1. Introduction

在本次的實驗報告中詳細的介紹了每個實作的過程，並且在training上也使用了兩個不同的loss計算方式並且做比對，在後半部的inpainting則是以三個不同的mask function在一定的iteration下對FID分數做比對，發現在linear 的mask function且iteration為2時FID分數可以達到最低的分數，並且使用的Loss計算方式為直接比較整體的預測而不是如論文中所提到直接以mask的作Loss計算。

2. Implementation Details

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

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

在Multi-Head Self-Attention的Class中以下圖做設計

一張含有 文字, 螢幕擷取畫面, 寫生, 圖表 的圖片

自動產生的描述

輸入分為 Q (Query)、V(Value) 和 K(Key)，在程式第31-33行時會先將輸入的x進到對應的Linear層並且做view的轉換，轉成[batch\_size,256,16,48]，其中256是輸入latent space的image\_token(16x16)，而16則是代表 num\_heads數量，48則是根據輸入的維度(768 第8行定義)分配給每個num\_heads(除以 num\_heads 數量)得到的數字，為每一個heads的維度。

為了去對 num\_heads做運算，因此會先對V、K、Q做transpose，將num\_heads的維度往前調整(程式35-37行)，接著要算每個head的atten weight(注意力分數)，也是下方 formula 所表示的(程式39-42行)：

一張含有 字型, 文字, 行, 白色 的圖片

自動產生的描述

為了防止training時overfitting所以在計算softmax之後做一層的dropout(程式第40行)，有效避免overfitting和提高泛化能力，做完之後要再次transpose一次，因為先前transpose過了這次是要為了將其concatenate因此在做一次，接著再把它view轉成原先的形狀(程式第43行)，最後則是經過一層Linear作結束。

其中43行部分使用contiguous是保證記憶體的連續性，然後再使用view去做concatenate的動作。

**B. 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

在38-41行encode\_to\_z時，會先透過vqgan的encoder拿到quant\_z，像是否種對應需要修改地方的codebook，但是我們並不會使用到，因為vqgan是以pretrained的model去使用的，在lab05中只需要對bidirectional transformer做訓練，因此在後面後續都是直接以 indices去做選擇，但是從vqgan的encoder拿到的indices會是batch\_size\*16\*16，其中16\*16是latent space的大小，因此會先轉乘[batch\_size,256]的形式做輸出，做後續的動作。

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

如上述所說，第84行直接省略了取quant\_z的變數，而是透過後續轉成transformer(第100行)，在那之前會先做一系列的動作，首先先設隨機產生一個ratio，代表mask的ratio，主要是用於隨機生成mask的比例，而gen\_mask的寫法如下圖所示，拿到mask之後要把mask從np轉乘torch的tensor (第90行)，接著設定transformer的參數，第96 和98行分別代表產生codebook ，需要去讓transformer學習的部分，如果有mask的地方則是使用codebooks值 (預設為1024)，如果沒有的畫則是保留原先的z\_indices，最後產生新的a\_indices給transformer去進行處理，拿到logits，代表每個區域的預測機率分布。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

建立一個table儲存每一個 (i, j) 對之後做打亂，根據前masked\_cnt項去設定mask值，回傳給forward function中。

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

在gamma\_func時則是簡單寫對應的linear、cosine、square function，寫成lambda形式可以在後續使用時直接帶入function。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

在training階段時本實驗嘗試了兩個不同的loss計算方式，第一個是採用論文中所提到的：

一張含有 文字, 字型, 白色, 行 的圖片

自動產生的描述

第二個則是不針對 ，的情況，不針對有mask的情況，而是直接以logits和z\_indices做loss function，兩者在訓練50個epoch後差差異很大，會在後續discussion時做討論，上述說到的兩種loss function，第一種採用論文內部寫法的是於程式碼內42到53行，會先求出logits和z\_indices，然後要帶入mask (程式第50行)，根據batch\_size去產生同樣大小的mask tensor，做後續只針對mask的比較(第51行)，但是可能會存在沒有mask的部分因此第54行為避免出現沒有loss可以計算的情形而直接略過，接著再算期望值就可以了。而另一種方式以cross\_entropy的方式，在程式碼第 46行註解掉的部分，直接計算z\_indices(vqgan encoder結果)和logits(transformer 計算後的結果)的loss。

