Diffusion Models, Stable Diffusion, and CLIP

Volker Tresp

Winter 2024-2025

Diffusion Models

DDPM: Denoising Diffusion Probabilistic Model

- Assume a $P(\mathbf{h}, \mathbf{x}) = P(\mathbf{x}|\mathbf{h})P(\mathbf{h}) = P(\mathbf{h}|\mathbf{x})P(\mathbf{x})$
- $\mathbf{x}$ and $\mathbf{h}$ are both M-dimensional
- Simple marginalization give us the generator

$$P(\mathbf{x}) = \int P(\mathbf{x}|\mathbf{h})P(\mathbf{h})d\mathbf{h}$$

- Simple marginalization gives us an encoder

$$P(\mathbf{h}) = \int P(\mathbf{h}|\mathbf{x})P(\mathbf{x})d\mathbf{x}$$

- We can approximate $P(\mathbf{x})$ with the training images
- We can then generate samples from $P(\mathbf{h}|\mathbf{x}_i)$ with any reasonable conditional distribution

DDPM: Denoising Diffusion Probabilistic Model (cont'd))

- Based on the $(\mathbf{h}^s, \mathbf{x}^s)$ samples, we could learn a $P(\mathbf{x}|\mathbf{h})$; here we can use supervised learning since inputs and outputs are defined by the samples
- This does not work very well since most $\mathbf{x}|\mathbf{h}$ would not Ã$\frac{1}{4}$produce meaningful images

A Markov Chain

- We start with a forward Markov chain

$$P(\mathbf{h}_1, \ldots, \mathbf{h}_T, \mathbf{x}) = P(\mathbf{x})P(\mathbf{h}_1|\mathbf{x}) \prod_{t=2}^{T} P(\mathbf{h}_t|\mathbf{h}_{t-1})$$

- The trick is to design (not learn) a very simple Markov chain (now $\mathbf{h} \equiv \mathbf{h}_T$); then

$$P(\mathbf{h}_T|\mathbf{x}) = \int P(\mathbf{h}_1|\mathbf{x}) \prod_{t=2}^{T} P(\mathbf{h}_t|\mathbf{h}_{t-1})\, d\mathbf{h}_1 \ldots \mathbf{h}_T$$

  This is the forward model (encoder)

- We assume an extremely simple model

$$P(\mathbf{h}_t|\mathbf{h}_{t-1}) = \mathcal{N}(\mathbf{h}_t; (1-\beta_t)\mathbf{h}_{t-1}, \beta_t \mathbf{I})$$

- The noise variance is $0 < \beta_1 < \beta_2 \ldots < \beta_T \leq 1$

- We get that $P(\mathbf{h}_T|\mathbf{x}) = \mathcal{N}(\mathbf{h}_T|0, \mathbf{I})$, thus we get our simple Gaussian distribution back for $\mathbf{h}_T$
- From the chain, we generate samples $(\mathbf{h}_1^s, \ldots, \mathbf{h}_T^s, \mathbf{x}^s)$ (where $\mathbf{x}^s$ is a training image)

The Markov Chain Backwards

- We now consider the backward Markov chain

$$P(\mathbf{h}_1, \ldots, \mathbf{h}_T, \mathbf{x}) = P(\mathbf{x}|\mathbf{h}_1) \prod_{t=2}^{T} P(\mathbf{h}_{t-1}|\mathbf{h}_t)$$

- We impose a very simple conditional backward model by learning

$$P(\mathbf{h}_{t-1}|\mathbf{h}_t) = \mathcal{N}(\mathbf{h}_{t-1}; \vec{\mu}(\mathbf{h}_t, t), \beta_t \mathbf{I})$$

  $\vec{\mu}(\mathbf{h}_t, t)$ is a DNN with M inputs and M outputs; the larger T (can be a few thousands), the better the approximation

- $\vec{\mu}(\mathbf{h}_t, t)$ can be learned from the samples generated in the forward path

- After learning convergences, we can sample from the backward path and generate images!

