

Lab3 Report

Deep Learning

MaskGIT for Image Inpainting

姓名：陳科融

學號：314551010

July 19, 2025

1. Introduction

本次實驗旨在實作 MaskGIT (Masked Generative Image Transformer) 模型，並將其應用於 圖像修復 (Image Inpainting) 任務。該模型以一個雙向 Transformer 架構為核心，透過學習重建被遮蔽的視覺 token，達成有效的圖像生成目標。

在實驗中，我們採用預先訓練完成的 VQGAN 模型作為第一階段，用以將輸入圖像編碼為離散的 token 序列，為後續 Transformer 的訓練提供基礎資料。根據任務要求，本次實作的重點包括以下三個部分：

1. 自行從零開始訓練第二階段的 Transformer 模型；
2. 設計並實作一個用於圖像修復的迭代式解碼演算法；
3. 建立一個自定義的多頭注意力機制 (Multi-head Attention)。

模型訓練與測試皆在提供的貓臉資料集上進行，並採用 FID (Fréchet Inception Distance) 分數作為主要評估指標，衡量生成圖像與真實圖像之間的相似程度。我們亦深入探討三種不同的遮罩排程策略 (linear、cosine、square) 對推論階段圖像品質的影響，進一步驗證其在修復效果上的差異性與優劣。

2. Implementation Details

The detail of your model

2.1.1 Self-Attention

假設輸入資料為 x_1 與 x_2 ，首先會經過 Input Embedding 層，將原始輸入映射至更高維度的向量表示，分別產生出 a_1 與 a_2 。這個嵌入步驟能夠幫助模型更有效地理解資料中的語意與結構特徵。

接下來，這些向量會分別乘上三個參數矩陣： W_q (Query)、 W_k (Key) 以及 W_v (Value)，以生成對應的 q 、 k 、 v 。這三組向量是 Self-Attention 機制的核心，負責計算不同位置之間的關聯程度。

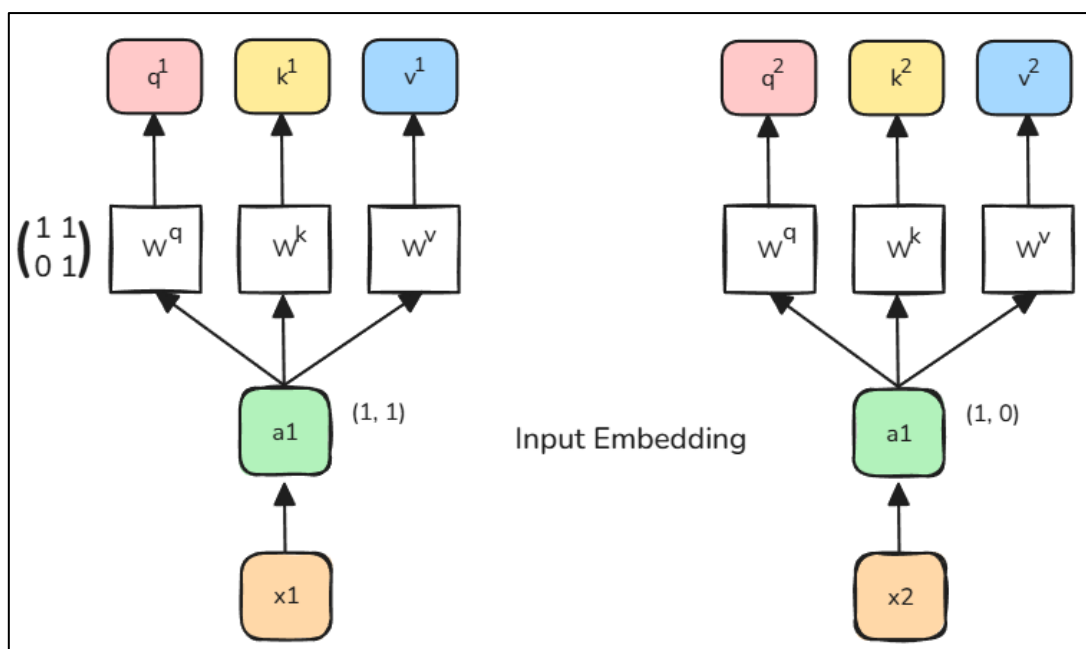


圖 1. Self-Attention input 到 q、k 和 v

具體來說，系統會利用查詢向量 q 與其他位置的鍵向量 k 進行點積（dot product）運算，以獲得注意力權重。這些權重經過 softmax 正規化後，再與對應的位置的值向量 v 進行加權平均，從而得到最終的輸出表示。這樣的設計使模型能夠根據整體上下文資訊，自動調整每個輸入位置對其他位置的關注程度，進而捕捉更深層的語意關聯與長距依賴。

$$Attention(Q, K, V) = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right)V$$

q : query(to match other) $q^i = a^i W^q$	$q^i = a^i W^q$
k : key(to be match)	$k^i = a^i W^k$
v : information to be extracted	$v^i = a^i W^v$