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

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

此段程式為講解在inpainting時，每一個iterative decoding的動作，輸入為：

* z\_indices：上一個iteration的預測值
* org\_mask\_token：原先要修復的mask圖(於16x16的latent space中)
* masked\_token：上一個iteration的mask圖(於16x16的latent space中)
* current\_iteration：這一次的iteration，主要計算mask ratio
* total\_iteration：總共的iteration次數，主要計算mask ratio
* gamma\_func：所挑選的mask scheduling function

程式114-117行做的跟forward時差不多的動作，先丟給transformer做預測得到logits，接著再將logits轉成機率分布(程式122行)，再對這些分布做排序，排出信心值由高至低的排列(程式125行)，在130-134時則是會針對排序好的信心值昨依些額外的gumbel noise，這一部分是有利於加速inpaining的結果，在137行時會針對沒有mask的位置改成inf，這樣在挑選時都會一值挑選到，保證會一值更新現有需要修正的mask，接著由原圖(org\_mask\_token)去計算所需的mask數量(第142行)，這樣做是因為如果都以上一張圖為例的話，可能會導致一個token中mask的數量越來越少，因此找一個定值(原圖)去乘上一個會隨著iteration而變動的ratio(第145、146行)去計算總供需要mask的數量，最後設產出新的mask和針對該mask產出新的logits給下一個iteration(當作下一個iteration 的z\_indices，如程式碼147到159行)。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

在外層的inpainting時是針對整個iteration去呼叫內部的inpainting做每次的iterative decoding，唯一有修改的地方是 第61行會先產出一開始的z\_indices，在進到iterative decoding去做一次的iteration，拿到新的預測值(z\_indices\_predict) 和新的mask token (mask\_bc)，接著在到下一個iteration去做(程式第74行)，在78行時則是針對z\_indices去做更新，因為每一次都是利用上一次的結果，mask token也是使用上一次的結果，程式的後半段則是保證了在非mask區域是由vqgan所產生的預測(程式86行)。

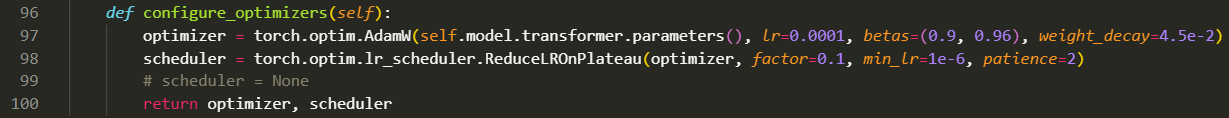
3. Discussion

**A. Anything you want to share**

在設計Loss function時候有提到，使用兩種不同的 Loss function，分別是針對mask的情況下去訓練transformer和直接對所有值(無針對mask的情況下) logits、z\_indices去做cross entropy，發現針對所有值去做cross entropy的效果遠大於針對mask情況下去做loss function和backward。

在訓練這兩個loss function過程中發現收斂速度都很快，收斂的點loss 也很大，但是兩者經過 inpainting的inference後結果卻大相逕庭。

在training時的optimizer和scheduler都是以AdamW，初始lr為0.0001，schedular 用ReduceLROnPlateau，factor為0.1，最小為1e-6。



在結果上為下面兩張圖(所用的種子碼都是一樣的)，分別都訓練50個epoch。

|  |  |
| --- | --- |
| 使用全部的值做Cross Entropy的訓練 | 使用Masked的值做Cross Entropy的訓練 |

發現在Masked時會有train不起來的情況，並且FID在每個iteration都是49左右。

4. Experiment

**A. Show iterative decoding (Prove your code implementation is correct)**

在此都是以最佳FID的結果做表示，並且下方展示的圖都已第42號圖為例子。

Cosine

(Mask in latent domain)



(Predicted image)

一張含有 蠟燭, 藝術 的圖片

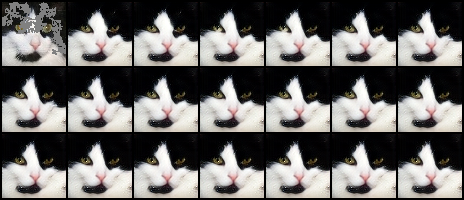
自動產生的描述

Linear

(Mask in latent domain)



