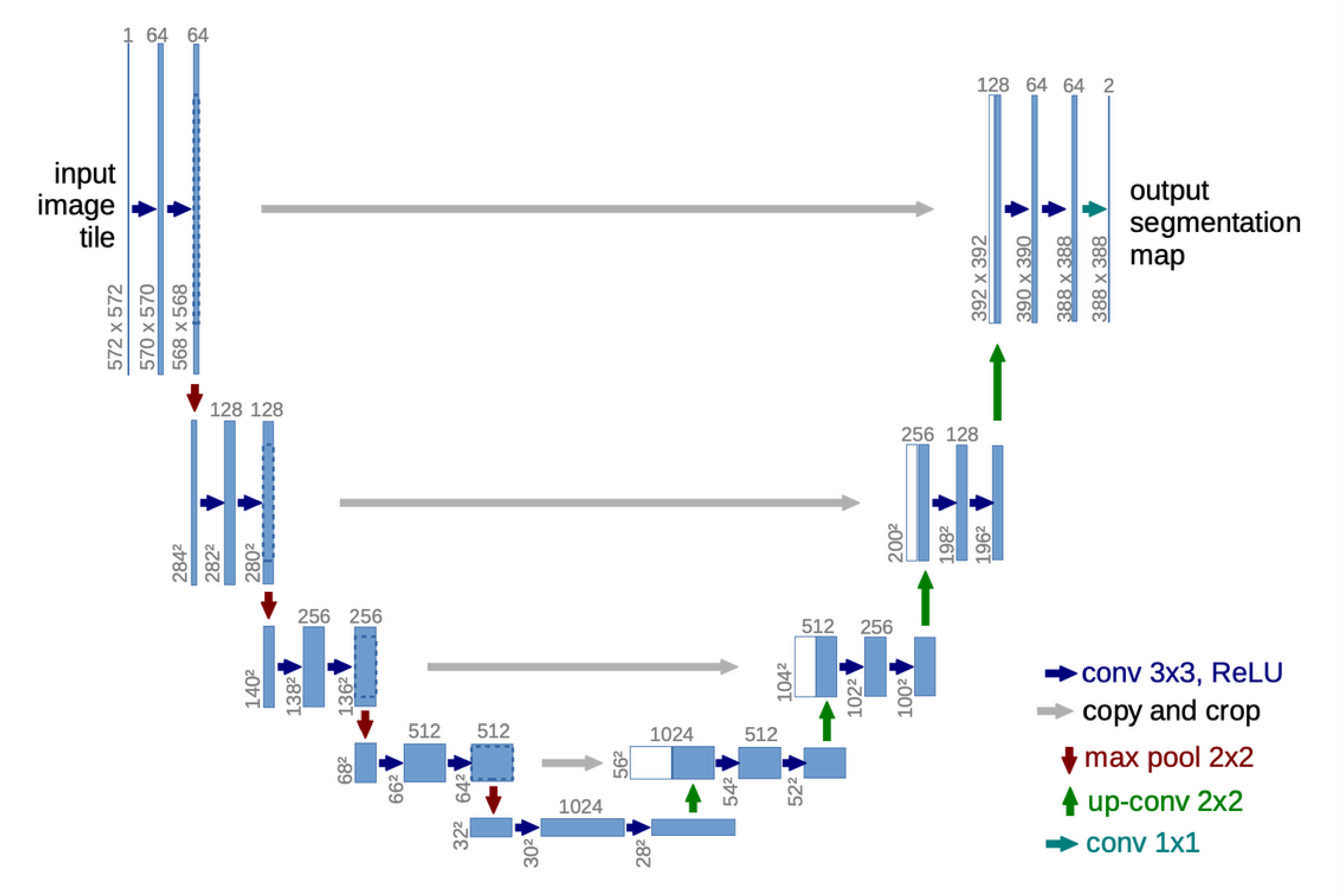
KIIT DU | Harshit Agarwal

AUTOENCODER FOR DENOISING

KIIT DU | Harshit Agarwal

```python
class Block(nn.Module):
  def __init__(self, in_ch, out_ch, time_emb_dim, up=False):
    super().__init__()
    self.time_mlp = nn.Linear(time_emb_dim, out_ch)
    if up:
      ## up channel - go big big big bigg from smol smol smol with 3x3 kernel
      self.conv1 = nn.Conv2d(2*in_ch, out_ch, 3, padding=1)
      self.transform = nn.ConvTranspose2d(out_ch, out_ch, 4, 2, 1)
    else:
      self.conv1 = nn.Conv2d(in_ch, out_ch, 3, padding=1)
      self.transform = nn.Conv2d(out_ch, out_ch, 4,2,1)
    self.conv2 = nn.Conv2d(out_ch, out_ch, 3, padding=1)
    self.relu = nn.ReLU()
    self.batch_norm1 = nn.BatchNorm2d(out_ch)
    self.batch_norm2 = nn.BatchNorm2d(out_ch)

  def forward(self, x, t, ):
    h = self.batch_norm1(self.relu(self.conv1(x)))
    time_emb = self.relu(self.time_mlp(t))
    time_emb = time_emb[(..., ) + (None, ) * 2]
    h = h + time_emb
    h = self.batch_norm2(self.relu(self.conv2(h)))
    return self.transform(h)
```



AUTOENCODER FOR DENOISING

KIIT DU | Harshit Agarwal

```python
class SimpleUnet(nn.Module):
    def __init__(self):
        super().__init__()
        image_channels = 3
        down_channels = (64, 128, 256, 512, 1024)
        up_channels = (1024, 512, 256, 128, 64)
        self.device = "cuda" if torch.cuda.is_available() else "cpu"

        out_dim = 3
        time_emb_dim = 32

        ## timestep stored as positional encoding in terms of sine
        self.time_mlp = nn.Sequential(
                PositionEmbeddings(time_emb_dim),
                nn.Linear(time_emb_dim, time_emb_dim),
                nn.ReLU()
            )

        self.conv0 = nn.Conv2d(image_channels, down_channels[0], 3, padding=1)
        self.down_blocks = nn.ModuleList([
            Block(down_channels[i], down_channels[i+1], time_emb_dim)
            for i in range(len(down_channels)-1)
        ])
        self.up_blocks = nn.ModuleList([
            Block(up_channels[i], up_channels[i+1], time_emb_dim, up=True)