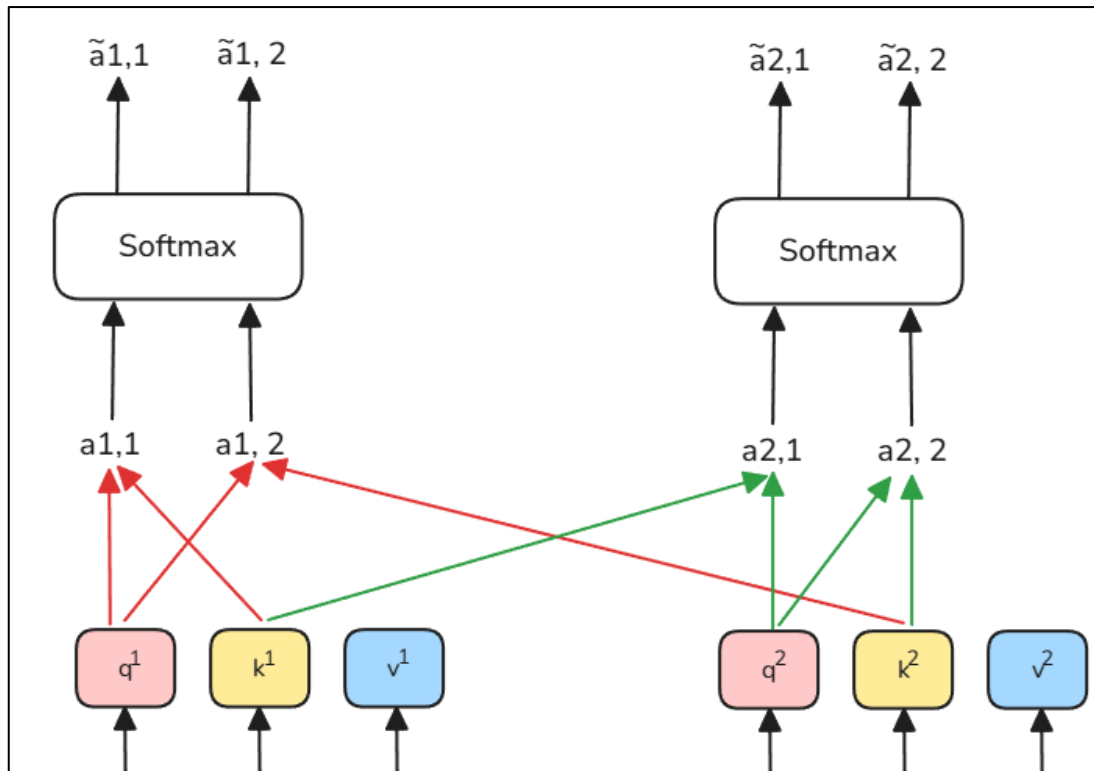


圖 2. Self-Attention 做 Scaled Dot-Product Attention

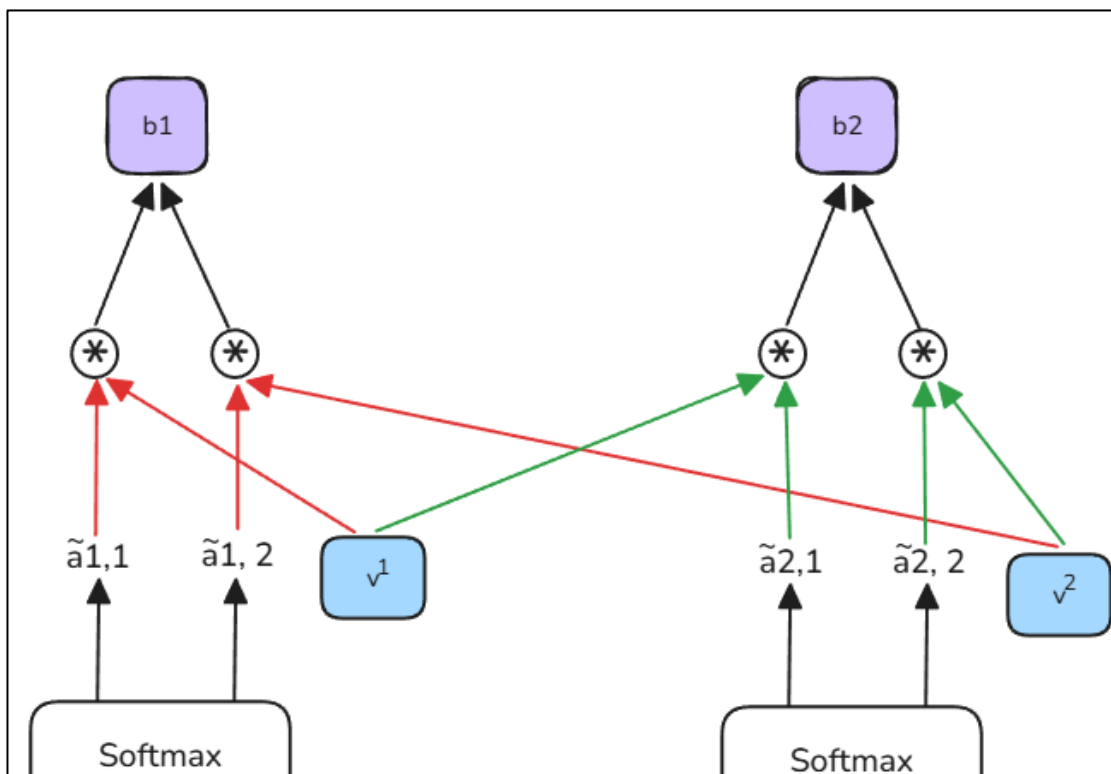


圖 3. Self-Attention 最後乘上 V，進行相加

2.1.2 Multi-head Self-Attention

在 Self-Attention 機制中，已經可以有效地捕捉輸入序列中各位置之間的相對關聯性。然而，為了讓模型能夠從多個子空間（subspace）中學習不同層次的語意資訊，Transformer 架構中引入了 Multi-head Self-Attention 機制。