(Predicted image)

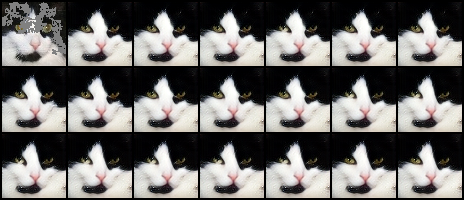


Square

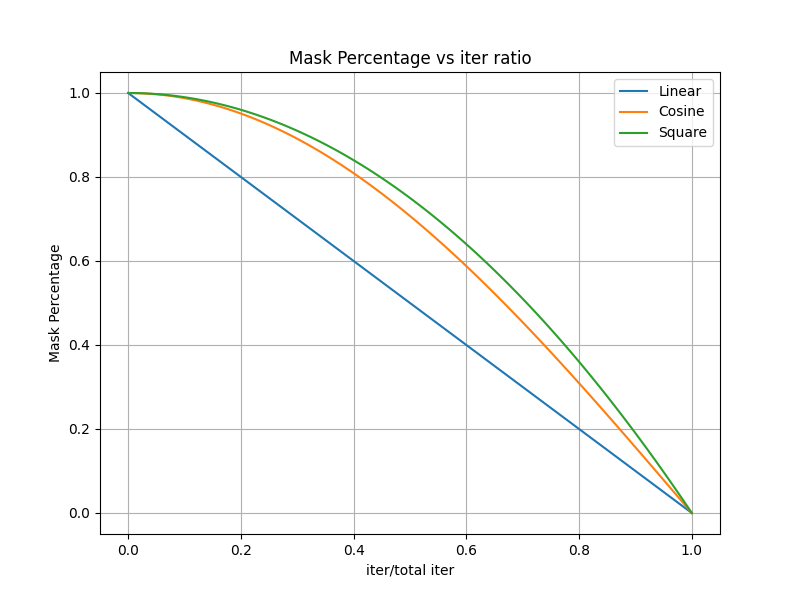
(Mask in latent domain)



(Predicted image)



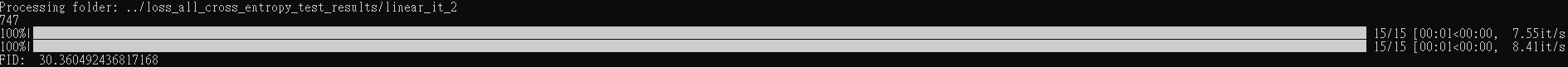
由linear的圖中可以看到，mask減少的範圍是定量的，在square和cosine的比較可以由下圖來看：



可以看到整體覆蓋比率(Mask Percentage)會隨著iteration次數增加而衰減至0，而Linear算是衰減速度較快的，Square在後期才會開始衰減，Cosine則是介於兩者之間，由上方的圖中也可以看到Square在後期一次減少的mask數量較多，Linear較平均等。

**B. The Best FID Score**

**Screenshot(Best FID Score)**



一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

此挑選的結果為使用直接比對所有的值做cross entropy的loss，和使用linear作為inpainting的mask function，此總共的iteration數為19，在第2個iteration次得到的結果(sweet point = 2)。

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linear iteration 2  type | **Image 073** | **Image 086** | **Image 094** | **Image 410** |
| **Masked**  **Images** |  |  |  |  |
| **MaskGIT**  **Inpainting**  **Results** |  |  |  |  |
| **Ground**  **Truth** |  |  |  |  |

**The setting about training strategy, mask scheduling parameters**

在Discussion中有提到使用了兩個不同的Loss function作為training的strategy比較，此外在訓練時設定的 mask scheduling gamma function都是以cosine的方式去生成。在inference的inpainting時有使用三種不同的gamma function (Cosine、Linear、Square)，如下圖所示，根據所有的iteration得到不同的FID結果

一張含有 文字, 圖表, 行, 繪圖 的圖片

自動產生的描述

在iteration增加時表現的FID會越來越差，其中發現linear在前期的mask function表現最好，但隨著ratio的增加，cosine的表現效果是最好的。