Lab5 MaskGIT for Image Inpainting

Important Date

- Submission Deadline: 5/21 (Tue) 11:59 a.m.
- Demo date: 5/21 (Tue)

Submission format

- If the zip file name or the report spec have format error, you will be punished (-5)
- Turn in: a. Experiment Report (.pdf) b. Source code
- Notice : zip all files in one file and name it like「DL_LAB5_YourStudentID_name.zip」, ex: [DL_LAB5_312581028_詹雨婷.zip」

Lab Objective

In this lab, we focus on implementing MaskGIT for the inpainting task. During testing, images contain gray regions indicating missing information, which we aim to restore using MaskGIT.

The key practical emphasis of this lab lies in three main areas: multi-head attention, transformer training, and inference inpainting.

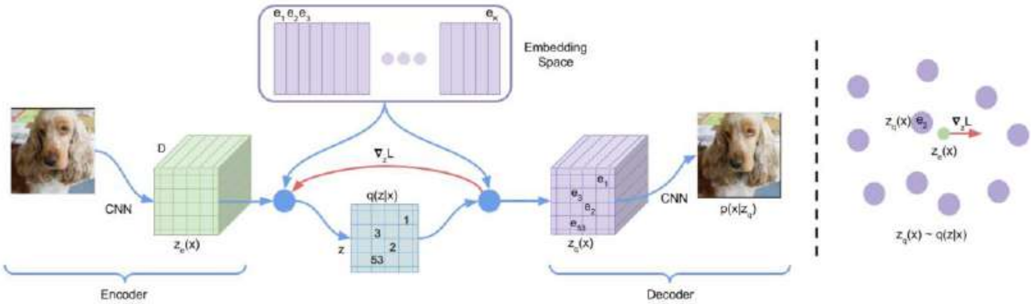
Additionally, we can experiment with different settings of mask scheduling parameters to compare their impact on inpainting results.

Requirement

1. Download the dataset and pretrained weight of VQGAN (MaksGIT stage1).
2. Implement the Multi-head attention module on your own, if you use any function directly, your demo score will -10.
3. Train your transformer model (MaskGIT stage2) from scratch.
4. Implement iterative decoding for inpainting task.
5. Compare the FID score with different settings of mask scheduling parameters and visualize the iterative decoding process for masks in latent domain or predicted images, if you don't show the visualization of iterative decoding during the demo, your demo score will -20, meaning that you won't get any experiment score.

Introduction (important concept you should know first)

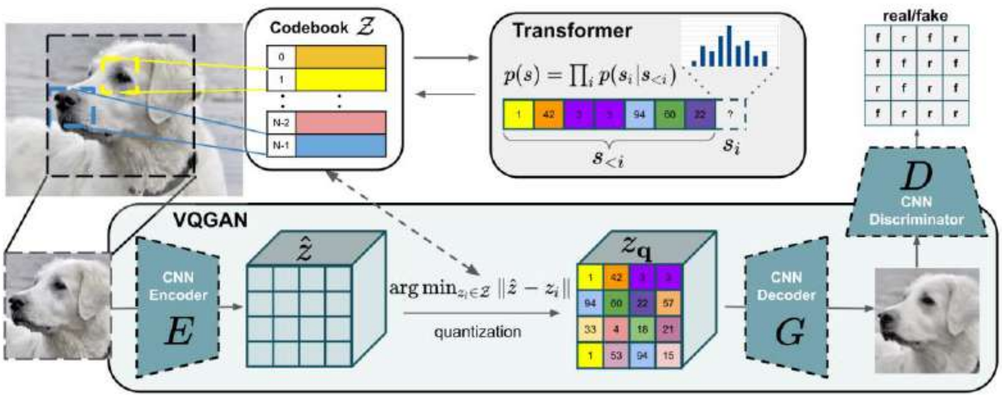
1)VQVAE: vector quantization



$$q(z=k|x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j \|z_e(x) - e_j\|_2, \\ 0 & \text{otherwise} \end{cases}$$

The traditional VAE model processes the input fed to the encoder to generate a continuous latent representation. In contrast, VQVAE proposes a discrete representation for the latent space. It requires learning the embedding space, also known as the codebook, which maps the continuous latent channel dimension vector to a codebook entry (the index 1, 2, … , k of e1, e2, … , ek). The decision of which codebook entry assign to the token is made based on the minimum Euclidean distance between the latent vector and a specific vector in the embedding space (ek).