具體來說，當我們已經從輸入中計算出查詢（q）、鍵（k）、與值（v）向量後，Multi-head 機制會將這些向量在維度上拆分為多個部分，平均分配給每一個注意力頭（head）。每一個 head 都會各自進行一次獨立的 Self-Attention 運算。這個拆分操作可以視為在不同的表示子空間上，分別捕捉輸入資料的語意特徵。

在每一個 head 中，q、k、v 向量會分別經過不同的線性映射參數（ W_q 、 W_k 、 W_v ），接著進行注意力計算，得到一組對應的輸出。這些來自多個 head 的注意力結果，最後會被串接（concatenate）起來，再通過一個線性變換層進行整合，作為最終的 Multi-head Attention 輸出。

這種機制讓模型不僅能捕捉單一語意關係，更能從多角度同時觀察輸入序列中的資訊，增強模型的表現力與語意理解能力。

$$d_k = d_v = d_{model}/h$$

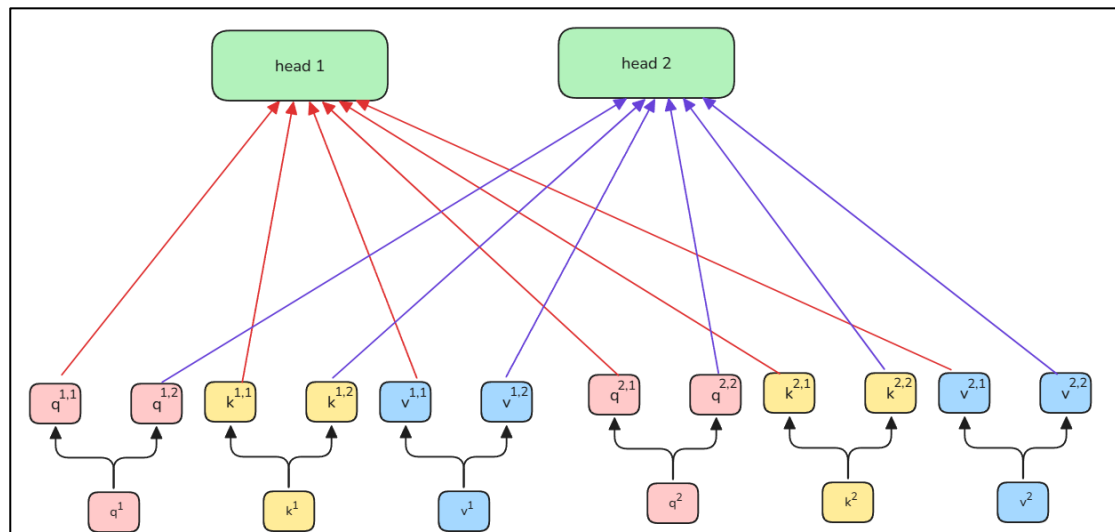


圖 4. 將 q、k 和 v 平均分配給每一個 head

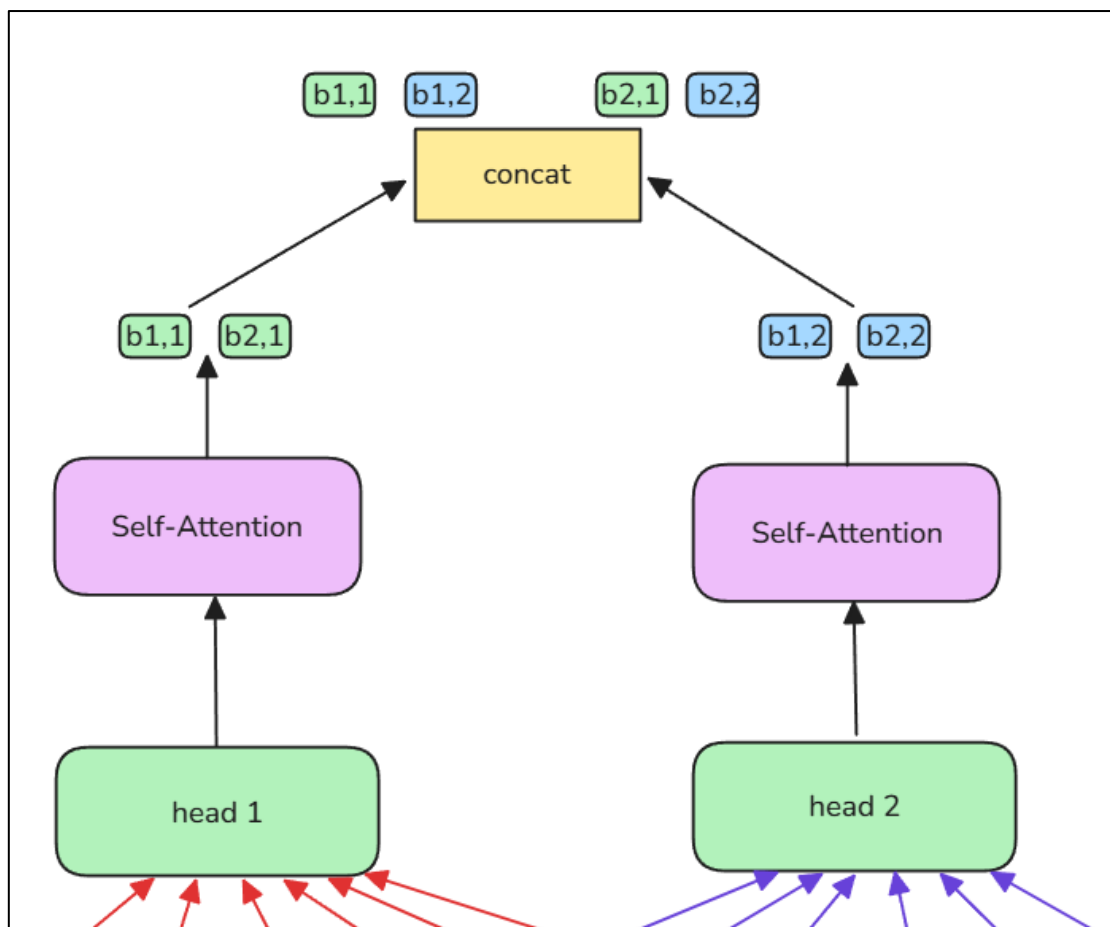


圖 5. 每個 head 做 self-attention，最後 concat 起來

The details of your stage2 training

2.1.1 MVTM

Masked Vision Transformer Modeling(MVTM)是訓練的核心機制。在每一次 forward pass 中，整個流程如下：

Token 編碼：輸入影像 x 經由 `encode_to_z` 轉換為 token 索引 `ground_truth_indices`。

1. **隨機遮罩**：生成隨機遮罩 `random_mask`，將部分 token 以 [MASK] 取代，形成遮罩後的輸入 `masked_input_indices`。
2. **Transformer 預測**：將遮罩後的序列輸入 `bidirectional Transformer` 中，進行預測。
3. **輸出與對齊**：模型輸出為 `logits` 分佈，訓練目標為原始的 token 索引 `ground_truth_indices`。

這項設計的核心目標為訓練 Transformer 能夠根據上下文資訊，準確還原被遮罩的 token，具備 bidirectional context 的學習能力。

```
##TODO2 step1-3:
def forward(self, image_batch):
    # 訓練時的前向傳播
    # image_batch: (B, C, H, W) -> (批次大小, 通道數, 高度, 寬度)

    # 1. 將輸入影像編碼成離散的 token 序列
    _, ground_truth_indices = self.encode_to_z(image_batch)

    # 2. 隨機生成遮罩
    # 建立一個與 token 序列形狀相同、值全為 MASK token ID 的張量
    mask_token_tensor = torch.full_like(ground_truth_indices, self.mask_token_id)
    # 依據 0.5 的機率，獨立為每個 token 位置決定是否要遮罩
    random_mask = torch.bernoulli(0.5 * torch.ones_like(ground_truth_indices, dtype=torch.float)).bool()

    # 3. 產生 Transformer 的輸入
    # 使用遮罩，將部分真實 token 替換為 MASK token
    masked_input_indices = torch.where(random_mask, mask_token_tensor, ground_truth_indices)

    # 4. 透過 Transformer 進行預測
    # Transformer 的目標是預測出被遮罩位置的原始 token
    predicted_logits = self.transformer(masked_input_indices)

    # 5. 回傳預測結果和真實標籤以計算損失