- The generator becomes

$$P(\mathbf{x}|\mathbf{h}_T) = \int P(\mathbf{x}|\mathbf{h}_1) \prod_{t=2}^{T} P(\mathbf{h}_{t-1}|\mathbf{h}_t)\, d\mathbf{h}_1 \ldots \mathbf{h}_T$$

Direct Models

- Given the encoder, the conditional probabilities of the decoder are not really Gaussian; the Gaussian approximation can be motivated by a variational approximation

- We can derive for the *direct forward model*

$$P(\mathbf{h}_t|\mathbf{x}) = \mathcal{N}(\mathbf{h}_t; \sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbf{I})$$

  with $\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$ (α_t is close to 1 for small t and gets closer to 0 for large t)

- For the *direct backward model*, we use

$$P(\mathbf{x}|\mathbf{h}_t) = \mathcal{N}(\mathbf{x}; \mathbf{h}_t - \mathbf{g}(\mathbf{h}_t, t), (1-\alpha_t)\mathbf{I})$$

- Note that the expression $\mathbf{h}_t - \mathbf{g}(\mathbf{h}_t, t)$ can be thought of as a classical generator; for a $t < T$ the generator becomes more local

- $\mathbf{g}(\mathbf{h}_T, T)$ is approximated by a U-net

A Direct Backward Model

- We actually model $\mathbf{g}(\mathbf{h}_t, t)$; the relationship is

$$\vec{\mu}(\mathbf{h}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{h}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{h}_t, t)\right)$$

- In the pseudocode $\mathbf{h} \equiv \mathbf{z}$

Algorithm 20.1: Training a denoising diffusion probabilistic model

```
Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
       Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$
----------------------------------------------------------------
for $t \in \{1, \ldots, T\}$ do
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alphas from betas
end for
repeat
    $\mathbf{x} \sim \mathcal{D}$ // Sample a data point
    $t \sim \{1, \ldots, T\}$ // Sample a point along the Markov chain
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}$ // Evaluate noisy latent variable
    $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ // Compute loss term
    Take optimizer step
until converged
return $\mathbf{w}$
```

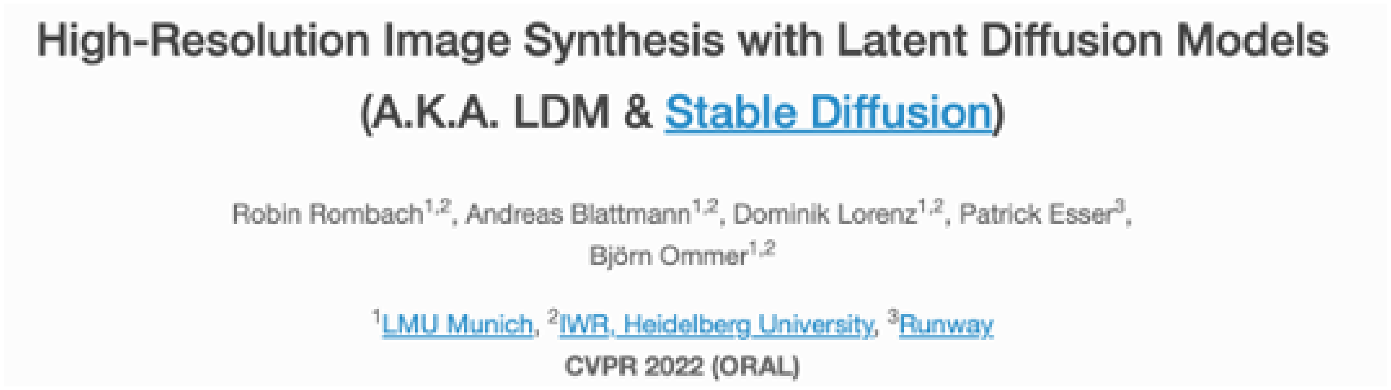
Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

```
Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
       Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ // Sample from final latent space
for $t \in T, \ldots, 2$ do
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alpha
    // Evaluate network output
    $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
    $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ // Add scaled noise
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ // Final denoising step
return $\mathbf{x}$
```

