Assignment 2
COL775: Deep Learning. Semester II, 2023-2024.
Due Date: ~~May 6, 2024~~ May 7, 2024

April 5, 2024

The dataset for this assignment can be found from `/scratch/cse/dual/cs5190448/A2_dataset` on HPC.

PART 1

1 Object-Centric Learning With Slot Attention

Neural models are considered as black box methods since they often work with raw data as input, and implement complex function manipulations over them, to get the final output, without much interpretability of what happens at intermmediate steps. For image related tasks, recent literature [**?**] has discovered that having object-centric representation in latent space not only improves the interpretability but also improves the performance of models. Slot attention [3] is one of the methods to learn object centric representations from images. In this assignment you will implement slot attention from scratch.

1.1 Slot Attention

Slot attention was proposed in the paper [3]. And is defined as follows: "The Slot Attention module maps from a set of N input feature vectors to a set of K output vectors that are refer to as slots. Each vector in this output set can, for example, describe an object or an entity in the input. Slot Attention uses an iterative attention mechanism to map from its inputs to the slots. Slots are initialized at random and thereafter refined at each iteration t = 1 . . . T to bind to a particular part (or grouping) of the input features." Read the original slot attention paper [3] for more information. A pseducode for algorithm is shown in figure 1.

- More formally given image $x \in \mathbf{R}^{H \times W \times 3}$, a CNN encoder $\mathcal{E}$ gives feature maps $\mathcal{E}(x) = z \in \mathbf{R}^{\sqrt{N} \times \sqrt{N} \times D_{inputs}}$.

Algorithm 1 Slot Attention module. The input is a set of N vectors of dimension $D_{\texttt{inputs}}$ which is mapped to a set of K slots of dimension $D_{\texttt{slots}}$. We initialize the slots by sampling their initial values as independent samples from a Gaussian distribution with shared, learnable parameters $\mu \in \mathbb{R}^{D_{\texttt{slots}}}$ and $\sigma \in \mathbb{R}^{D_{\texttt{slots}}}$. In our experiments we set the number of iterations to $T = 3$.

```
1:  Input: inputs $\in \mathbb{R}^{N \times D_{\texttt{inputs}}}$, slots $\sim \mathcal{N}(\mu,\ \mathrm{diag}(\sigma)) \in \mathbb{R}^{K \times D_{\texttt{slots}}}$
2:  Layer params: $k,\ q,\ v$: linear projections for attention; GRU; MLP; LayerNorm (x3)
3:    inputs = LayerNorm (inputs)
4:    for $t = 0 \ldots T$
5:      slots_prev = slots
6:      slots = LayerNorm (slots)
7:      attn = Softmax ($\frac{1}{\sqrt{D}} k$(inputs) $\cdot$ $q$(slots)$^T$, axis='slots')        # norm. over slots
8:      updates = WeightedMean (weights=attn + $\epsilon$, values=$v$(inputs))        # aggregate
9:      slots = GRU (state=slots_prev, inputs=updates)        # GRU update (per slot)
10:     slots += MLP (LayerNorm (slots))        # optional residual MLP (per slot)
11:   return slots
```

