Lab5 MaskGIT for Image Inpainting

313551104 黃暐洺

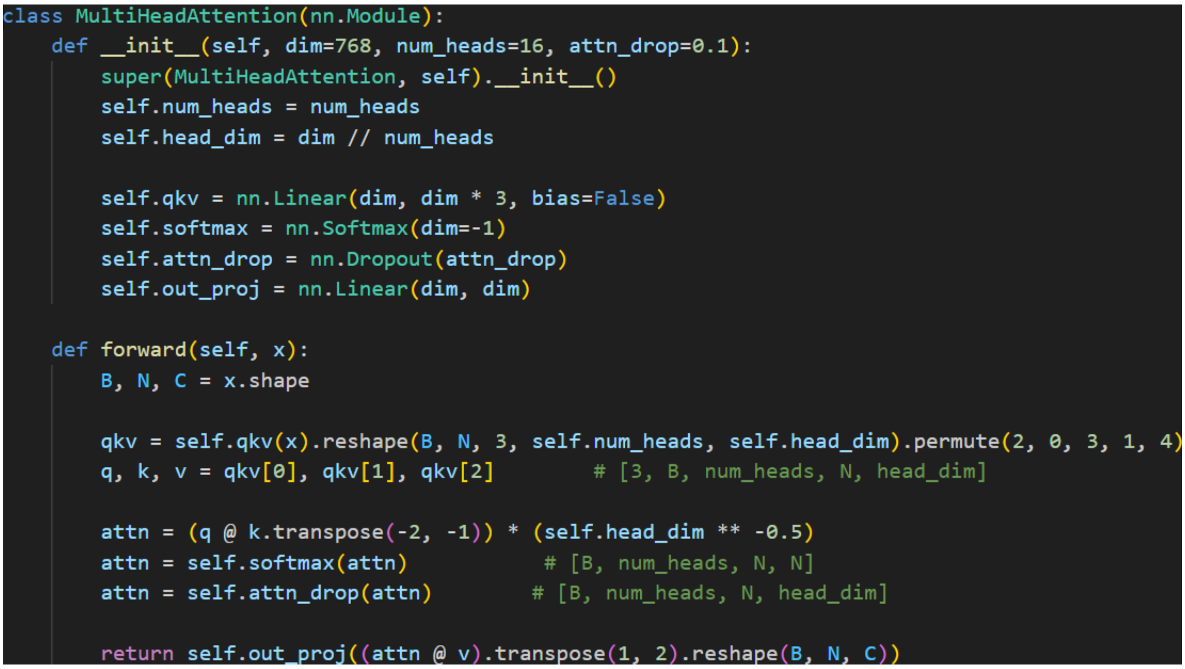
Report

1. Introduction (5%)

   本次 Lab 聚焦於實現並應用 MaskGIT 來進行圖像預測任務，具體實作了多頭注意力機制、了解 Masked Visual Token Modeling 的詳細訓練方式並應用在 Transformer 上、以及完成 inpainting 這個功能作為模型的預測階段。最後使用 Fréchet Inception Distance 當作評分標準，探索訓練和預測的不同超參數，目標是獲得成績最好的預測模型。

2. Implementation Details (45%)

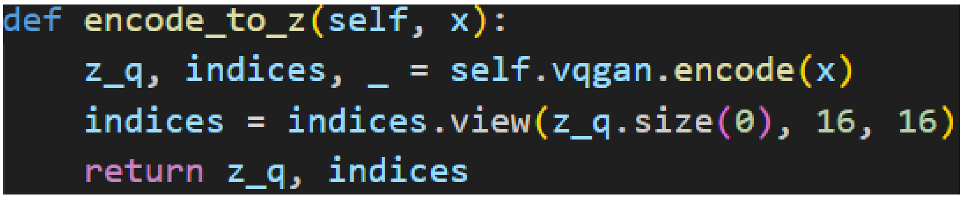
   A. The details of your model (Multi-Head Self-Attention)



