**NYCU DL Lab5**

**MaskGIT for Image Inpainting  
 313552049 鄭博涵**

1. **Introduction**

**在這次實驗中, 我們的目標是訓練 MaskGIT 中的雙向 Transformer，並使用它來解決Inpainting的任務, 在實作方面，我們需要完成三個部分, 分別是Multi-head Attention Layer的實作、雙向 Transformer 的train strategy，以及最後用於圖像修補的Iteration Decoding, 最後, 我們比較不同的遮罩率增減方式(cosine, linear, square) 對圖像修補結果的影響，並使用 FID 作為評估指標**

**最終我的FID是38.9**

1. **Implementation Details**
2. **The details of your model**

**(Multi-Head Self-Attention)**

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

**\_\_init\_\_(): 首先在模組初始化時會設置一些參數, 其中num\_head是指將注意力機制分成了多少個平行的注意力頭, 每個頭會獨立地學習不同部分的特徵, attn\_drop則是注意力機制的drop參數**

**然後將key, query, value三個向量分別設置了nn.Linear, 用來將輸入的特徵轉換到合適的維度, 最後的輸出通過一個nn.Linear進行投影**

**Forward(): 首先B, T, D指的是batch\_size, num\_image\_tokens, dim**

**再來通過線性轉換層計算出key, query, value張量, 然後將其分成多個head, 每個head的維度為 D // num\_head**

**然後使用query, key來計算注意力分數, 這個分數矩陣表示序列中不同位置之間的關聯性, 再對注意力分數進行softmax操作, 使其正規化為概率分佈**

**y = att @ v : 將注意力分數與value相乘, 得到最終的注意力加權表示**

**y = y.transpose(1, 2).contiguous().view(B, T, D): 將所有頭的輸出重新排列, 並且將輸出形狀設為(Batch\_size, num\_image\_tokens, dim)的張量, 最後再return self.proj(y): 表示通過投影層進行線性轉換**

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

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**\_\_init\_\_(): 首先load\_vqgan()會讀取配置文件, 並根據其中的參數初始化VQGAN, 其中我想解釋的參數有:**

**self.num\_image\_token: 是model將處理的圖像token數量, self.mask\_token\_id: 是遮罩token的id**

**self.choice\_temperature: 是一種溫度參數, 較低時會使model更自信, 較高時則會增加預測的隨機性**

**再來是Encoder\_to\_z:**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

**這邊的功能是利用VQGAN model的編碼器來生成圖像的潛在向量(codebook\_mapping) 和對應的量化索引(codebook\_indices)**

**接著是Gamma\_function:**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**gamma\_func的主要功能是生成遮罩率(ratio), 這個遮罩率決定了圖像中的token有多大比例被mask, 其中有Linear, cosine, square這三種不同的遮罩率排成策略**

**MVTM: 全名是Multi-Variate Transformer Masking, 在MaskGIT中指的是一種遮罩策略, 用來隨機選擇圖像中的一部分token進行遮罩, 然後用transformer model來預測這些被mask的token**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

**在forward當中, 透過mask = torch.rand\_like(z\_indices, dtype=torch.float) < ratio來實現MVTM的核心概念, 即通過隨機遮罩來模擬圖像信息缺失的情況**

**而z\_indices\_input = z\_indices.masked\_fill(mask,self.mask\_token\_id)**

**則是被選中的遮罩部分會被填充為一個特殊的mask\_token\_id, 然後利用Transformer model根據上下文來預測被遮罩的信息**

**(logits = self.transformer(z\_indices\_input) 的部分)**

**Loss: 一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述**

**\_\_init\_\_: 初始化了model, device, lr, optim等基本參數, 並設置了learningrate scheduler及log的保存路徑**

**Train\_one\_epoch: 負責執行model在一個train epoch中的所有步驟, 在每個batch中隨機生成mask ratio並將其應用於輸入數據,**

**之後計算loss使用cross\_entropy, 並透過backward來更新model weight, 在每次訓練後使用梯度裁減以防止梯度爆炸, 並累積損失以計算平均損失**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**進行多個epoch的training, 每個epoch都包括train\_one\_epoch和eval\_one\_epoch, 每次訓練完成後, 會計算train\_loss以及val\_loss,**