Figure 1: Pseudo Code for slot attention

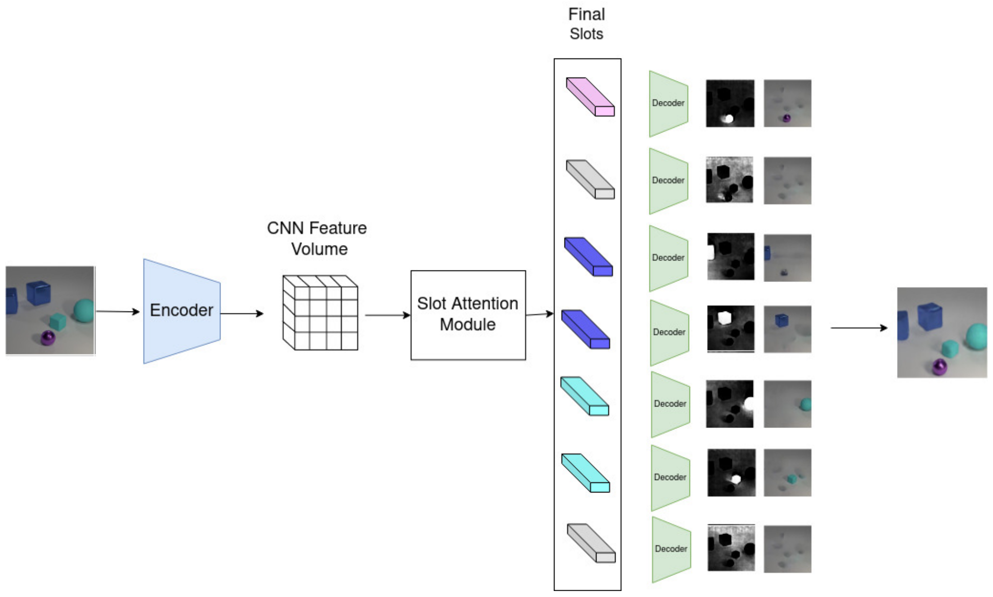

Figure 2: High level diagram

- A 2D positional encoding $P \in \mathbf{R}^{\sqrt{N}\times\sqrt{N}\times D_{inputs}}$ is added to z,

$$z' = z + P$$

- z' is flattened spatially to get N input vectors. These are passed through `MLP` to get final `inputs` $\in \mathbf{R}^{N\times D_{inputs}}$

$$\texttt{inputs} = \texttt{MLP}(\texttt{Flatten}(z'))$$

- Then K slot vectors, `slots` $\in \mathbf{R}^{K\times D_{slots}}$ are initialized by sampling them from normal distibution $\mathcal{N}(\mu, \text{diag}(\sigma))$, where both μ and σ are learnable. Optionally on complex datasets it has been found beneficial to use learnable initialization of slots, that is you initialize slots with learnable K vectors rather than sampling. [10]
- Then `slots` attend to `inputs` itertively for T times and updated `slots` at the end of each iteration. For detailed steps refer to pseudo code 1.
- Each of K vectors from `slots` are decoded with **Spatial Broadcast Decoder** [9], $\mathcal{D}$ to produce 4 channeled image. Denote $\texttt{output}_i = \mathcal{D}(\texttt{slots}[i])$, note that $\texttt{output}_i \in \mathbf{R}^{H\times W\times 4}$
- To get the final image, first K masks are generated one for each slot by taking softmax for each pixel of first channel, across slots.

$$\texttt{masks}_i = \texttt{Softmax}(\{\texttt{output}_j[:,:,0]\}_{j=1}^{K})_i$$

- The final reconstructed image is obtained as

$$\hat{x} = \sum_{i=1}^{K} \texttt{masks}_i \cdot \texttt{content}_i$$

  where $\texttt{content}_i = \texttt{output}_i[:,:,1:]$
- You backpropagate on the image reconstruction loss for batch of size B

$$\mathcal{L}(\theta) = \frac{1}{BHW}\sum_{i=1}^{B}(x^{(i_b)} - \hat{x}^{(i_b)})^2$$

These are spatially flattened to get N input vectors `inputs` $\in \mathbf{R}^{N\times D_{inputs}}$.

1.2 Experiments

You're provided with a sub-sampled CLEVRTex dataset which comprises of synthetic images of objects with different attributes, locations and properties. Train the Slot-Attention based network as shown in 2 on this data and visualize the generated masks and the reconstructed images. **We recommend using similar hyper-parameters to [3] [10]. However you're free to experiment with other values as well.** Use number of slots $K = 11$ for this experiment.

- Report the Adjusted Rand Index (ARI) score between the ground-truth and predicted object segmentation masks on the val split.
- Plot the train and val image reconstruction loss vs epochs. To save training time you may choose validate after certain number of epochs rather than every epoch.
- **Compositional Generation**: Create a slot library using the training data, apply K-means clustering with K being the number of slots in the trained model. Sample new slots from each of the cluster to generate an image with compositional attributes. Generate N_{val} such images, where N_{val} is number of validation images and report the clean-fid [4] metric using the validation images as ground truth.

PART 2

2 Slot Learning using Diffusion based Decoder

In this part you'll implement a conditional diffusion model to decode the image conditioned on the slots, to replace Spatial Broadcast Decoder and again learn the object-centric slots keeping the hyper-parameters for the encoder and other modules same.

2.1 Diffusion Models

Diffusion models[2] are a class of generative models which learns the data-distribution through a fixed noising process and a learned reverse process to denoise the image. In this part we'll focus on a sub-category of diffusion models known as Latent space Diffusion model (LDM)[5] which trains the model in the latent space of a pretrained Variational AutoEncoder(VAE)[7] to lower the computational requirement by compressing the input space. The LDM is trained to generate the image latent (obtained by pretrained VAE) conditioned upon text but here you will condition it upon the slots. We provide the checkpoint of a pre-trained VAE and the inference script here.