            for i in range(len(up_channels)-1)
        ])

        ## readout layer
        self.output = nn.Conv2d(up_channels[-1], out_dim, 1)
```

```python
@torch.no_grad()
def sample(self, noise):
    """
    Generate an image by denoising a given noise tensor using the reverse diffusion
    
    Args:
        noise (torch.Tensor): Initial noise tensor (e.g., sampled from a Gaussian d

    Returns:
        torch.Tensor: Denoised image.
    """
    img = noise  # Start with the provided noise tensor
    T = self.num_timesteps  # Total timesteps for diffusion
    stepsize = 1  # You can adjust if needed

    # Iterate through the timesteps in reverse order
    for i in range(0, T)[::-1]:
        t = torch.full((noise.size(0),), i, device=noise.device, dtype=torch.long)
        img = sample_timestep(self, img, t)  # Perform one reverse diffusion step
        img = torch.clamp(img, -1.0, 1.0)  # Clamp the image to ensure values stay

    return img
```

```python
def forward(self, x, timestep):
    t = self.time_mlp(timestep)
    x = self.conv0(x)
    residual_inputs = []
    for down in self.down_blocks:
        x = down(x, t)
        residual_inputs.append(x)
    for up in self.up_blocks:
        residual_x = residual_inputs.pop()
        x = torch.cat((x, residual_x), dim=1)
        x = up(x, t)
    return self.output(x)
```

TIMESTEP EMBEDDING


```
class PositionEmbeddings(nn.Module):
  def __init__(self,dim):
    super().__init__()
    self.dim = dim

  def forward(self, time):
    device = time.device
    half_dim = self.dim // 2
    embeddings = math.log(10000) / (half_dim - 1)
    embeddings = torch.exp(torch.arange(half_dim, device=device) * -embeddings)
    embeddings = time[:, None] * embeddings[None, :]
    embeddings = torch.cat((embeddings.sin(), embeddings.cos()), dim=-1)
    return embeddings
```

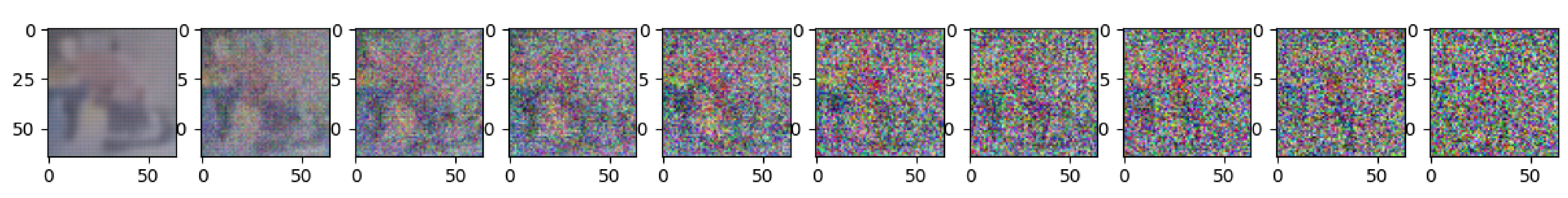
```
time_emb_dim = 32

## timestep stored as positional encoding in terms of sine
self.time_mlp = nn.Sequential(
    PositionEmbeddings(time_emb_dim),
    nn.Linear(time_emb_dim, time_emb_dim),
    nn.ReLU()
)
```