    return predicted_logits, ground_truth_indices
```

程式碼 1. MVTM 程式碼架構

2.1.2 Forward and Loss

在訓練過程中，模型學習的任務實質上為一個多分類問題：預測 [MASK] 所代表的原始 token。為此，我們採用 Cross Entropy Loss 作為損失函數，並透過以下步驟完成一次訓練流程：

- 每個 batch 中，將輸入影像送入模型，產生 logits（預測分佈）與 z_indices（真實索引）。
- 損失計算中，透過 ignore_index 參數忽略未被遮罩的位置，以避免模型被錯誤地懲罰。
- 最後進行梯度反向傳播與權重更新，並根據累積梯度設定進行參數更新。

這樣的設計使模型只專注於預測遮罩區域的 token，進一步強化其對影像局部與全域關係的建模能力。

```

def train_one_epoch(self, train_dataloader, epoch, args):
    # 單一訓練週期 (epoch) 的邏輯
    losses = [] # 用於記錄每個 batch 的損失
    # 使用 tqdm 顯示進度條
    pbar = tqdm(train_dataloader, desc=f"Epoch {epoch}/{args.epochs}", unit="batch")
    self.model.train() # 將模型設定為訓練模式
    for batch_idx, (images) in enumerate(pbar):
        # 將圖片資料移至指定設備
        x = images.to(args.device)
        # 模型前向傳播, 得到預測的 logits 和真實的 token 索引
        logits, z_indices = self.model.forward(x)
        # 計算交叉熵損失, 忽略 MASK token 的位置
        loss = F.cross_entropy(logits.view(-1, logits.size(-1)), z_indices.view(-1), ignore_index=-1)
        # 反向傳播計算梯度
        loss.backward()
        losses.append(loss.item())

        # 梯度累積: 每 accum_grad 個 batch 才更新一次權重
        if (batch_idx + 1) % args.accum_grad == 0:
            self.optim.step() # 更新模型權重
            self.optim.zero_grad() # 清除梯度

        # 在進度條上顯示當前的損失和學習率
        pbar.set_postfix(loss=loss.item(), lr=self.optim.param_groups[0]['lr'])

        # 如果啟用 wandb, 則記錄當前 batch 的資訊
        if args.use_wandb and batch_idx % args.wandb_log_interval == 0:
            wandb.log({
                "batch": batch_idx + epoch * len(train_dataloader),
                "train_batch_loss": loss.item(),
                "learning_rate": self.optim.param_groups[0]['lr']
            })

    avg_loss = np.mean(losses) # 計算整個 epoch 的平均損失
    print(f"Epoch {epoch}/{args.epochs}, Average Training Loss: {avg_loss:.4f}")

    # 如果有設定學習率排程器, 則更新學習率
    if self.scheduler is not None:
        self.scheduler.step()

    return avg_loss