2.2 Implementation

Training a slot conditioned diffusion model requires the following components, you're required to implement them from scratch in pytorch. You can read more about the implementation in [10]

2.2.1 Slot Encoder

This module is same as in 1.1. Given image x and trainable CNN based encoder $\mathcal{E}$, $\mathcal{E}(x)$ will give input feature vectors and slot attention module will give final `slots` given the input features of x.

2.2.2 Diffusion Decoder

Given an image $x \in \mathbf{R}^{H\times W\times 3}$ and it's corresponding $\texttt{slots} \in \mathbf{R}^{K\ D_{slots}}$, the goal of the diffusion model is to generate latent $z \in \mathbf{R}^{P\times P\times C}$, which is encoding of x through $z = \texttt{VAE}(x)$. Formally if $\mathcal{D}$ denotes diffusion model then $\mathcal{D}(x, \texttt{slots}) = \hat{z}$. Note that during training we train slot encoder and diffusion model **jointly** and `VAE` need to be **freezed**.

- **Forward-Process :** The training latents z (since we're working on LDM) are gradually corrupted with Gaussian noise with a variance schedule (β_1 ... β_T), implement a linear schedule which starts at $\beta_1 = 0.0015$ and ends at $\beta_T = 0.0195$ with 1000 steps, i.e $T = 1000$

- **Reverse-Process :** In the reverse process the diffusion model generates latent $\hat{z}$ given `slots`. It does so by denoising the $z_0 \sim \mathcal{N}(I, 0)$ iteratively for $T = 1000$ steps. Architecturally Diffusion models use a modified Unet architecture [6] to learn the denoising of latent z_{t-1}, to generate z_t. The Unet will comprise of a Residual Block (used for downsampling and upsampling) and a Transformer Block (used for slot conditioning) as shown in figure 3 which will be used in both downsampling and upsampling parts of UNet. A detailed architecture for the Unet Model can be found 5

  - *Residual Block*: Figure 4 shows detailed architecture of residual block. Note that residual block is used in both downsampling and upsampling part of UNet. To reduce the learnable parameters we can replace the first convolution layer of the residual blocks with different non parametric operations depending on whether the residual block is used for downsampling or upsampling.

    * *Downsample ResBlock:* In case of the downsampling replace the first convolution layer with average pooling with stride.
    * *Upsample ResBlock:* In case of upsampling replace first convolution layer with interpolation operation. [1]
    * In neither of case (that is output and input have same shape) keep the convolution layer with same padding.

  - *Transformer Block*: To condition the denoising on `slots`, cross attention with `slots` layers are added to UNet. These are called transformer block. With similar architecture to transformer decoder in [8]. This takes the slots from the slot-attention model and the intermediate latent of UNet and applies self-attention, followed by cross-attention as shown in figure 4. These attentions are multi-headed.

- **Decoding (Generation):** Implement Ancestral Sampling which is just iterative denoising of z_t to generate z_{t+1}. Start with a random noise z_0 of shape same as latent, iteratively run the noise through the unet from $T = 1000$ to $T = 0$. Refer to Algorithm 2 in [2]

