

NYCU DL Lab5 – MaskGIT for Image Inpainting

Student ID: 313553014 Name: 廖怡誠

1. Introduction

在這次的作業中，我們要實作 MaskGIT 並應用於 Inpainting 的任務，其中必須了解 Multi-Head self-Attention、MVTM 與 Inpainting 的原理，通過 Multi-Head self-Attention、MVTM 使 model 學習到更多圖片的資訊(Encoder)，再使用 Decoder 將圖片從雜訊中恢復成原圖(Inpainting)。

2. Implementation Details

A. The details of your model (Multi-Head Self-Attention)

multi-head self-attention 是 self-attention 的進階版本，使用多個 head 能夠使模型學習到不同類別之間的相關資訊。首先，我將 q、k、v 都須通過的 Linear layer 設計成一個 input、output 為(dim, dim * 3) 的 Linear layer，其輸出能夠得到維度為 (batch, token, 3 * dim)的輸出，其中 dim 又可以拆成 head 數量(num_head)乘上每個 head 的維度(dim_head)，將其分成 3 個維度為(batch, num_head, token, dim_head)的矩陣，稱為 q、k、v，接著進入到 scaled dot-product attention 的部分，其中如示意圖中逐一實作，最後將 3 個矩陣組合，維度為最初輸入的(batch, token, dim)，並再通過一層 Linear layer 即完成 multi-head self-attention。

```
class MultiHeadAttention(nn.Module):
    def __init__(self, dim=768, num_heads=16, attn_drop=0.1):
        super(MultiHeadAttention, self).__init__()

        self.dim_head = dim // num_heads
        self.heads = num_heads
        # 1 / (self.dim_head) ^ 0.5
        self.scale = self.dim_head ** -0.5

        self.to_qkv = nn.Linear(dim, dim * 3, bias=False)

        # output
        self.W_o = nn.Sequential(
            nn.Linear(dim, dim),
            nn.Dropout(p=attn_drop)
        )

    def forward(self, x):
        """ Hint: input x 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.
        """
        h = self.heads
        # first linear to qkv
        qkv = self.to_qkv(x)
        # divide to q, k and v
        q, k, v = tuple(qkv.view(3, x.shape[0], h, -1, self.dim_head))
        # scaled dot-product attention
        attn = torch.matmul(q, k.transpose(-2, -1)) * self.scale
        weight = attn.softmax(dim=-1)
        context = torch.matmul(weight, v)
        # concat multi-head
        concat_content = context.view(x.shape[0], -1, self.dim_head * h)
        # last linear
        return self.W_o(concat_content)
```

圖 1. multi-head self-attention 程式碼

B. The details of your stage2 training (MVTM, forward, loss)

在介紹 Stage 2 之前，要先說明 Encoder 的程式碼(圖 2)，Encoder 會將輸入 mapping 到 latent code 中，並使用 codebook 紀錄轉換出的 embedding，所以在圖二中，可以看到 VQ-GAN 的 Encoder 輸出會有一個 codebook mapping 與 codebook indices，分別為記錄用 codebook 與輸入 x 對應到 codebook 的 induces，此輸出屬於 Ground Truth。

而在主要的 MVTM 實作(圖 3，forward 部分)，首先會通過 Encoder 得到 Ground Truth，接著使用 Bernoulli Distribution 得到只有 0 或 1 的 mask，使用此 mask 將值為 1 所對應到的 pixel 加上 mask；反之，值為 0 其對應到的 pixel 保留原圖，最後通過 Bidirectional Transformer 得到 predicted tokens。將 Ground Truth 讀 visual token 與 predicted tokens 回傳，並使用 Cross Entropy 計算其 Loss (圖 4)。

```
@torch.no_grad()
def encode_to_z(self, x):
    codebook_mapping, codebook_indices, _ = self.vqgan.encode(x)
    return codebook_mapping, codebook_indices.view(codebook_mapping.shape[0], -1)
```