Stable Diffusion

Stable Diffusion

- Stable diffusion models, also known as latent diffusion models or LDMs
- Use a VAE in a preprocessing step to map images to latent spaces and back
- The diffusion model is applied to the latent state defined by the VAE; let $\mathbf{h}(\mathbf{x})$ be the $M_{hidd} < M$-dimensional latent representation generated by the VAE
- $\mathbf{h}(\mathbf{x})$ assumes the role of the $\mathbf{x}$ before; all $\mathbf{h}_t$ also have dimension M_{hidd}
- $\mathbf{g}(\mathbf{h}_t, t)$ is modelled as a U-net
- To provide information about the prompt, a BERT model provides an embedding vector that is then used in form of cross attention in the U-net
- So we have both the VAE encoder-decoder and the diffusion encoder-decoder



High-Resolution Image Synthesis with Latent Diffusion Models (A.K.A. LDM & Stable Diffusion)

Robin Rombach[1,2], Andreas Blattmann[1,2], Dominik Lorenz[1,2], Patrick Esser[3], Björn Ommer[1,2]

[1]LMU Munich, [2]IWR, Heidelberg University, [3]Runway

CVPR 2022 (ORAL)

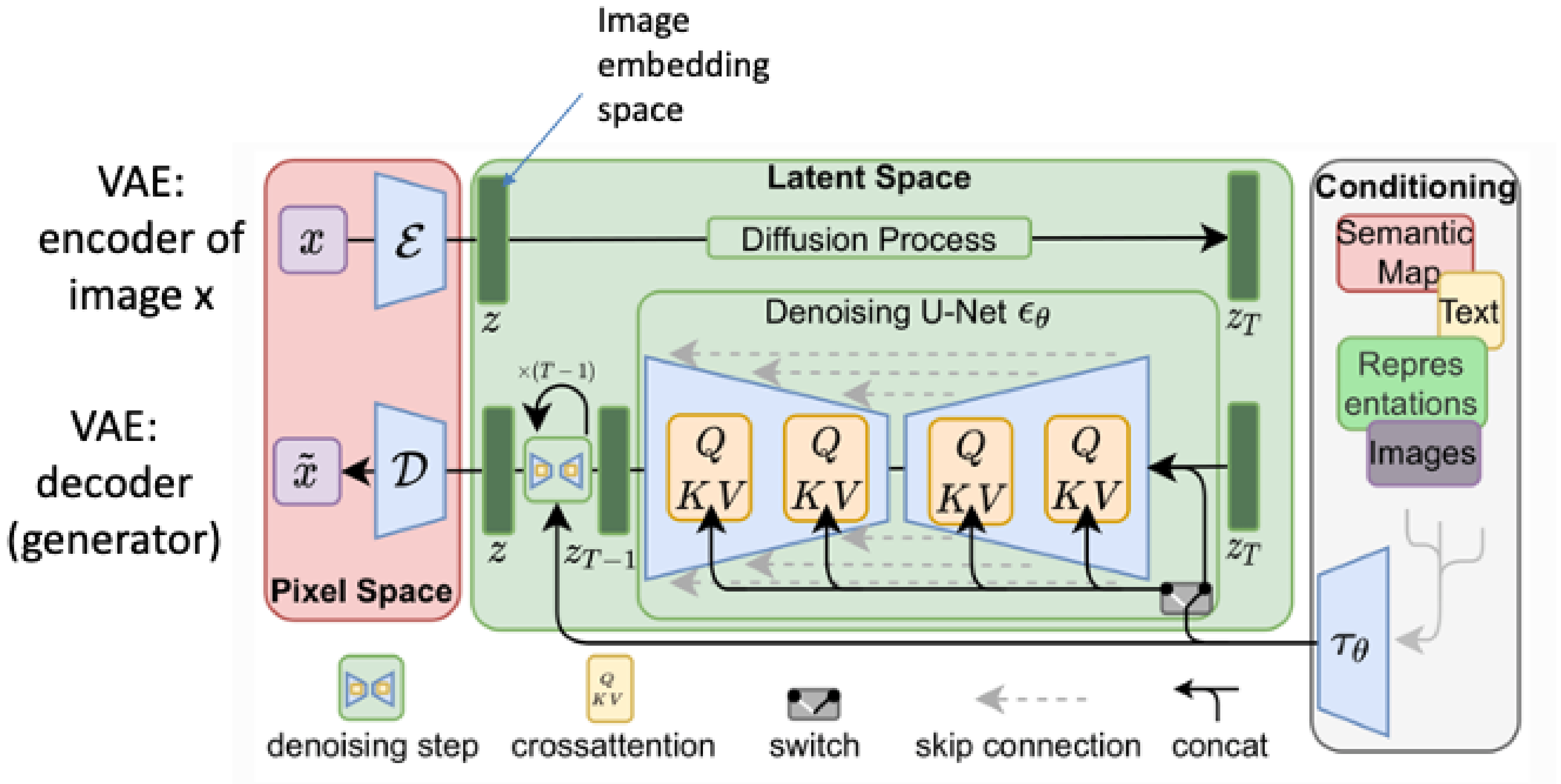
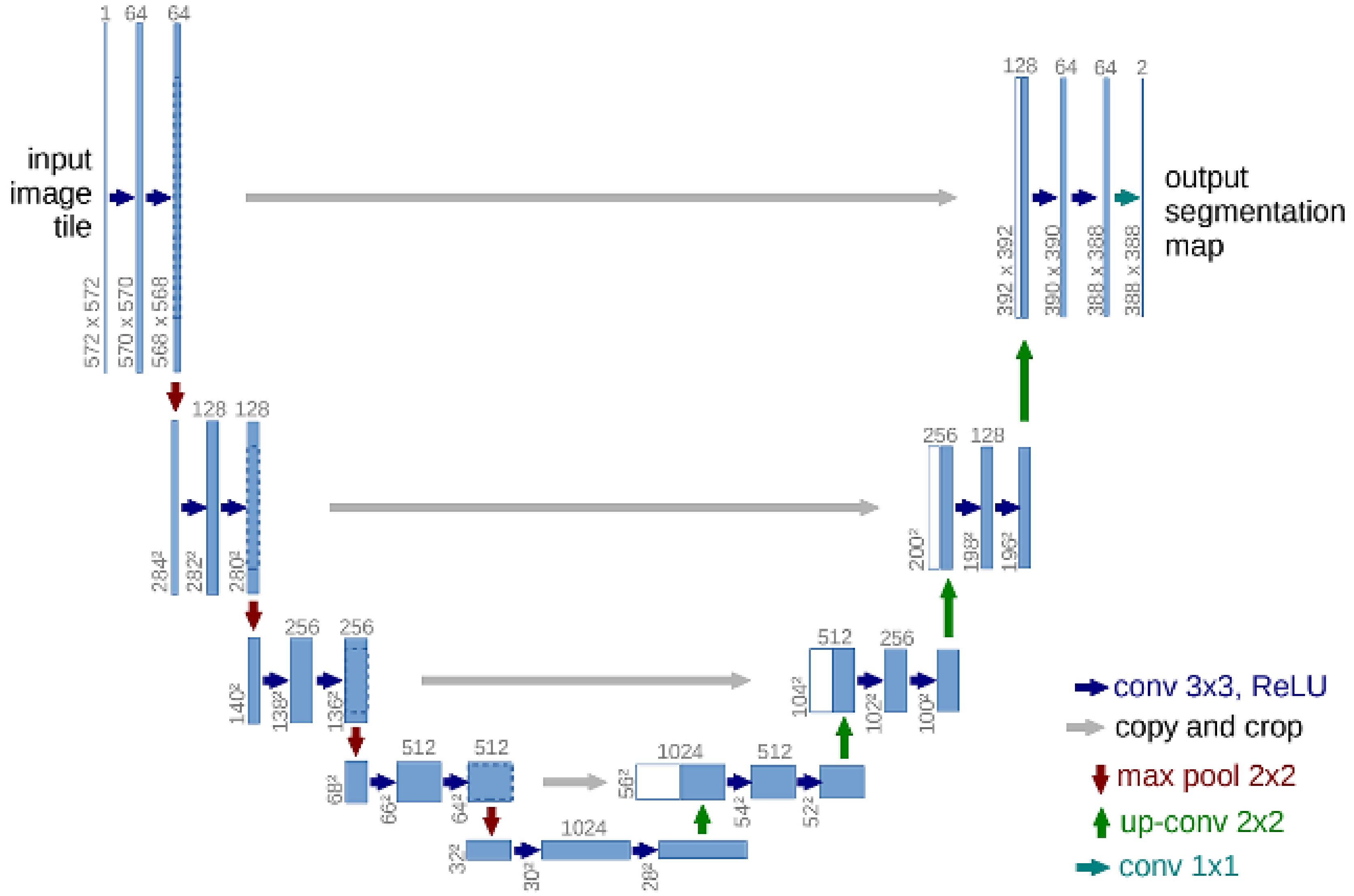
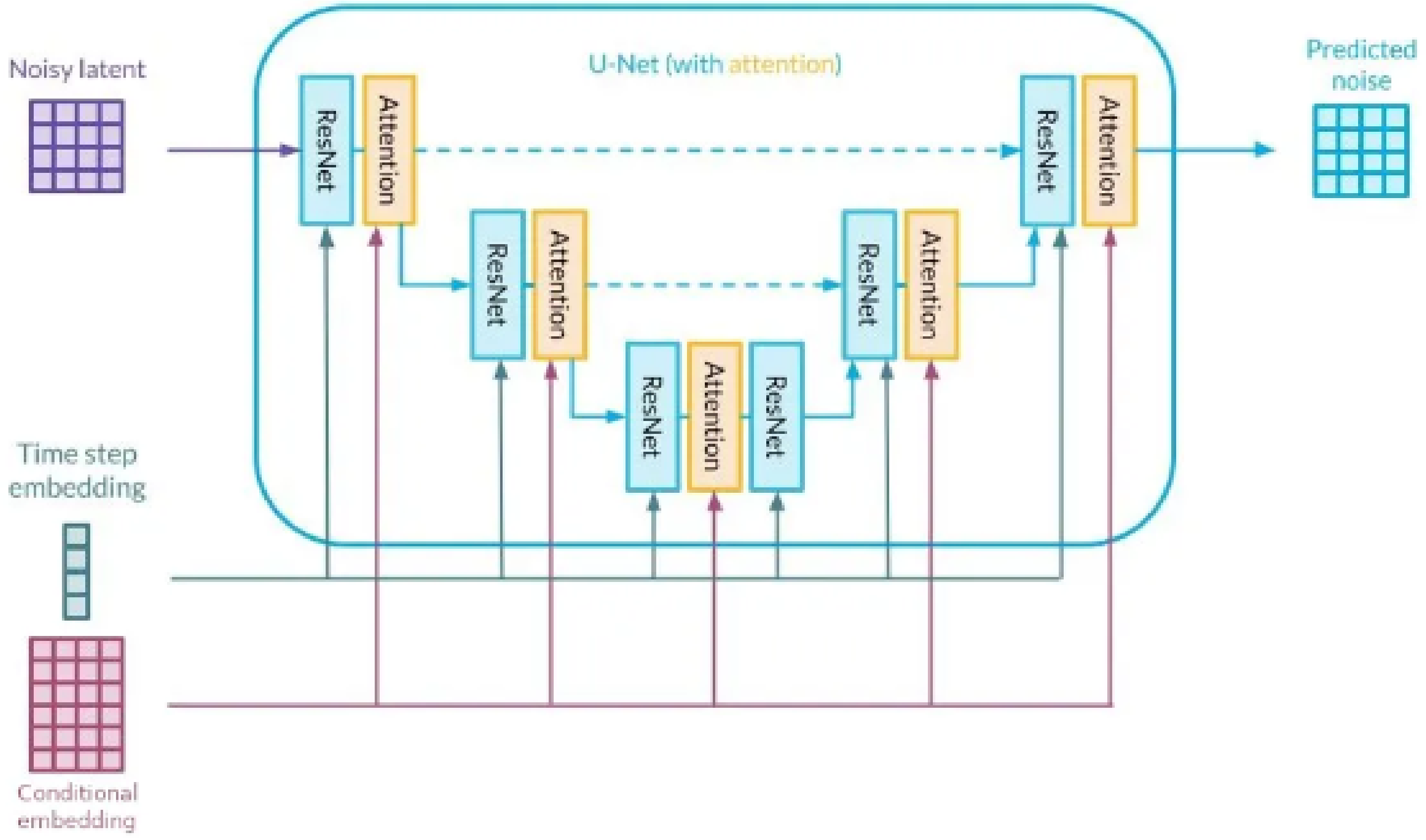