```

程式碼 2. Training_transformer

The details of your inference for inpainting task

在 Inpainting 的每一次迭代中, 模型會依序接收以下資訊作為輸入: 本次迭代的 token 序列、目前的遮罩狀態、初始被遮罩的位置數量、當前的遮罩比例 (ratio), 以及所選擇的遮罩排程函數 (mask function)。

初始階段, 系統會根據輸入的遮罩, 將原始 token 序列中部分位置以 [MASK] token 替代。接著, 將此遮罩後的序列輸入至 Transformer 模型中, 進行預測。Transformer 輸出每個被遮罩位置對應於所有可能 token 的機率分佈 (logits)。根據這些分佈, 我們會為每個位置進行採樣, 生成一組新的 token 序列作為目前迭代的輸出。

之後, 我們根據當前的 ratio 和所選的遮罩函數來計算此次迭代應保留多少比例的預測 token。這裡所指的「保留」僅針對原始遮罩為 True 的位置。模型會依據每個位置預測機率的最大值 (代表信心度) 進行排序, 保留信心度

較高的位置，其餘位置則再次以 [MASK] token 遮罩，以便在下一輪迭代中重新預測。

最終，該迭代會回傳兩個結果：本次經過採樣後的 token 序列，以及更新後的遮罩狀態，供下一次迭代使用。

```
def inpainting(self, current_indices, current_mask, total_masked_count, iteration_ratio,
               # 執行單步迭代式解碼 (inpainting)

    # 1. 準備 Transformer 輸入：將被遮罩的位置替換為 MASK token
    masked_input = torch.where(current_mask, self.mask_token_id, current_indices)

    # 2. 透過 Transformer 預測所有 token 的機率分佈
    logits = self.transformer(masked_input)
    token_probabilities = torch.softmax(logits, dim=-1)

    # 3. 從機率分佈中取樣，得到預測的 token
    predicted_indices = torch.distributions.Categorical(logits=logits).sample()

    # 4. 確保預測出的 token 不會是 MASK token
    while torch.any(predicted_indices == self.mask_token_id):
        predicted_indices = torch.distributions.Categorical(logits=logits).sample()

    # 5. 將預測出的 token 填入當前被遮罩的位置，形成一個完整的 token 序列
    updated_indices = torch.where(current_mask, predicted_indices, current_indices)

    # 6. 取得每個預測 token 的機率 (信心度)
    predicted_probs = token_probabilities.gather(-1, predicted_indices.unsqueeze(-1)).squeeze(-1)
    # 對於本來就未遮罩的位置，我們不考慮其信心度，設為無限大
    predicted_probs = torch.where(current_mask, predicted_probs, torch.inf)

    # 7. 根據 Mask Scheduling 函數計算下一步要保留的遮罩比例
    next_mask_ratio = self.gamma_func(mask_schedule_func)(iteration_ratio)

    # 8. 計算下一步要保留的遮罩數量
    num_masks_to_keep = torch.floor(total_masked_count * next_mask_ratio).long()

    # 9. 計算每個 token 的排序分數 (Gumbel-Max Trick)
    # 透過加入 Gumbel 噪音和溫度係數，增加取樣的隨機性，避免模型過於單調
    gumbel_noise = torch.distributions.Gumbel(0, 1).sample(predicted_probs.shape).to(predicted_probs)
    temperature = self.choice_temperature * (1 - iteration_ratio)
    ranking_scores = predicted_probs + temperature * gumbel_noise

    # 10. 決定下一步的遮罩
    # 找出排序分數最低的 token，這些是模型最不確定的部分，將在下一步繼續被遮罩
    sorted_scores, _ = torch.sort(ranking_scores, dim=-1)
    # 取得分數的截斷閾值
    confidence_threshold = sorted_scores[:, num_masks_to_keep].unsqueeze(-1)
    # 分數低於閾值的 token 位置，即為下一步的遮罩
    next_step_mask = (ranking_scores < confidence_threshold)

    return updated_indices, next_step_mask
```

程式碼 3. Inpainting 程式架構

3. Discussion

3.1 Train

3.1.1 執行指令(100 場)

cosine	<code>python training_transformer.py --gamma-type cosine --device cuda:0 --batch-size 40 --epochs 100</code>
linear	<code>python training_transformer.py --gamma-type linear --device cuda:0 --batch-size 40 --epochs 100</code>
square	<code>python training_transformer.py --gamma-type square --device cuda:0 --batch-size 40 --epochs 100</code>