圖 2. Encoder 程式碼

```
def forward(self, x):
    # stage 2 MVTM
    # ground truth tokens
    _, visual_tokens = self.encode_to_z(x)
    # create mask
    mask = torch.bernoulli(0.5 * torch.ones(visual_tokens.shape, device=visual_tokens.device)).bool()
    # for masked tokens
    masked_indices = self.mask_token_id * torch.ones_like(visual_tokens, device=visual_tokens.device)
    # replace Yi with [mask] if m = 1, otherwise, when m = 0
    masked_tokens = mask * masked_indices + (~mask) * visual_tokens
    predicted_tokens = self.transformer(masked_tokens) #transformer predict the probability of tokens

    logits = predicted_tokens
    z_indices = visual_tokens

    return logits, z_indices
```

圖 3. MVTM 程式碼

```
def train_one_epoch(self, train_loader, epoch):
    self.model.train()
    batch_loss = []
    pbar = tqdm(enumerate(train_loader))

    for i, batch in pbar:
        img = batch.to(self.args.device)
        # output: gt, pred
        logits, z_indices = self.model(img)
        loss = F.cross_entropy(logits.reshape(-1, logits.size(-1)), z_indices.reshape(-1))
        loss.backward()

        batch_loss.append(loss.item())
        if i % args.accum_grad == 0:
            self.optim.step()
            self.optim.zero_grad()
        pbar.set_description_str(f"epoch: {epoch} / {self.args.epochs}, iter: {i} / {len(train_loader)}, loss: {np.mean(batch_loss)}")

    return np.mean(batch_loss)
```

圖 4. Training (Loss) 程式碼

C. The details of your inference for inpainting task (iterative decoding)

Inference 主要的任務是 Decoder 會將有 mask 的圖片重新修復，在 paper 中大致分為 predict、mask schedule、sample 與 mask 四個部分。圖六中，第一步，先通過 transformer 預測所有 masked 位置的機率，並找出每個 token 中最大機率的 index，第二步，使用當前 iteration 與總 iteration(圖 5)通過 gamma 函式得到 mask 的比例，並計算出當前 iteration 要 mask 的 token 數量，第三步，將 predict 出的機率加入 temperature annealing gumbel noise 作為 confidence score，也就是模型對於預測結果的信心程度，最後，會將信心度高的 token 保留，剩餘的 token 則會被重新 mask 作為下一次 predict 的對象，重複循環此過程，直到所有的 token 都不被 mask。

```
self.model.eval()
with torch.no_grad():
    z_indices = self.model.encode_to_z(image[0].unsqueeze(0))[1] #z_indices: masked tokens (b,16*16)
    mask_num = mask_b.sum() #total number of mask token
    z_indices_predict=z_indices
    mask_bc=mask_b
    mask_b=mask_b.to(device=self.device)
    mask_bc=mask_bc.to(device=self.device)
    |
    ratio = 0
    #iterative decoding for loop design
    #Hint: it's better to save original mask and the updated mask by scheduling separately
    for step in range(self.total_iter):
        if step == self.sweet_spot:
            break
        ratio = (step + 1) / (self.total_iter) #this should be updated
    z_indices_predict, mask_bc = self.model.inpainting(z_indices_predict, mask_bc, mask_num, ratio)
```

圖 5. Inpainting 程式碼

```
@torch.no_grad()
def inpainting(self, z_indices, mask, mask_num, ratio):
    """ predict """
    masked_z_indices = mask * self.mask_token_id + (~mask) * z_indices
    logits = self.transformer(masked_z_indices)
    #Apply softmax to convert logits into a probability distribution across the last dimension.
    prob = torch.softmax(logits, dim=-1)
    #Find max probability for each token value & max prob index
    z_indices_predict_prob, z_indices_predict = prob.max(dim=-1)

    """ mask schedule """
    # gamma(t / iter)
    mask_ratio = self.gamma(ratio)
    # number of taken to mask = ceil(mask_ratio * mask_num)
    mask_len = torch.ceil(mask_num * mask_ratio).long()

    """ sample """
    #predicted probabilities add temperature annealing gumbel noise as confidence
    g = torch.distributions.gumbel.Gumbel(0, 1).sample(z_indices_predict_prob.shape).to(z_indices_predict_prob.device) # gumbel noise
    temperature = self.choice_temperature * (1 - mask_ratio)
    confidence = z_indices_predict_prob + temperature * g

    """ mask """
    #hint: If mask is False, the probability should be set to infinity, so that the tokens are not affected by the transformer's prediction
    confidence[~mask] = torch.inf
    #sort the confidence for the rank
    _, idx = confidence.topk(mask_len, dim=-1, largest=False)
    #define how much the iteration remain predicted tokens by mask scheduling
    mask_bc = torch.zeros(z_indices.shape, dtype=torch.bool, device=z_indices_predict_prob.device)
    #At the end of the decoding process, add back the original token values that were not masked to the predicted tokens
    mask_bc = mask_bc.scatter_(dim=1, index=idx, value=True)
    return z_indices_predict, mask_bc
```