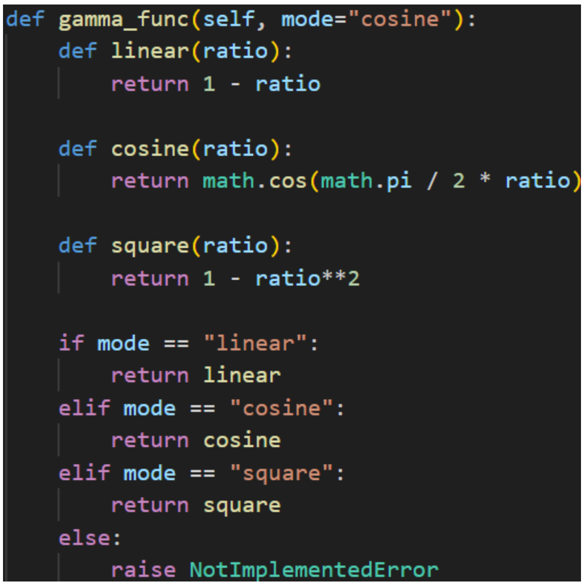
```
class MultiHeadAttention(nn.Module):
    def __init__(self, dim=768, num_heads=16, attn_drop=0.1):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
        self.head_dim = dim // num_heads

        self.qkv = nn.Linear(dim, dim * 3, bias=False)
        self.softmax = nn.Softmax(dim=-1)
        self.attn_drop = nn.Dropout(attn_drop)
        self.out_proj = nn.Linear(dim, dim)

    def forward(self, x):
        B, N, C = x.shape

        qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, self.head_dim).permute(2, 0, 3, 1, 4)
        q, k, v = qkv[0], qkv[1], qkv[2]        # [3, B, num_heads, N, head_dim]

        attn = (q @ k.transpose(-2, -1)) * (self.head_dim ** -0.5)
        attn = self.softmax(attn)           # [B, num_heads, N, N]
        attn = self.attn_drop(attn)        # [B, num_heads, N, head_dim]

        return self.out_proj((attn @ v).transpose(1, 2).reshape(B, N, C))
```

qkv 原本分別要乘上各自的 weight，這邊寫成 dim 到 dim3 的 Linear 層一起運算，效果一樣，讓程式碼更整潔，其餘就照著給定的 spec 去做，為了可以讓模型有更好的泛化能力，在 soft max 後面加了 dropout 層。