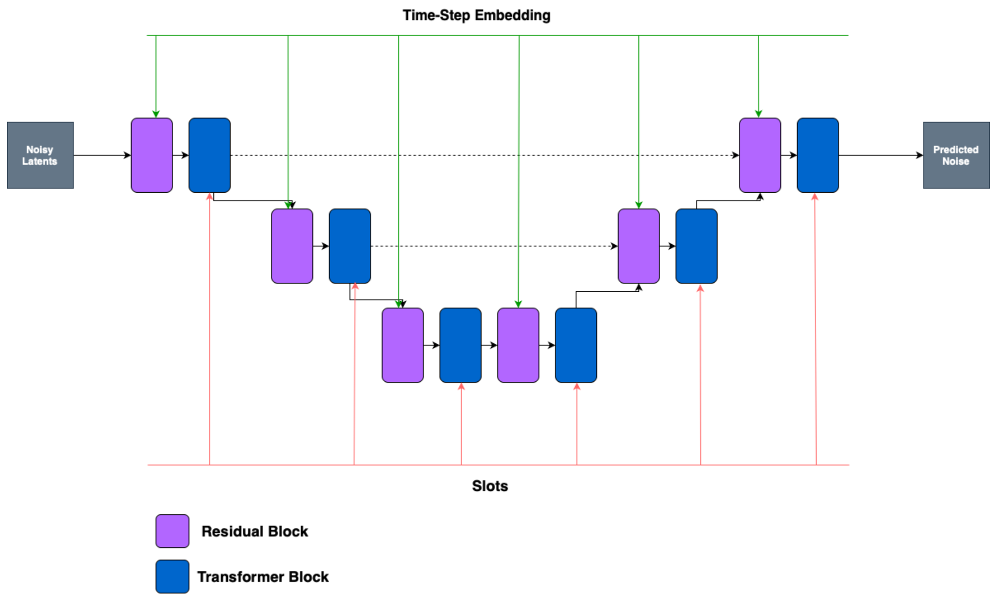

Figure 3: Overview of Unet Model

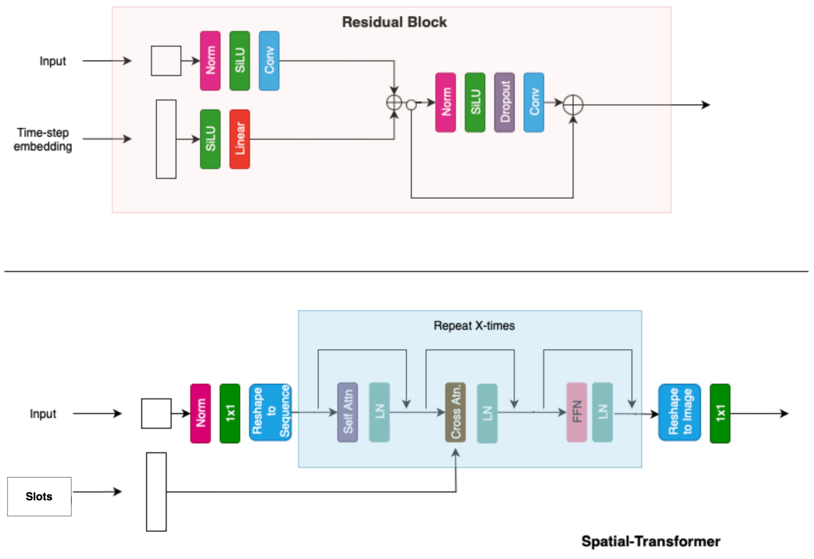

Figure 4: Detailed Block Architecture

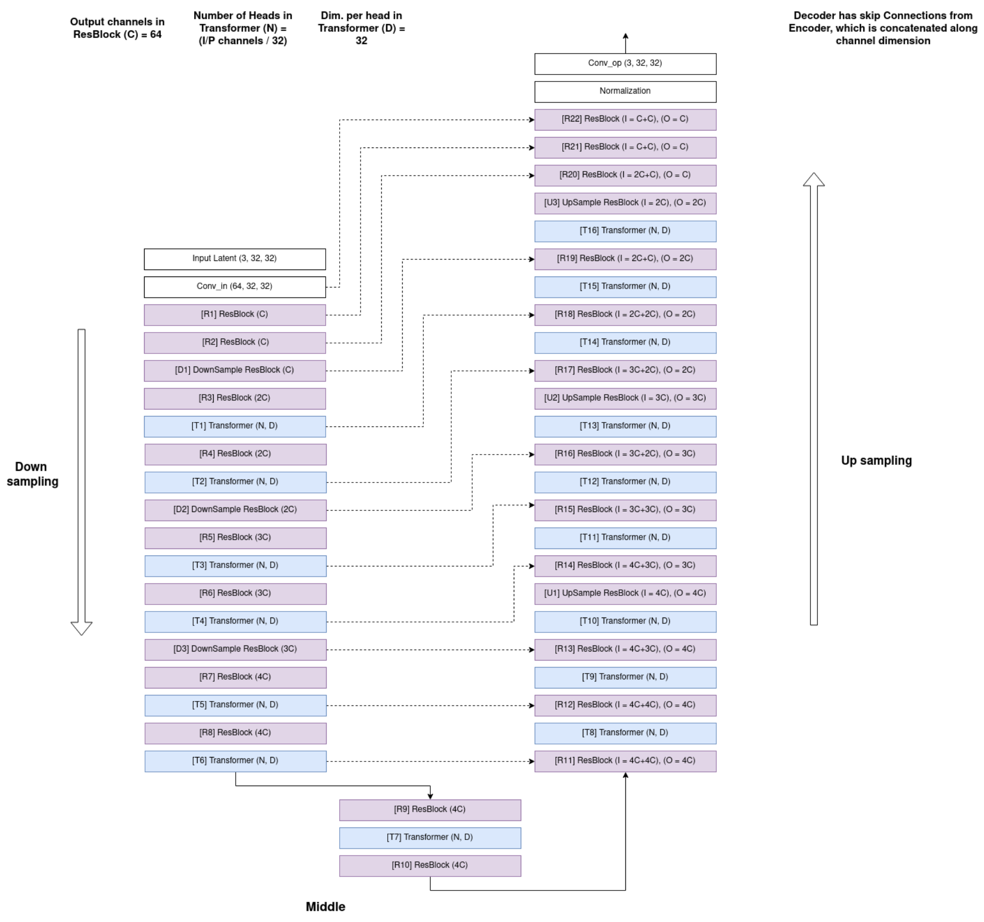
Output channels in ResBlock (C) = 64
Number of Heads in Transformer (N) = (I/P channels / 32)
Dim. per head in Transformer (D) = 32
Decoder has skip Connections from Encoder, which is concatenated along channel dimension
Conv_op (3, 32, 32)
Normalization
[R22] ResBlock (I = C+C), (O = C)
[R21] ResBlock (I = C+C), (O = C)
[R20] ResBlock (I = 2C+C), (O = C)
[U3] UpSample ResBlock (I = 2C), (O = 2C)
[T16] Transformer (N, D)
[R19] ResBlock (I = 2C+C), (O = 2C)
[T15] Transformer (N, D)
[R18] ResBlock (I = 2C+2C), (O = 2C)
[T14] Transformer (N, D)
[R17] ResBlock (I = 3C+2C), (O = 2C)
[U2] UpSample ResBlock (I = 3C), (O = 3C)
[T13] Transformer (N, D)
[R16] ResBlock (I = 3C+2C), (O = 3C)
[T12] Transformer (N, D)
[R15] ResBlock (I = 3C+3C), (O = 3C)
[T11] Transformer (N, D)
[R14] ResBlock (I = 4C+3C), (O = 3C)
[U1] UpSample ResBlock (I = 4C), (O = 4C)
[T10] Transformer (N, D)
[R13] ResBlock (I = 4C+3C), (O = 4C)
[T9] Transformer (N, D)
[R12] ResBlock (I = 4C+4C), (O = 4C)
[T8] Transformer (N, D)
[R11] ResBlock (I = 4C+4C), (O = 4C)
Up sampling
Input Latent (3, 32, 32)
Conv_in (64, 32, 32)
[R1] ResBlock (C)
[R2] ResBlock (C)
[D1] DownSample ResBlock (C)
[R3] ResBlock (2C)
[T1] Transformer (N, D)
[R4] ResBlock (2C)
[T2] Transformer (N, D)
[D2] DownSample ResBlock (2C)
[R5] ResBlock (3C)
[T3] Transformer (N, D)
[R6] ResBlock (3C)
[T4] Transformer (N, D)
[D3] DownSample ResBlock (3C)
[R7] ResBlock (4C)
[T5] Transformer (N, D)
[R8] ResBlock (4C)
[T6] Transformer (N, D)
Down sampling
[R9] ResBlock (4C)
[T7] Transformer (N, D)
[R10] ResBlock (4C)
Middle

Figure 5: Detailed Block Architecture