圖 6. Inpainting 程式碼

3. Discussion

A. Comparison with training loss and evaluation loss

我訓練的設定 epochs 為 100，learning rate 為 $1e-4$ ，loss 變化如圖 7 所示，training loss 有穩定下降，而 evaluation loss 則是在大約 40 epoch 左右時，其 loss 就沒有太大的變化，有明顯的 overfitting，我認為最有效的改善方法是使用 paper 中有使用的 random resize and crop 的 data augmentation 以及使用 label smoothing 的技巧使模型可以更容易學習 latent。

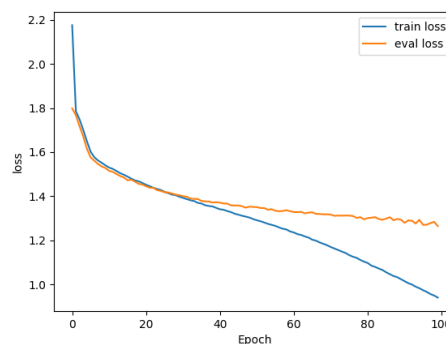


圖 7. Training & evaluation loss

B. Comparison with different mask schedule

除了預設使用的 $1 - x^2$ 、 $\cos x \frac{\pi}{2}$ 、 $1 - x$ 之外，我還加入了 $1 - \sin x \frac{\pi}{2}$ 、 $1 - \sqrt{x}$ 與 $1 - x^3$ ，圖 8 是這六種函式所對應到的曲線圖，以及其對應的 FID score 表格，原先我推測越接近 cosine 曲線的效果會越好，但從表格中可以得知這個想法是錯誤的， $1 - \sin x \frac{\pi}{2}$ 曲線的效果與 $\cos x \frac{\pi}{2}$ 的效果相差約 0.01 左右，但兩者的曲線差異極大，因此，推測只要適當變化，即可有好效果，不能如 linear 曲線變化平均，也不能如 cube 曲線變化激烈。

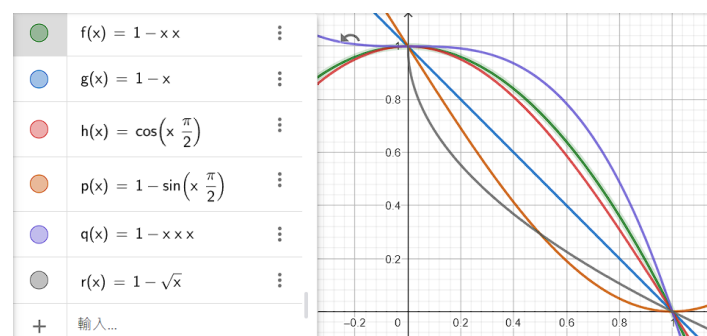


圖 8. 六種函式對應曲線圖

method	square	linear	cosine	sin	cube	sqrt
FID	30.01	29.72	29.29	29.30	31.60	30.90