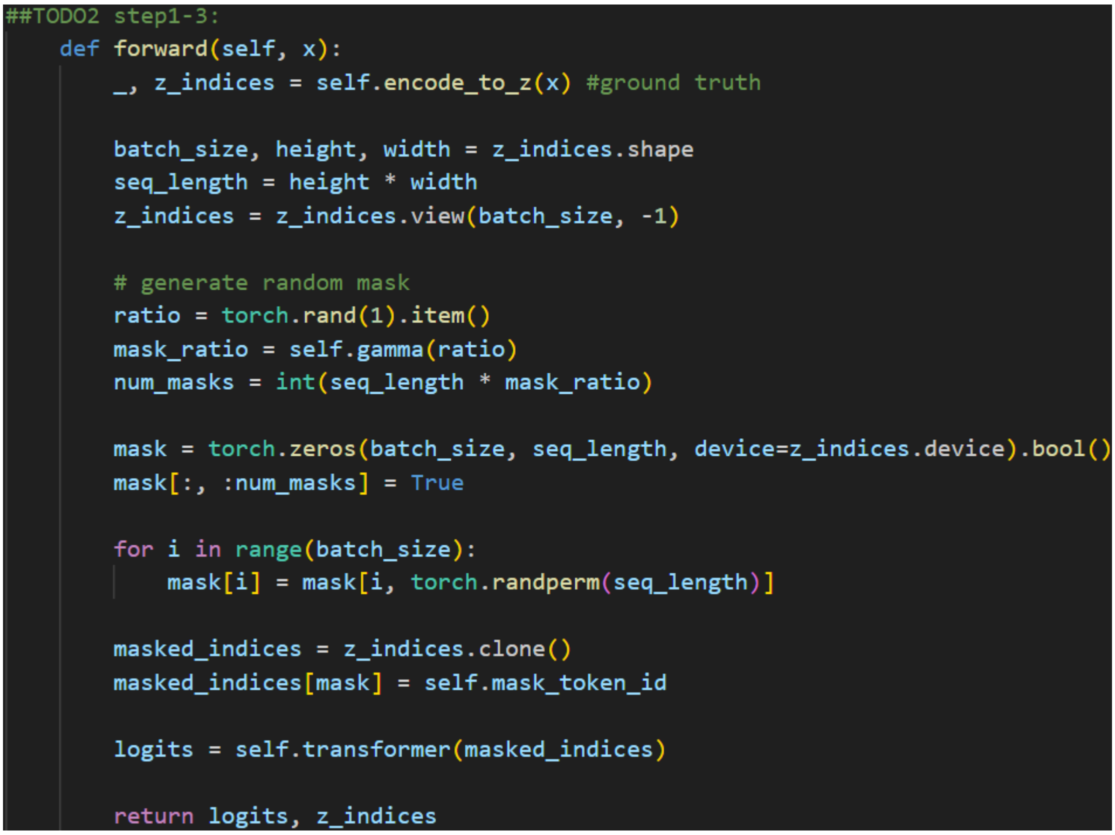
B. The details of your stage2 training (MVTM, forward, loss)



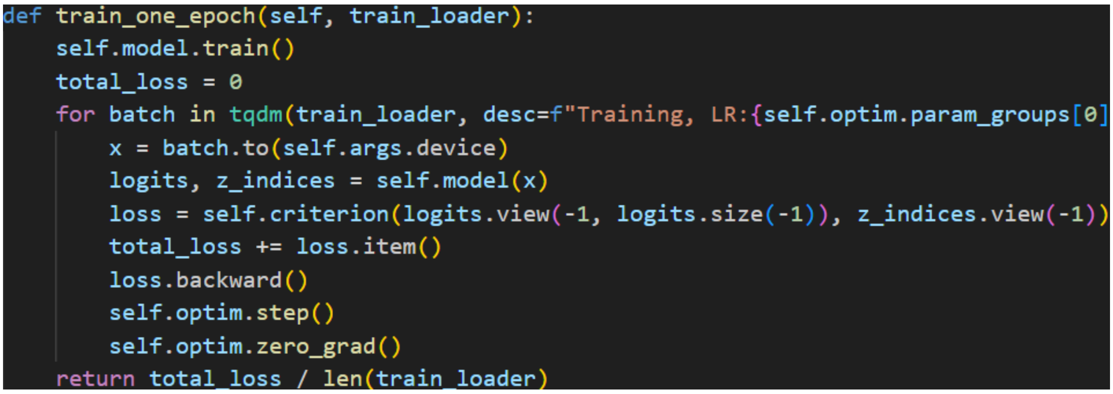
```
def encode_to_z(self, x):
    z_q, indices, _ = self.vqgan.encode(x)
    indices = indices.view(z_q.size(0), 16, 16)
    return z_q, indices
```



```
def gamma_func(self, mode="cosine"):
    def linear(ratio):
        return 1 - ratio

    def cosine(ratio):
        return math.cos(math.pi / 2 * ratio)

    def square(ratio):
        return 1 - ratio**2

    if mode == "linear":
        return linear
    elif mode == "cosine":
        return cosine
    elif mode == "square":
        return square
    else:
        raise NotImplementedError
```