2.3 Experiments

Repeat the experiments from part 1 on same ub-sampled CLEVRTex dataset. For hyperparameters we **strongly advice** you to start with this model mentioned above. Also refer to slot diffusion paper [10] for unmentioned hyperparameters. Note that there are no slot-masks for reporting ARI, instead we will use slot-input attention maps `attn` $\in \mathbf{R}^{\sqrt{N}\times\sqrt{N}}$ (line number 7 in 1) as proxy to slot mask. You need to resize this attention maps to $H \times W$.
To use provided checkpoint and code of `VQE` use the following code snippet: The image input to `VAE` need to be 128×128 dimension and normalized in $[-1, 1]$.

```
import VAE from vae_
import torch
vae = VAE()

#load the checkpoint
# (replace with your checkpoint path)
ckpt = torch.load('vae_checkpoint.pth')
vae.load_state_dict(ckpt)

#to encode an image
#images: [Batch size x 3 x H x W]
z = vae.encode(images)

#to deocde z
#z: [Batch size x 3 x 32 x 32]
rec_images = vae.decode(z)
```

References

[1] MS Windows NT kernel description. `https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html`. Accessed: 2010-09-30.

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

[3] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention, 2020.

[4] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *CVPR*, 2022.

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

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

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

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

[9] Nicholas Watters, Loic Matthey, Christopher P. Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes, 2019.

[10] Ziyi Wu, Jingyu Hu, Wuyue Lu, Igor Gilitschenski, and Animesh Garg. Slotdiffusion: Object-centric generative modeling with diffusion models, 2023.