2)VQGAN: use autoregressive transformer to predict tokens (zq)

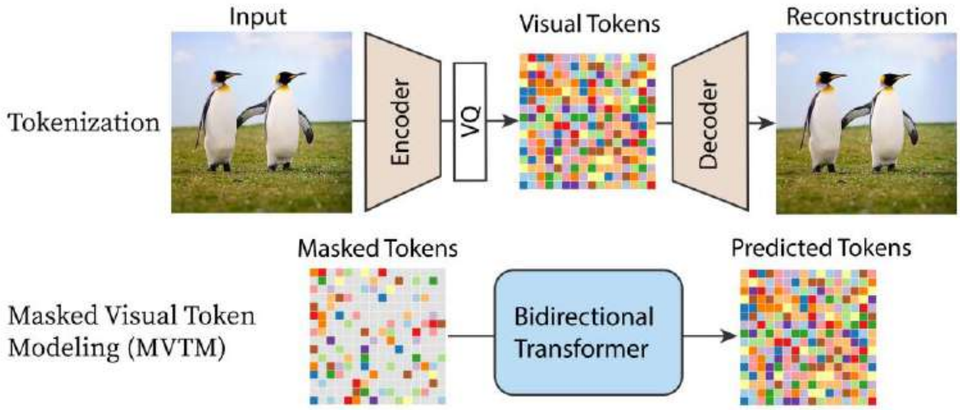


VQGAN is an improved version of VQVAE. However, the improvements made in VQGAN are highly effective, and it incorporates an autoregressive transformer to assist in generating constrained images.

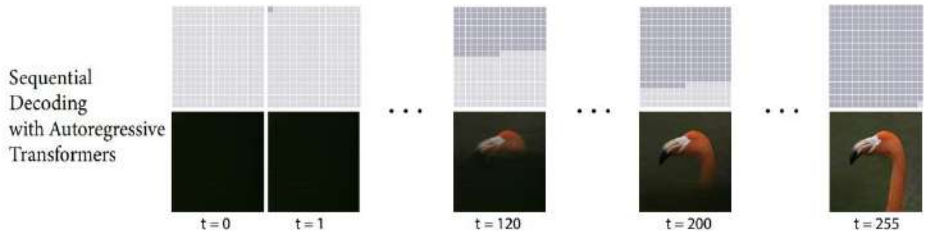
Specifically, VQGAN has two main improvements. Firstly, the authors replaced the original Mean Squared Error (MSE) Loss with Perceptual Loss as the reconstruction error in VQGAN. They also introduced the adversarial training mechanism of GANs, incorporating a patch-based Discriminator to include GAN loss in the total error.

Secondly, they introduced the Autoregressive Transformer to integrate constrained contextual information for generating tokens. The self-attention mechanism of the Transformer helps the model better capture local and global features, resulting in more accurate and diverse images.

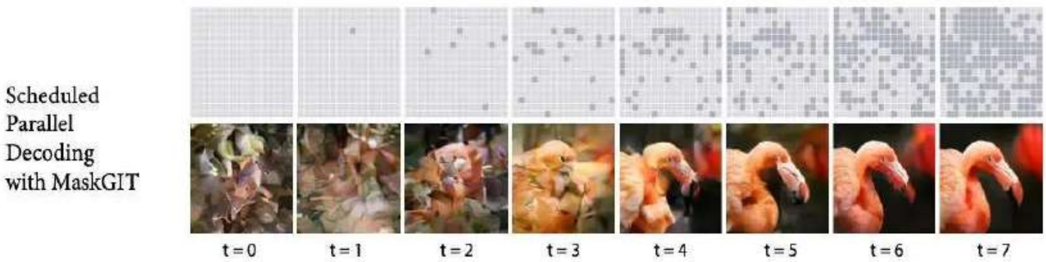
3)MaskGIT: use bidirectional transformer



MaskGIT's model largely follows the approach of VQGAN, with a primary focus on addressing the poor performance of the Autoregressive Transformer in VQGAN, where unidirectional prediction requires referencing long sequences of tokens, resulting in slow generation speeds.