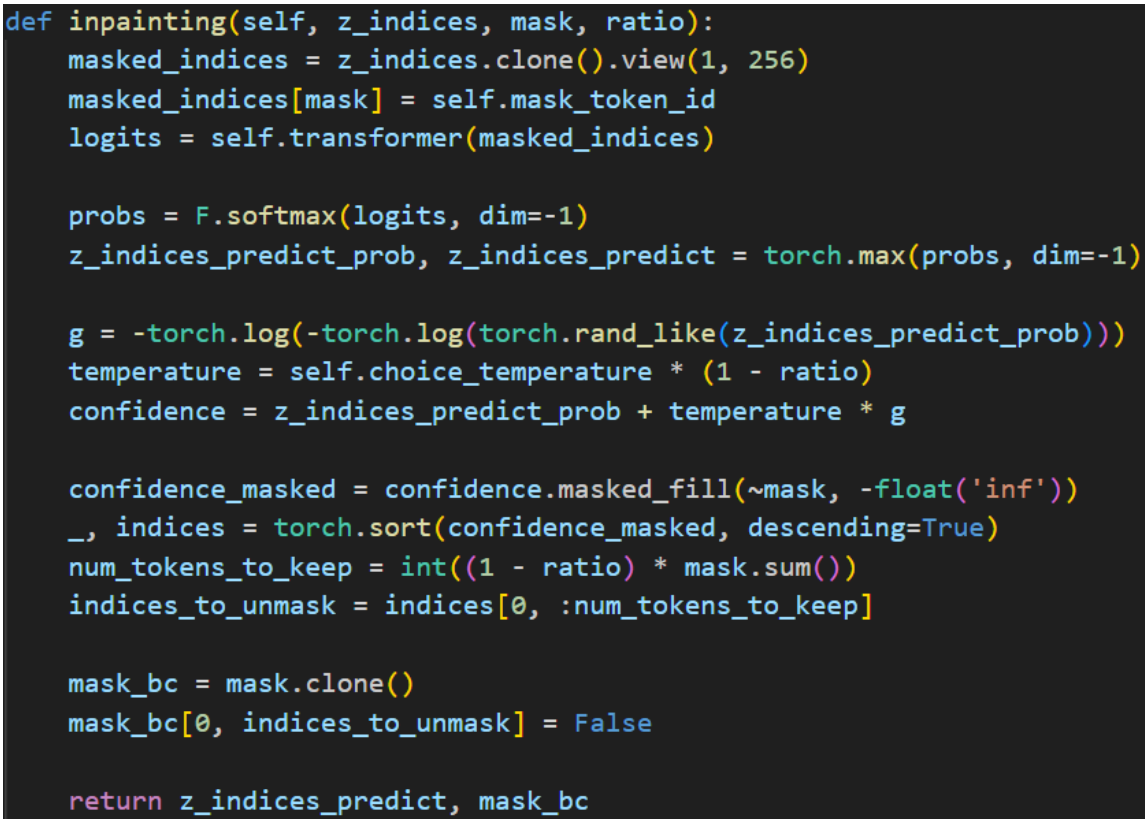
```
##TODO2 step1-3:
    def forward(self, x):
        _, z_indices = self.encode_to_z(x) #ground truth

        batch_size, height, width = z_indices.shape
        seq_length = height * width
        z_indices = z_indices.view(batch_size, -1)

        # generate random mask
        ratio = torch.rand(1).item()
        mask_ratio = self.gamma(ratio)
        num_masks = int(seq_length * mask_ratio)

        mask = torch.zeros(batch_size, seq_length, device=z_indices.device).bool()
        mask[:, :num_masks] = True

        for i in range(batch_size):
            mask[i] = mask[i, torch.randperm(seq_length)]

        masked_indices = z_indices.clone()
        masked_indices[mask] = self.mask_token_id

        logits = self.transformer(masked_indices)

        return logits, z_indices
```