3.1.2 wandb 指令(100 場)

cosine	<code>python training_transformer.py --use_wandb --wandb_project "MaskGit-Lab3" --wandb_run_name "gamma_cosine" --gamma-type cosine --device cuda:0 --batch-size 40 --epochs 100</code>
linear	<code>python training_transformer.py --use_wandb --wandb_project "MaskGit-Lab3" --wandb_run_name "gamma_linear" --gamma-type linear --device cuda:0 --batch-size 40 --epochs 100</code>
square	<code>python training_transformer.py --use_wandb --wandb_project "MaskGit-Lab3" --wandb_run_name "gamma_square" --gamma-type square --device cuda:0 --batch-size 40 --epochs 100</code>

3.2 Inpainting

原先的程式碼只有 test_results，我有增加 masked images 儲存到 masked_input 資料夾中，inpainting 可直接執行，使用預設參數。因為怕誤刪到照片，所以切換 scheduler 必須手動刪除(備份用)。

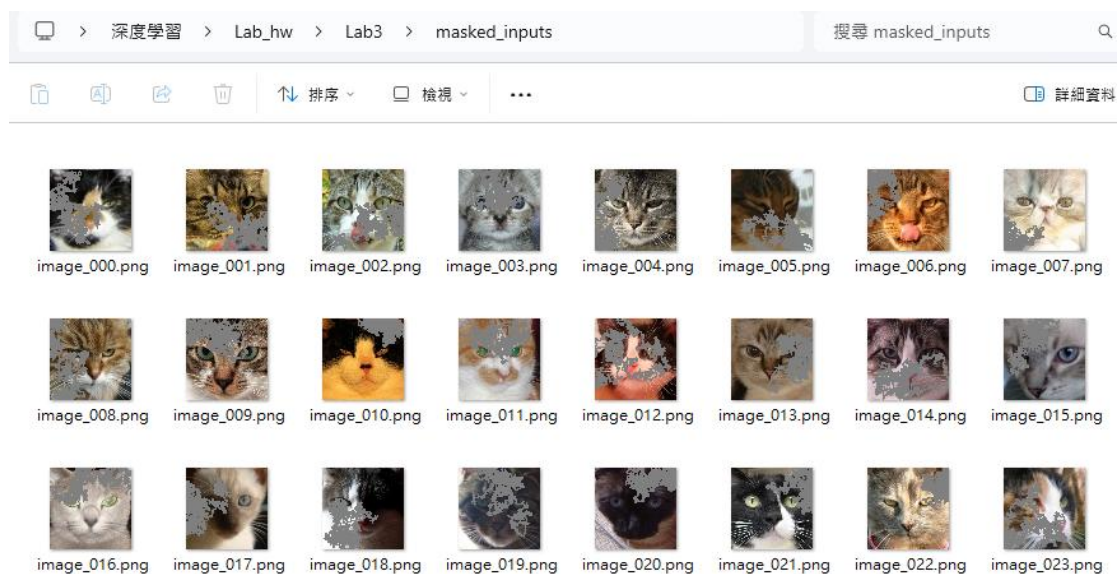


圖 6. Mask_images 儲存位置

3.2.1 指令參數

參數名稱	指令參數	說明	預設值
遮罩排程函數	--mask-func	控制遮罩比例如何隨迭代步驟遞減。選項： cosine、linear、square	cosine
總迭代次數	--total-iter <數字>	設定整個推論過程的總迭代次數	7
sweet spot	--sweet-spot <數字>	若希望提早在某一個步驟停止，可設定此值；設 為 -1 代表不使用	-1

3.2.2 執行指令(預設)

cosine	python Lab_hw/Lab3/inpainting.py --mask-func cosine --total-iter 7 --sweet-spot -1
linear	python Lab_hw/Lab3/inpainting.py --mask-func linear --total-iter 7 --sweet-spot -1
square	python Lab_hw/Lab3/inpainting.py --mask-func square --total-iter 7 --sweet-spot -1

3.2.3 指令(調整 sweet spot 和 iteration，固定使用 cosine.pth)