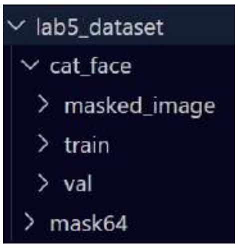
MaskGIT addresses this issue by employing a Bidirectional Transformer for token generation. The bidirectionality enables the model to predict all tokens in a single pass, significantly improving generation speed.



Furthermore, MaskGIT incorporates the Masked Visual Token Modeling (MVTM) training mechanism, inspired by human drawing logic. This core concept involves initially retaining a subset of tokens with high credibility and gradually refining the masked tokens.

Implementation Details

1. Dataset (resolution: 64*64)



a.Training dataset: image: 12000 png files (./cat_face/train)

b.Validation dataset: image: 3000 png files (./cat_face/val)

c.Testing dataset: { masked image: 747 png files (./cat_face/masked_image)
mask: 747 png files (./mask64)

d.Download dataset

i. ON your own machine

?> sftp -P 10046 pp037@140.113.215.196 (passwd: pp037OnClass)

?> get lab5_dataset.zip

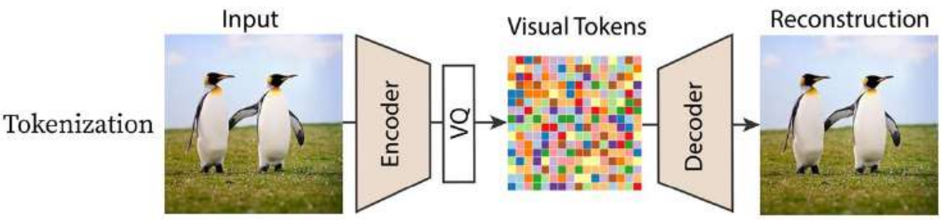
ii. ON Provided machine

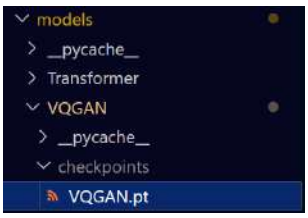
?> sftp pp037@192.168.201.46 (passwd: pp037OnClass)

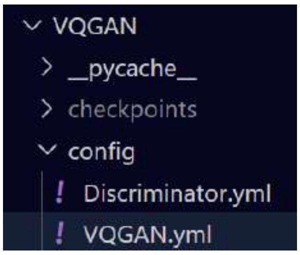
?> get lab5_dataset.zip

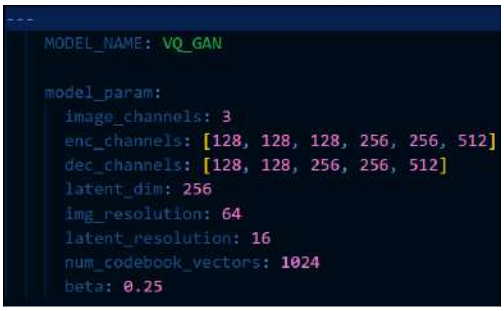
2. VQGAN Stage1 Pretrained Weight

- You can't modify any model structure or retrain stage1.
- **Although you don't need implement stage1, but you should understand some details in stage1 to connect the stage2 design.**
  - ✧ Input: resolution 64*64, channel 3
  - ✧ Latent (output from encoder): resolution 16*16, channel 256
  - ✧ # of Codebook entries: 1024
  - ✧ Dimension of Codebook (channel dimension vector): 256
  - ✧ The length of Tokens (latent after mapping the codebook embedding space) : 256 (flatten 16*16)









i. ON your own machine

?> sftp -P 10046 pp037@140.113.215.196 (passwd: pp037OnClass)