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism.

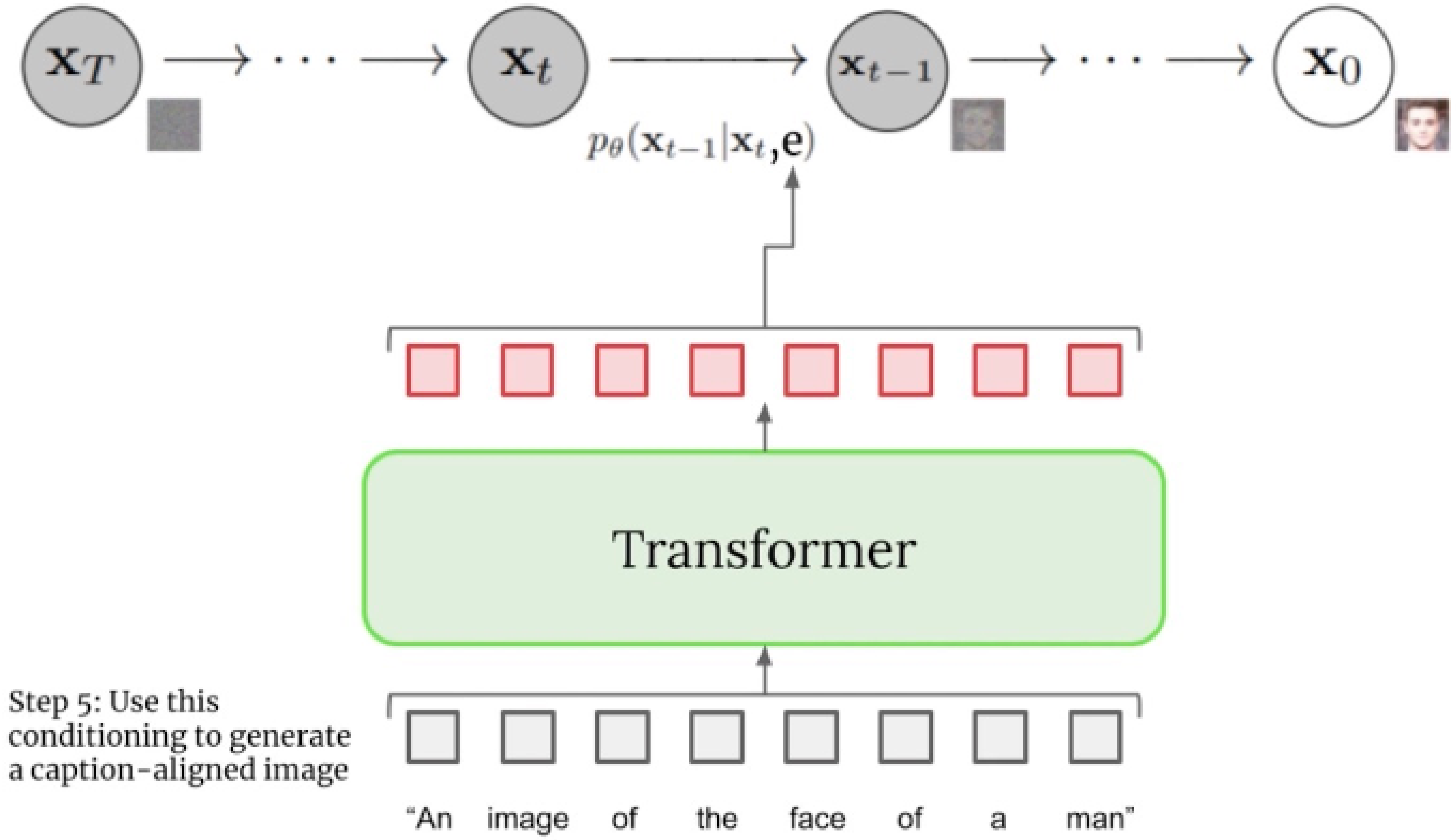
U-net



1
64
64
input image tile
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
56²
512
54²
52²
512
104²
256
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1



Noisy latent
U-Net (with attention)
Predicted noise
ResNet
Attention
Time step embedding
Conditional embedding


$\mathbf{x}_T$