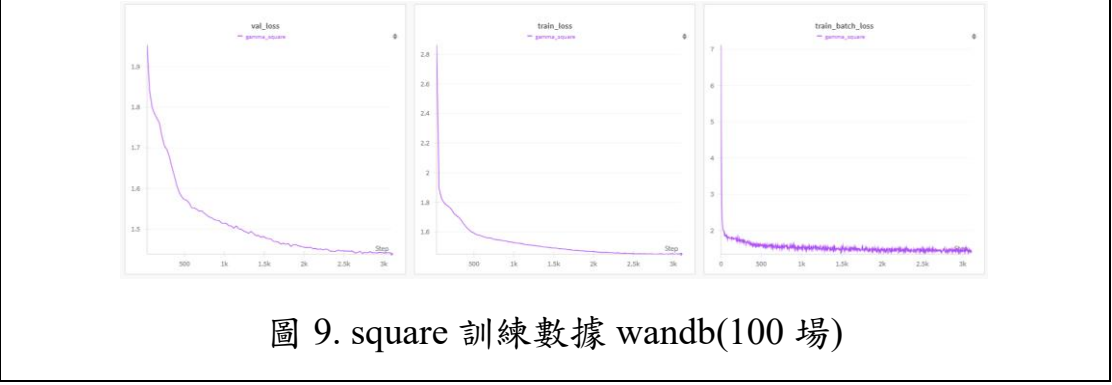
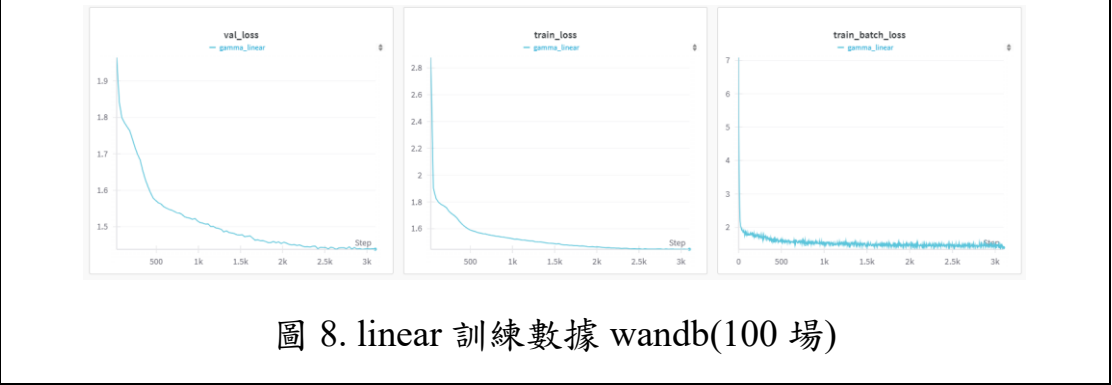
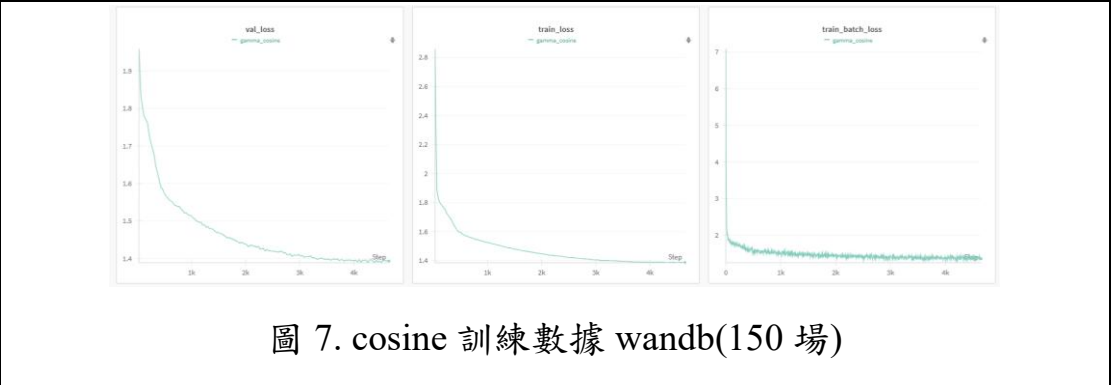
cosine (針對 sweet spot)	python inpainting.py --mask-func cosine --total-iter 7 --sweet-spot 8
linear (針對 sweet spot)	python inpainting.py --mask-func linear --total-iter 7 --sweet-spot 8
square (針對 sweet spot)	python inpainting.py --mask-func square --total-iter 7 --sweet-spot 8
cosine (針對 total-iter)	python inpainting.py --mask-func cosine --total-iter 12 --sweet-spot 8
linear (針對 total-iter)	python inpainting.py --mask-func linear --total-iter 12 --sweet-spot 8
square (針對 total-iter)	python inpainting.py --mask-func square --total-iter 12 --sweet-spot 8

3.3 FID Score

3.3.1 執行指令

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

3.4wandb 數據結果



Name	3 visualized	State	Notes	Use	Tag	Created	Runtime	Transf
gamma_square		Finished	Add notes	kramer		6h ago	4h 52m 6s	768
gamma_linear		Finished	Add notes	kramer		12h ago	4h 50m 37s	768
gamma_cosine		Finished	Add notes	kramer		21h ago	7h 9m 46s	768