?> get VQGAN.pt

ii. ON Provided machine

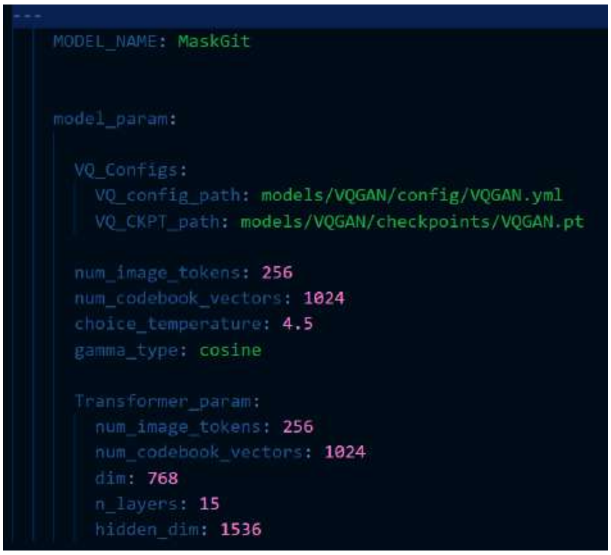
?> sftp pp037@192.168.201.46 (passwd: pp037OnClass)

?> get VQGAN.pt

3. Multi-Head Attention module by your own

(models/Transformer/modules/layers.py find #TODO1)

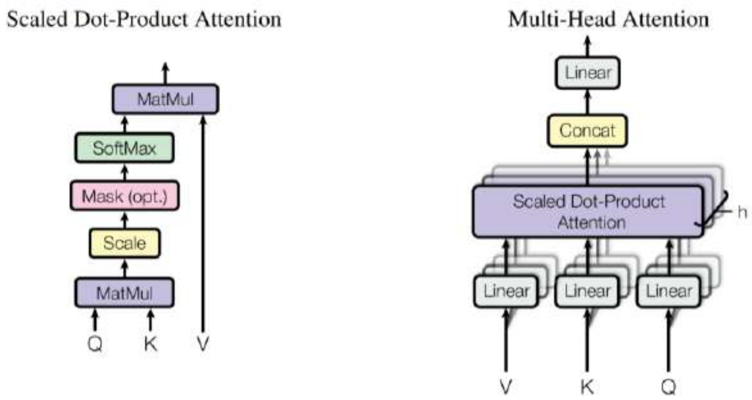
- **You can't use any functions directly ex. torch.nn.MutiheadAttention**
- **If you use any function, your demo score will -10.**
- **Hint: input tensor shape is (batch_size, num_image_tokens, dim), because the bidirectional transformer first will embed each token to dim dimension, and then pass to n_layers of encoders consist of Multi-Head Attention and MLP.**



- **# of head set 16**
- **Total d_k , d_v set to 768**
- **d_k , d_v for one head will be 768//16.**

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



4. MaskGIT Stage2: Training bidirectional transformer

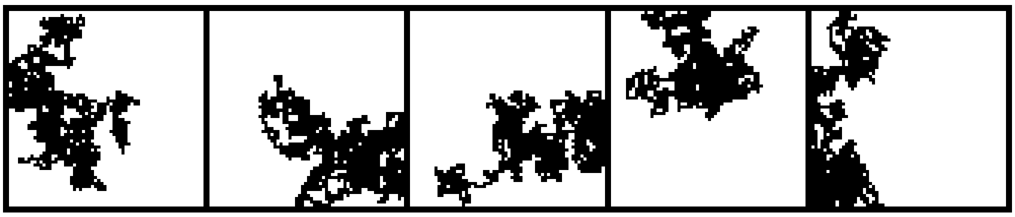
(models/VQGAN_Transformer.py and training_transformer.py find #TODO2, follow the step!)