C. Comparison with different sweet point & number of iterations

除了不同的 mask schedule 之外，在 inpainting 程式中還有 sweet point 與 iterations 的數量可以調整，sweet point 會決定要 decode 的次數，而 iterations 的數量會影響到 mask schedule 的 ratio 數值，所以我將以 cosine 為例，比較不同 iterations 數量與 sweet point 的結果。第一個實驗中，使用相同的 iterations 與 sweet point 進行實驗，使用(8, 8)的 FID 效果最好，但差異不大，從下面兩張圖雖然 iteration 的數量不同，但差不多都是在第一個 iteration 就已經去除掉大部分的 mask，所以有此結果；第二個實驗中，我固定 sweet point 並調找 iteration 的數量，比較不同 ratio 數值對 FID 的影響，實驗結果證明(8, 8)的效果最好，兩個實驗的數據都沒有明顯的線性變化，因此推測這兩個參數對於 FID 的影響不大。

number of iterations	sweet point	FID
4	4	30.10
8	8	29.29
12	12	29.95
15	15	29.77



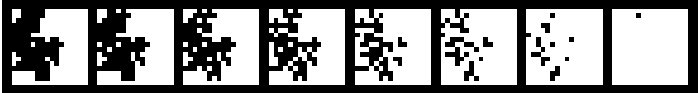
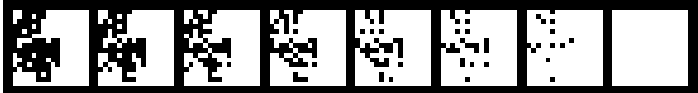
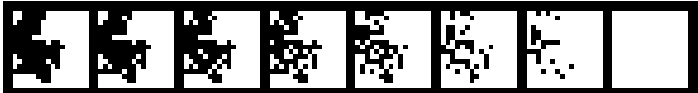
number of iterations	sweet point	FID
8	8	29.29
8	10	29.78
8	20	29.69
8	40	30.03

4. Prove your code implementation is correct

- Show iterative decoding. (cosine, linear, square)


i. Mask in latent domain

三種不同的 mask schedule 所形成的 mask 分布都不相同，但都會隨著 iteration 增加，而逐漸減少。

Cosine	
Linear	
Square	

ii. Predicted image

Decode 的图片會隨著 iteration 的增加變得越來越清晰，不過，三種 mask schedule 方法在圖片上的變化粗略看起來很相似，但是細看細節與特徵的話，可以發現還是略有不同之處。

Cosine	
Linear	
Square	

5. The Best FID Score

A. Screenshot

我分別使用三種 mask schedule 進行 Inpainting，並得到圖 9、10、11 的 FID score 結果，分別為 Cosine 是 29.29、Linear 是 29.72 和 Square 是 30.01 與論文 MaskGIT 的實驗結果有所差異，是 Cosine 的 mask schedule 得到的 FID score 最好，其次是 Linear，最後是 Square，不過，三種方法的數值差異不大。



圖 9. Cosine FID score

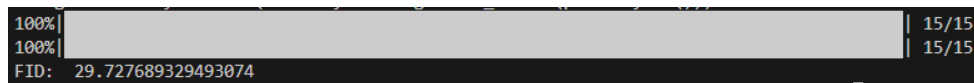


圖 10. Linear FID score



圖 11. Square FID score

B. Masked Images v.s MaskGIT Inpainting Results v.s Ground Truth

在此處的比較中，只展示 FID score 最好的 Cosine mask schedule，可以看到模型輸出的結果與 Ground Truth 還是有明顯的不同，像是第五張圖中，模型生成的貓伸出舌頭的特徵並不明顯而 Ground Truth 是很明顯的。

Masked Images						
MaskGIT Inpainting Results						
Ground Truth						

C. The setting about training strategy, mask scheduling parameters, and so on

Training Setting	
num_workers	4
batch-size	12
epochs	100
partial	1.0
accum-grad	10
learning-rate	1e-4
cmd	
python training_transformer.py --train_d_path ./lab5_dataset/train\ --val_d_path ./lab5_dataset/val --epochs 100	

Inference Setting (Inpainting)	
num_workers	4
batch-size	1
sweet-spot	8
total-iter	8
mask-func	使用 MaskGit.yml 控制
cmd	
python inpainting.py\ --test_maskedimage-path ./lab5_dataset/masked_image\ --test-mask-path ./lab5_dataset/mask64	

FID score	
Predicted-path	根據生成的 test_results 資料夾位置
cmd	
python faster-pytorch-fid/fid_score_gpu.py --predict-path ./test_results	