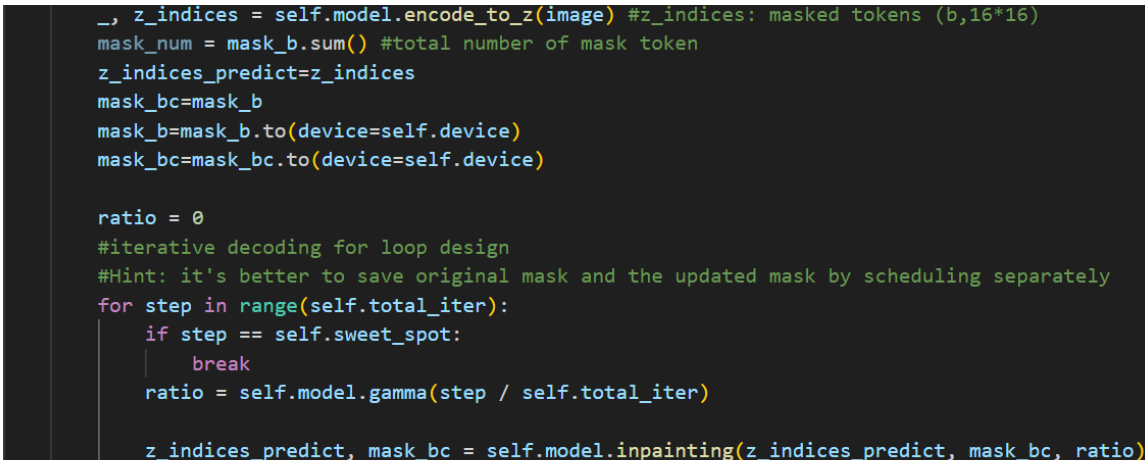
```
def train_one_epoch(self, train_loader):
    self.model.train()
    total_loss = 0
    for batch in tqdm(train_loader, desc=f"Training, LR:{self.optim.param_groups[0]
        x = batch.to(self.args.device)
        logits, z_indices = self.model(x)
        loss = self.criterion(logits.view(-1, logits.size(-1)), z_indices.view(-1))
        total_loss += loss.item()
        loss.backward()
        self.optim.step()
        self.optim.zero_grad()
    return total_loss / len(train_loader)
```

訓練時的 Loss function 使用 Cross Entropy。Forward 流程如下：首先把圖片量化，然後從 01 的均勻分布出 sample 一個值輸入 gamma function 得到 mask 的比率，接著將 mask 隨機打散蓋住 ground truth，得到待預測的圖像，最後輸入 transformer 得到預測 indices。而訓練流程就是透過 forward 得到預測跟 GT，這兩個去算 Loss，得到梯度後更新參數。

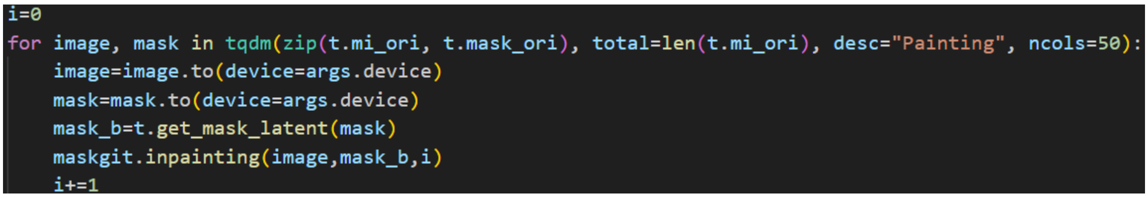
C. The details of your inference for inpainting task (iterative decoding)



```
def inpainting(self, z_indices, mask, ratio):
    masked_indices = z_indices.clone().view(1, 256)
    masked_indices[mask] = self.mask_token_id
    logits = self.transformer(masked_indices)

    probs = F.softmax(logits, dim=-1)
    z_indices_predict_prob, z_indices_predict = torch.max(probs, dim=-1)

    g = -torch.log(-torch.log(torch.rand_like(z_indices_predict_prob)))
    temperature = self.choice_temperature * (1 - ratio)
    confidence = z_indices_predict_prob + temperature * g

    confidence_masked = confidence.masked_fill(~mask, -float('inf'))
    _, indices = torch.sort(confidence_masked, descending=True)
    num_tokens_to_keep = int((1 - ratio) * mask.sum())
    indices_to_unmask = indices[0, :num_tokens_to_keep]

    mask_bc = mask.clone()
    mask_bc[0, indices_to_unmask] = False

    return z_indices_predict, mask_bc
```



```
        _, z_indices = self.model.encode_to_z(image) #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 = self.model.gamma(step / self.total_iter)

            z_indices_predict, mask_bc = self.model.inpainting(z_indices_predict, mask_bc, ratio)
```