- **You can't modify any model architecture. (config/MaskGIT.yml, models/Transformer/transformer.py and every module in models/Transformer/modules/layers.py except MultiHeadAttention)**
- You can design any training strategy.
- Training and validation dataset are already split, you can't use validation dataset to train.
- In training stage, the mask ratio will be random, it won't follow any iterative mask scheduling.
- The loss function is determined by the cross entropy loss between ground truth (input fed to encoder then vector quantization) and the tokens predicted by bidirectional transformer.

$$\gamma(r) \in (0, 1] \qquad \mathcal{L}_{\text{mask}} = -\mathop{\mathbb{E}}_{\mathbf{Y} \in \mathcal{D}} \Big[\sum_{\forall i \in [1,N], m_i = 1} \log p(y_i | Y_{\overline{\mathbf{M}}}) \Big]$$

5. Inference for Image Inpainting task

(models/VQGAN_Transformer.py and inpainting.py find #TODO3, follow the step!)



- In Inference stage, the mask ratio will follow the iterative mask scheduling you should try different settings.
- You may confuse that how the mask positions in 64*64 image corresponding to the initial mask positions in 16*16 token length.
  (you can check the get_mask_latent function)

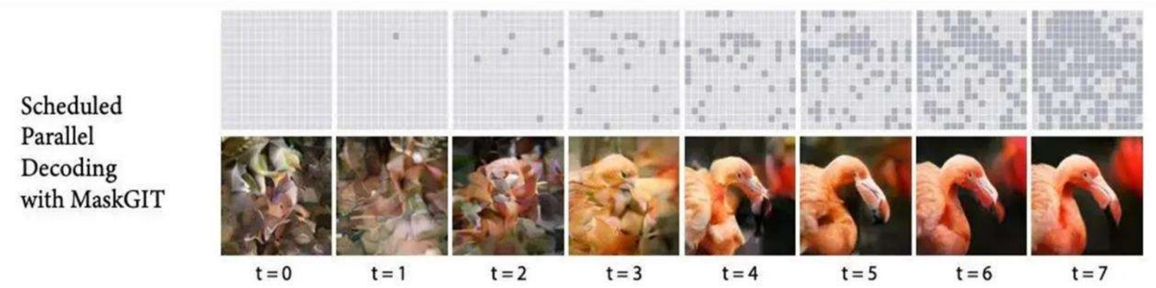
```python
class MaskedImage:
    def __init__(self, args):
        mi_ori=LoadTestData(root= args.test_maskedimage_path, partial=args.partial)
        self.mi_ori =  DataLoader(mi_ori,
                                  batch_size=args.batch_size,
                                  num_workers=args.num_workers,
                                  drop_last=True,
                                  pin_memory=True,
                                  shuffle=False)
        mask_ori =LoadMaskData(root= args.test_mask_path, partial=args.partial)
        self.mask_ori =  DataLoader(mask_ori,
                                  batch_size=args.batch_size,
                                  num_workers=args.num_workers,
                                  drop_last=True,
                                  pin_memory=True,
                                  shuffle=False)
        self.device=args.device

    def get_mask_latent(self,mask):
        downsampled1 = torch.nn.functional.avg_pool2d(mask, kernel_size=2, stride=2)
        resized_mask = torch.nn.functional.avg_pool2d(downsampled1, kernel_size=2, stride=2)
        resized_mask[resized_mask != 1] = 0       #1,3,16*16   check use
        mask_tokens=(resized_mask[0][0]//1).flatten()   ##[256] =16*16 token
        mask_tokens=mask_tokens.unsqueeze(0)
        mask_b = torch.zeros(mask_tokens.shape, dtype=torch.bool, device=self.device)
        mask_b |= (mask_tokens == 0) #true means mask
        return mask_b
```

- The initial mask ratio is set to 1, indicating that all regions are masked, similar to the black region of the binary image shown above. This ratio will decrease according to the mask scheduling strategy. The transformer predicts the probability of assigning each token to every codebook entry, akin to a classification task. It then selects the codebook entry with the highest probability for each token. These probabilities are sorted, and based on the current mask ratio, the positions for masking in the subsequent iteration are determined. Tokens designated to remain unmasked will use the predicted codebook entry from this step.