$\mathbf{x}_t$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{e})$
Transformer
Step 5: Use this conditioning to generate a caption-aligned image
"An image of the face of a man"

CLIP

Contrastive Pretraining Using Clip

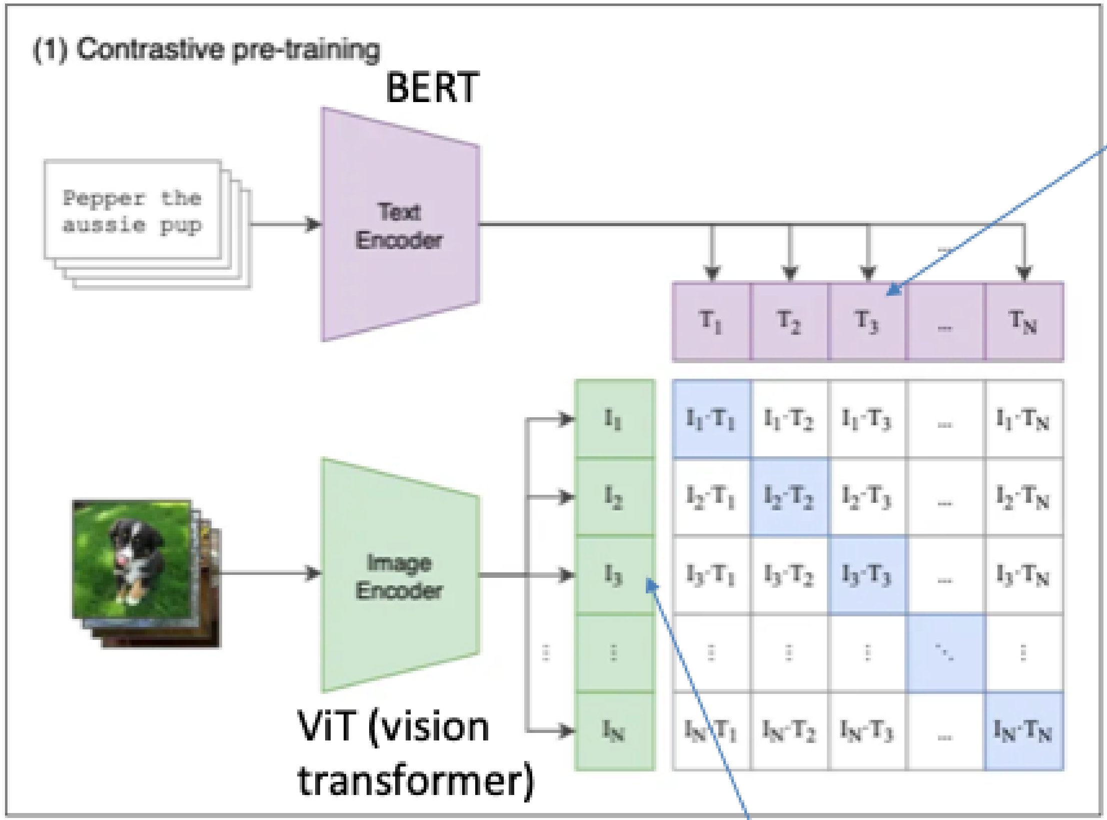
- The BERT model is pretrained on image annotations from the web using contrastive learning; it enforces similar embeddings for matching text-image pairs
- Cost function, e.g.,