圖 10. wandb 紀錄

3.4.1 每個 Epoch 內的 Batch 記錄方式：

在模型訓練過程中，設定為每經過固定數量的 batch（每 10 個 batch）就進行一次訓練記錄，用來追蹤損失值變化和訓練狀況，方便後續分析模型學習情形。

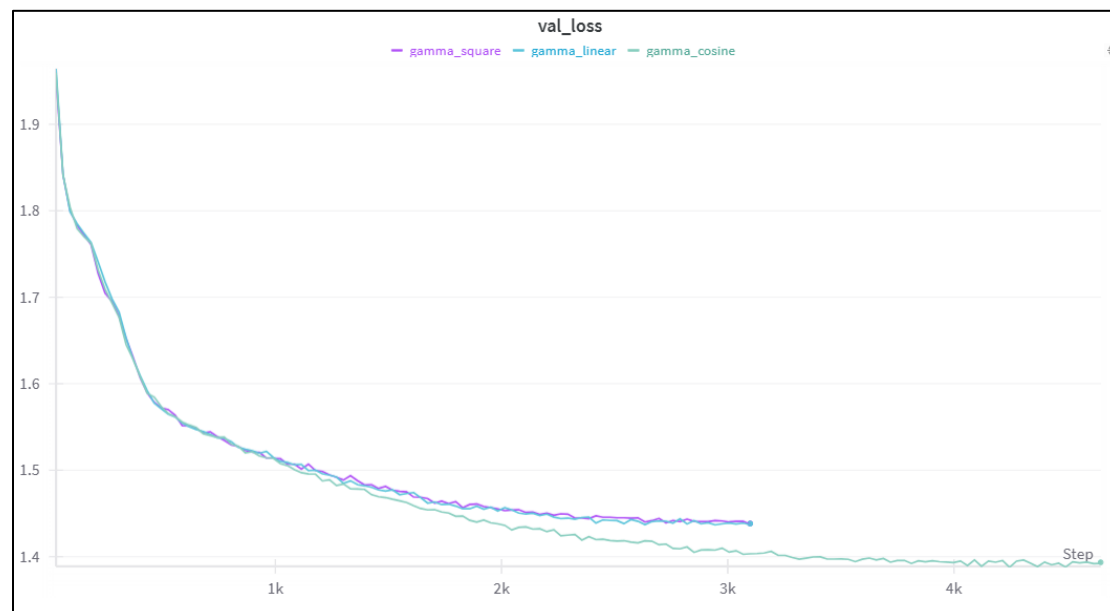


圖 11. Evaluate

3.5 sweet spot 與 total-iter 調整

Mask-func	sweet spot	Total-iter	FID score
cosine	-1(no use)	7	30.36
cosine	-1(no use)	7	31.56
cosine	-1(no use)	7	32.18
cosine	8	7	30.11
linear	8	7	29.73
square	8	7	30.46
cosine	8	12	29.77
linear	8	12	29.78
square	8	12	30.49

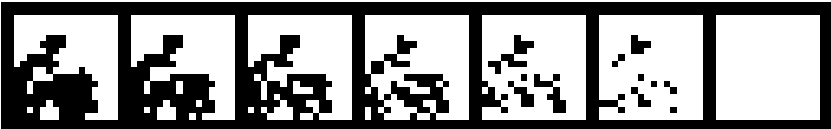

4. Experiment Score

4.1 Prove your code implementation is correct

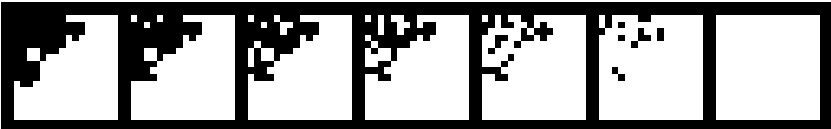

Mask Scheduling Configuration

- **Mask Scheduling Function:** cosine
- **Number of Iterations (T):** 7
- **Sweet Spot (t):** -1 （不使用 sweet spot）


4.1.1 cosine

Mask in laten domain	
Predicted image	

4.1.2 Linear

Mask in laten domain	
Predicted image	

4.1.3 square

Mask in laten domain	
Predicted image	

5. 參考資料

- [1] J. Yin, “NYCU Deep Learning Course Code,” *GitHub Repository*, [Online]. Available: <https://github.com/jayin92/NYCU-deep-learning>
- [2] alu98753, “NYCU Deep Learning 2025,” *GitHub Repository*, [Online]. Available: <https://github.com/alu98753/NYCU-Deep-Learning-2025>
- [3] Part-time-Ray, “DLP: Deep Learning Practice,” *GitHub Repository*, [Online]. Available: <https://github.com/Part-time-Ray/DLP>
- [4] Hank Lin, “NYCU Deep Learning Course Labs,” *GitHub Repository*, [Online]. Available: <https://github.com/hank891008/Deep-Learning>
- [5] 哔哩哔哩視頻, “Transformer Self-Attention 與 Multi-head Attention 機制講解,” *Bilibili*, [Online Video], Available: <https://www.bilibili.com/video/BV15v411W78M>
- [6] CSDN User qq_43426908, “PyTorch 中最全的学习率调整方式,” *CSDN Blog*, May 2022. [Online]. Available: https://blog.csdn.net/qq_43426908/article/details/125019923
- [7] CSDN User qq_44681809, “基于 MaskGIT 的图像生成原理解析与实战,” *CSDN Blog*, Jan. 2024. [Online]. Available: https://blog.csdn.net/qq_44681809/article/details/135462813
- [8] CSDN User qq_42208244, “MaskGIT 论文笔记+原理分析,” *CSDN Blog*, Jul. 2023. [Online]. Available: https://blog.csdn.net/qq_42208244/article/details/131284698
- [9] 知乎专栏, “图像生成之 MaskGIT: 逐步预测遮挡区域,” *知乎*, Dec. 2023. [Online]. Available: <https://zhuanlan.zhihu.com/p/618235198>

使用的 AI 工具：

✧ **ChatGPT-4o** (由 OpenAI 提供)

用於問答、報告草稿撰寫、英文轉中文及論文說明。

✧ **Google AI Studio**

作為補充問答工具，用於模型概念、架構或函式撰寫思路輔助。

✧ **VSCoDe + Google Gemini 2.5 Pro**

協助撰寫與補全 Python 程式碼，並支援簡易 debug。

✧ **VSCoDe GitHub Copilot (GPT-4o)**

用於實作過程中進行即時程式補全、語法建議與除錯輔助。