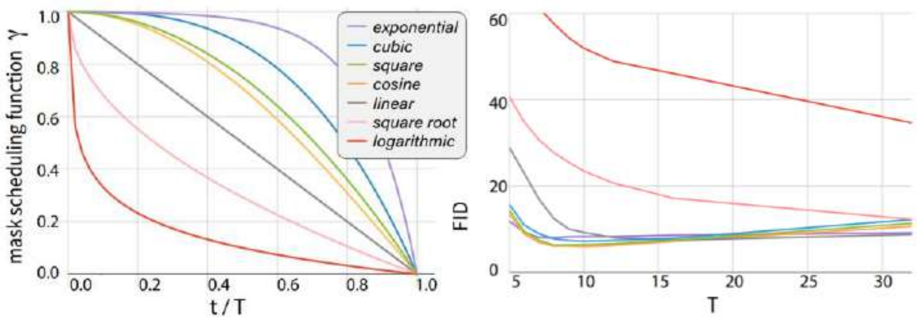
$$n = \lceil \gamma(\tfrac{t}{T}) N \rceil \qquad m_i^{(t+1)} = \begin{cases} 1, & \text{if } c_i < \text{sorted}_j(c_j)[n]. \\ 0, & \text{otherwise.} \end{cases}$$

6. Mask Scheduling parameters



- Mask Scheduling Functions

  •cosine • linear • square
- Number of iterations T (you can adjust)
- Sweet spot t (you can adjust)



Report Spec (40%)

1. Introduction (5%)
2. Implementation Details (60%)
   A. The details of your model (Multi-Head Self-Attention)
   B. The details of your stage2 training (MVTM, forward, loss)
   C. The details of your inference for inpainting task (iterative decoding)
3. Experimental results (30%)
   A. The best testing fid(21%)
      - Screenshot
      - Predicted image, Mask in latent domain with mask scheduling
      - The setting about training strategy, mask scheduling parameters…

   B. Comparison figures with different mask scheduling parameters setting (total 9%) (each 3%)

      •cosine • linear • square
4. Discussion(5%)
   A. Anything you want to share

Demo (70%)

Part1: Prove your code implementation is correct

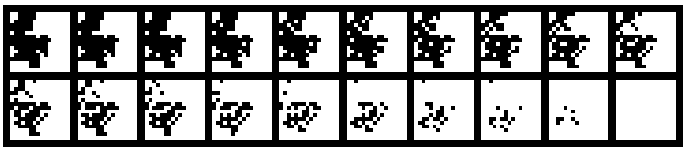
1.Show Multi-Head Attention module.

If you directly use any functions, your demo score will -10.

2.Choose either one to show iterative decoding.

If Both missing, your demo score will -20.

(a)Mask in latent domain (specific 2 serial number)



(b)Predicted image (specific 2 serial number)

Part2: Experiment Score (20%)

--predicted-path should be the folder of test results which you will get after inference for inpainting, please check the order of image name which can corresponding to the images in the folder of masked image.

The FID score which correlate well with human judgement of visual quality is a measure of similarity between two datasets of images, and lower scores have been shown to correlate well with higher quality images.

```
cd faster-pytorch-fid
python fid_score_gpu.py --predicted-path /path/your_inpainting_results_folder --device cuda:0
```

Average FID	Score
$40 \geq \text{FID}$	20
$45 \geq \text{FID} > 40$	17
$50 \geq \text{FID} > 45$	14
$55 \geq \text{FID} > 50$	11
$60 \geq \text{FID} > 55$	8
$65 \geq \text{FID} > 60$	5
$\text{FID} > 65$	0

Part3: Question (50%)

Reference

1. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NeurIPS, 2017. https://arxiv.org/pdf/2202.04200.pdf
2. A. van den Oord, O. Vinyals, et al., “Neural discrete representation learning,” in Advances in Neural Information Processing Systems, pp. 6306–6315, 2017. https://arxiv.org/abs/1711.00937
3. Esser, P., Rombach, R., and Ommer, B.: Taming Transformers for High-Resolution Image Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12873–12883 (2021) https://arxiv.org/abs/2012.09841
4. Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2022. https://arxiv.org/abs/2202.04200