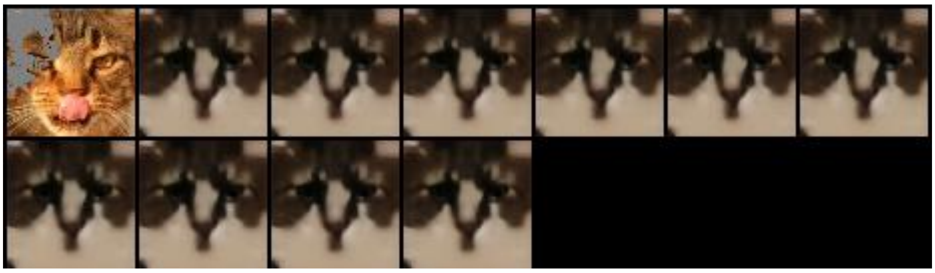
```
i=0
for image, mask in tqdm(zip(t.mi_ori, t.mask_ori), total=len(t.mi_ori), desc="Painting", ncols=50):
    image=image.to(device=args.device)
    mask=mask.to(device=args.device)
    mask_b=t.get_mask_latent(mask)
    maskgit.inpainting(image,mask_b,i)
    i+=1
```

在 inpainting 時，除了第一輪以外，每一輪會使用上一輪預測的輸出(indices 跟 mask)做為輸入做預測。而一輪中流程如下：首先將上了 mask 的 indices 交給 transformer 得到預測結果，接著用 soft max 得到每一種 token 的預測機率，然後保留每個 vector 最大機率值之 token 的 index 與其機率，然後加入噪音讓預測結果可以更加靈活，從而算出模型對每個 vector 的信心程度，之後把沒有被 mask 的信心度設為負無窮以免被誤算，根據 ratio 可以知道要被 unmask 的數量，從預測 token 中選出信心最大的那幾個，將其從 mask 中去除，得到預測後的 mask 以及預測 indices。如果覺得還不夠可以把信心度不好的預測 indices 替換回原本的 indices，讓最後 sweet spot 可以保留原本的圖像，從而得到比較好的表現，但是這邊並沒有這麼做。

3. Discussion (bonus: 10%)

A. Anything you want to share

我在訓練時觀察到 Learning rate 在 1e-3 跟 1e-4 中間訓練出來的結果有非常大的差距，若只觀察訓練的 Loss 可能沒什麼差別，分別是 3.xxx 跟 2.xxx，但在 inpainting result 上天差地遠。



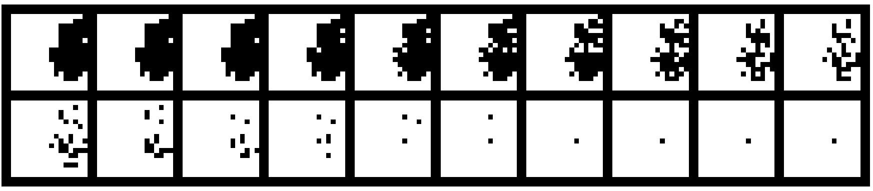
當 Learning rate 為 1e-3 以上時，訓練出的模型無論輸入為何，最終都會輸出同一張貓臉，但只要設為 1e-4 之後，在第一個 epoch 訓練完後便可以大致還原原本圖像的樣貌。