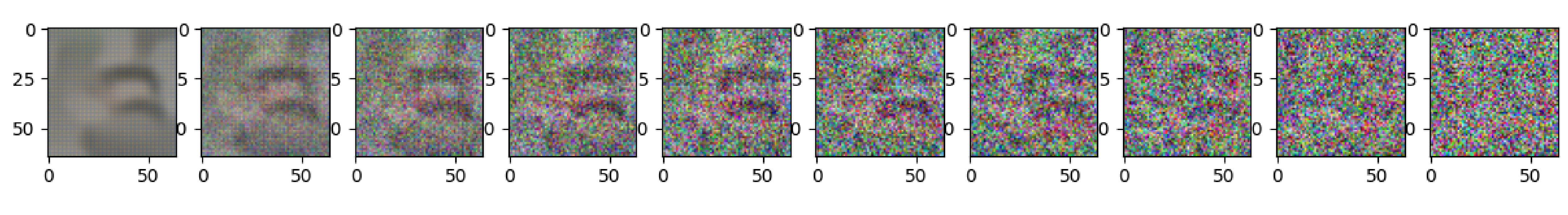
RESULT

10 step samples from the 5th training epoch of the model

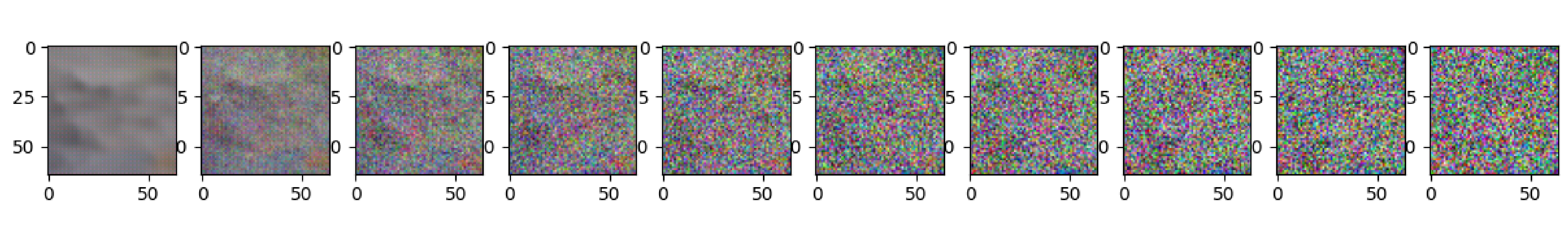
Simple Unet



Self Attention



CBAM



CONCLUSION

Denoising: process of removing noise from an image progressively to learn image features and meaning.

Diffusion: process of adding gaussian noise to images progressively that balance images

Variance: scheduling noise to be progressively more variant.

QUESTIONS AND SUGGESTIONS

- ## Gaussian Blur

  For smaller images, gaussian noise performs nice. But there are many more ways to create "noise". Try implementing a forward noise using gaussian blur.

- ## Better placement of CBAM and Attention Gates

  Currently the CBAM and Attention Gates are at bottleneck, there's a much smarter and better position for them to be placed at in the unet.

- ## Gradio Inference Implementation

  Implement an API service and a Gradio Inference page in the learndiffusion website for loading and generating images

QUESTIONS AND SUGGESTIONS

- Improve Placement of CBAM and Attention Gates in U-Net `enhancement`
  #4 · aharshit123456 opened 45 minutes ago
- Replace Gaussian Noise with Gaussian Blur for Forward Diffusion `enhancement`
  #3 · aharshit123456 opened 47 minutes ago
- Errors in the sampling function, boolean errors and etc. `bug` `good first issue`
  #2 · aharshit123456 opened 19 hours ago
- Add Gradio Inference for the model on the learndiffusion website. `enhancement`
  #1 · aharshit123456 opened 19 hours ago

KIIT Deemed University
Presentation for
Microsoft Learn Student Ambassador

THANK YOU

Harshit Agarwal | Jan 31st