$$\mathrm{softmax}((\mathbf{h}_i^{text})^T \mathbf{h}_i^{image})$$

  $\mathbf{h}_i^{text}$ is the text embedding vector generated from BERT and $\mathbf{h}_i^{image}$ is the image embedding vector generated, e.g., from a vision transformer (ViT)

- Often cosine distance is used instead of inner product
- Use embedding vector pairs (image-text) and mismatches of texts and mismatches of images (batch size for the softmax: e.g., 2000)

Contrastive Pre-Training using CLIP



(1) Contrastive pre-training
BERT
Pepper the aussie pup
Text Encoder
T_1 T_2 T_3 ... T_N
Image Encoder
I_1 I_2 I_3 ⋮ I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ ... $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ ... $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ ... $I_N \cdot T_N$
ViT (vision transformer)
Embedding vector of text-3
Embedding vector of image-3

Figure 1: Contrastive Pre-training step of CLIP (Source)

$$\mathcal{L} = -\frac{1}{2N}\sum_{i=1}^{N}\left(\overbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{N}e^{t\mathbf{x}_i\cdot\mathbf{y}_j}}}^{\text{image}\rightarrow\text{text softmax}} + \overbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{N}e^{t\mathbf{x}_j\cdot\mathbf{y}_i}}}^{\text{text}\rightarrow\text{image softmax}}\right)$$

Every positive pair is normalized by **all** negative pairs

Equation 1: CLIP uses softmax operation. Accordingly, the similarity of every positive-pair is normalized by *all* negative pairs. Thus, every GPU makes maintains an NxN matrix for all pairwise similarities. This brings quadratic complexity to CLIP.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\log \frac{1}{\underbrace{1+e^{z_{ij}(t\mathbf{x}_i\cdot\mathbf{y}_j+b)}}_{\text{Every pair (positive/negative) is independent of other pairs}}} \quad \text{s.t.} \quad z_{ij} = \begin{cases} 1, & \text{for positive pairs.} \\ -1, & \text{for negative pairs.} \end{cases}$$

Equation 2: SigLIP uses sigmoid operation and each image-text pair (positive or negative) is evaluated independently. There is no need to maintain a global NxN normalization matrix. Accordingly, SigLIP loss can be evaluated incrementally for large batch-sizes.

Concluding the Lecture

- The lecture was quite comprehensive
- We did not cover Deep Reinforcement Learning
- We did not cover Structured Distributions: Bayes nets, Markov nets
- We did not cover data on graphs (e.g., social networks), e.g., Graph Neural Networks