**並加入到Tensorboard中**

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

**inpainting(在VQGAN\_Transformer裡):**

**一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述**

**使用torch.where函數將遮罩位置的z\_indices替換為特殊的mask\_token\_id, 以標記需要預測的部分, 再將遮罩後的 z\_indices\_input 傳遞給 Transformer 模型進行預測, 並通過 softmax 轉換成概率分佈, 然後找出最有可能的 token 值及其對應的最大概率 , 接著計算ratio以及use Gumbel噪聲來引入隨機性, 並結合temperature來調整信心值, 避免model過於自信**

**之後使用torch.topk函數來選出信心值最低的token, 更新mask, 以確定哪些位置在下一步中需要進行進一步的預測,**

**最後返回更新後的z\_indices\_predict和mask\_bc, 會用於後續的inpainting中**

**inpainting (total iteration decoding):**

**一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述**

**初始化的部分:**

**maska: 用來保存每次迭代過程中生成的mask**

**imga: 用來保存每次迭代過程中解碼後的圖像**

**ori: 對初始輸入圖像進行標準化變換後的結果, 然後用imag [0] 保存這個基準圖像, 作為遮罩圖像的基線**

**model推理準備的部分:**

**設為評估模式並使用encoder\_to\_z將輸入圖像 編碼為潛在表示, 得到z\_indices, 然後mask\_num計算了初始遮罩中需要被修補的token數量**

**Iteration encoding:**

**一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述**

**先計算ratio, 再使用one iteration encoding的inpainting()**

**來預測當前被遮罩區域的token (z\_indices\_predict), 並更新遮罩(mask\_bc), 然後將每次迭代的遮罩mask\_image 和解碼圖像dec\_img\_ori保存到maska和imaga中, 用於後續分析**

**接著將預測出的token轉換為對應的嵌入向量z\_q, 並透過VQGAN model進行解碼, 得到重建的圖像, 當迭代達到預設的sweet\_spot時, 保存當前的解碼圖像到test\_results資料夾裡, 以便後續的FID計算**

**最終保存結果:**

**test\_results資料夾: 存放sweet\_spot的最佳修補圖像**

**mask\_scheduling資料夾: 保存所有迭代步驟(step)的遮罩圖像**

**imga資料夾: 保存所有迭代步驟的解碼圖像**

1. **Discussion**

**以下是我將sweet\_spot分別設為2, 5, 8, 11之後所計算出來的FID, 皆是用square方法, seed為固定, 所以由此可知, 越後面的step, 所預測出來的效果越差**



**一張含有 文字, 字型, 軟體, 螢幕擷取畫面 的圖片

自動產生的描述**

**我推測原因可能有以下幾點:**

1. **遮罩範圍的逐步縮小:**

**在maskGIT的修補過程中, 隨著迭代的進行, 遮罩範圍會逐步減少, 表示在後期時需要修補的區域越來越小,**

**剩下的未修補可能是較難以預測的部分, 而model在前幾個step中已經生成了相對穩定的圖像, 但後期在修補難以預測的部分時可能會引入一些誤差和不自然的細節, 導致FID上升**

1. **細節誤差逐步放大:**

**前幾次迭代中累積的細微誤差可能會逐漸放大, 所以在後期時, 這些誤差可能會更加顯著, 使得最終生成的圖像與真實圖像相比更不自然**

1. **Model信心值的變化:**

**在多次迭代過程中, model可能對於後期的預測逐漸失去信心, 使得後面的修補步驟更加不可靠, 這時model的預測可能會更依賴於隨機性, 所以導致FID上升**

**Reference:**

[KJLdefeated/NYCU\_DLP\_2024: 2024 NYCU Deep Learning (github.com)](https://github.com/KJLdefeated/NYCU_DLP_2024)