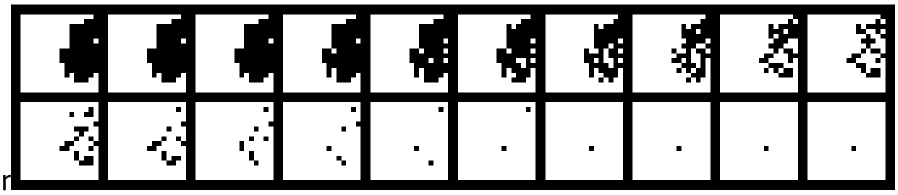
Experiment

Part1: Prove your code implementation is correct (30%)

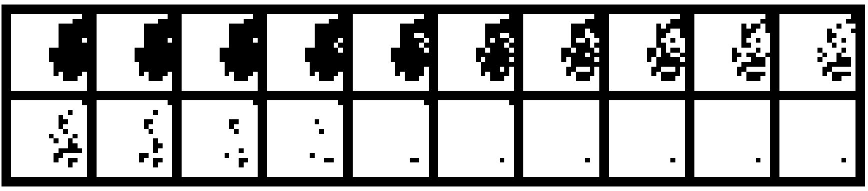
1. Show iterative decoding. (Mask in latent domain, Predicted image)

•cosine





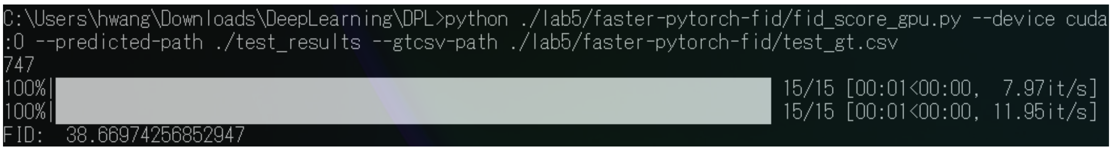
- linear



- square

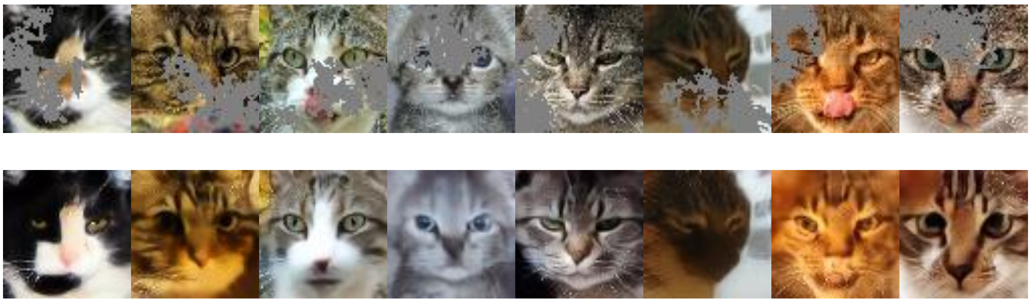
Part2: The Best FID Score (20%)

- Screenshot



```
C:\Users\hwang\Downloads\DeepLearning\DPL>python ./lab5/faster-pytorch-fid/fid_score_gpu.py --device cuda
:0 --predicted-path ./test_results --gtcsv-path ./lab5/faster-pytorch-fid/test_gt.csv
747
100%|                                                              | 15/15 [00:01<00:00,  7.97it/s]
100%|                                                              | 15/15 [00:01<00:00, 11.95it/s]
FID:  38.66974256852947
```

- Masked Images v.s MaskGIT Inpainting Results



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

(執行前請先確認檔案路徑都在正確位置，程式碼須放在名為 lab5 的資料夾中)

若要訓練，請執行 ：$ python ./lab5/training_transformer.py

learning-rate = 1e-4

batch-size = 20

epochs = 20

optimizer = torch.optim.Adam

scheduler = lr_scheduler.StepLR, step_size = 10, gamma = 0.1

若要測試，請執行 ：$ python .\lab5\inpainting.py

batch-size = 1

sweet-spot = 7

total-iter = 20

mask-func = cosine