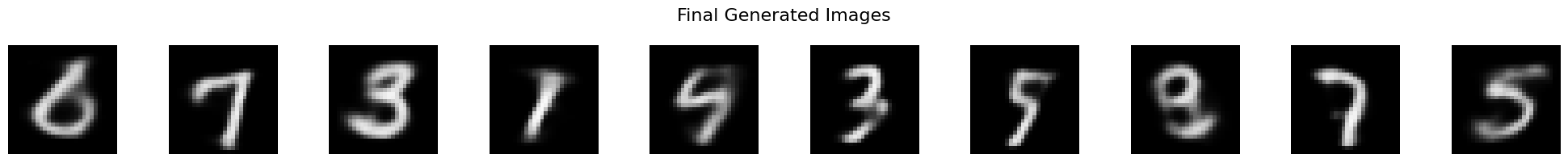
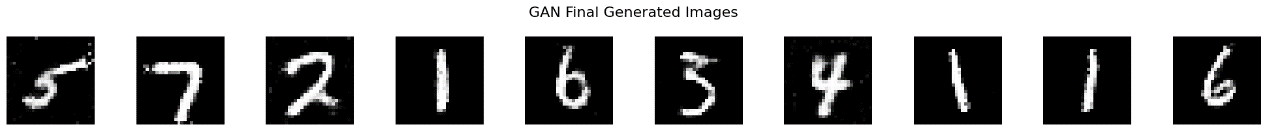
對照圖

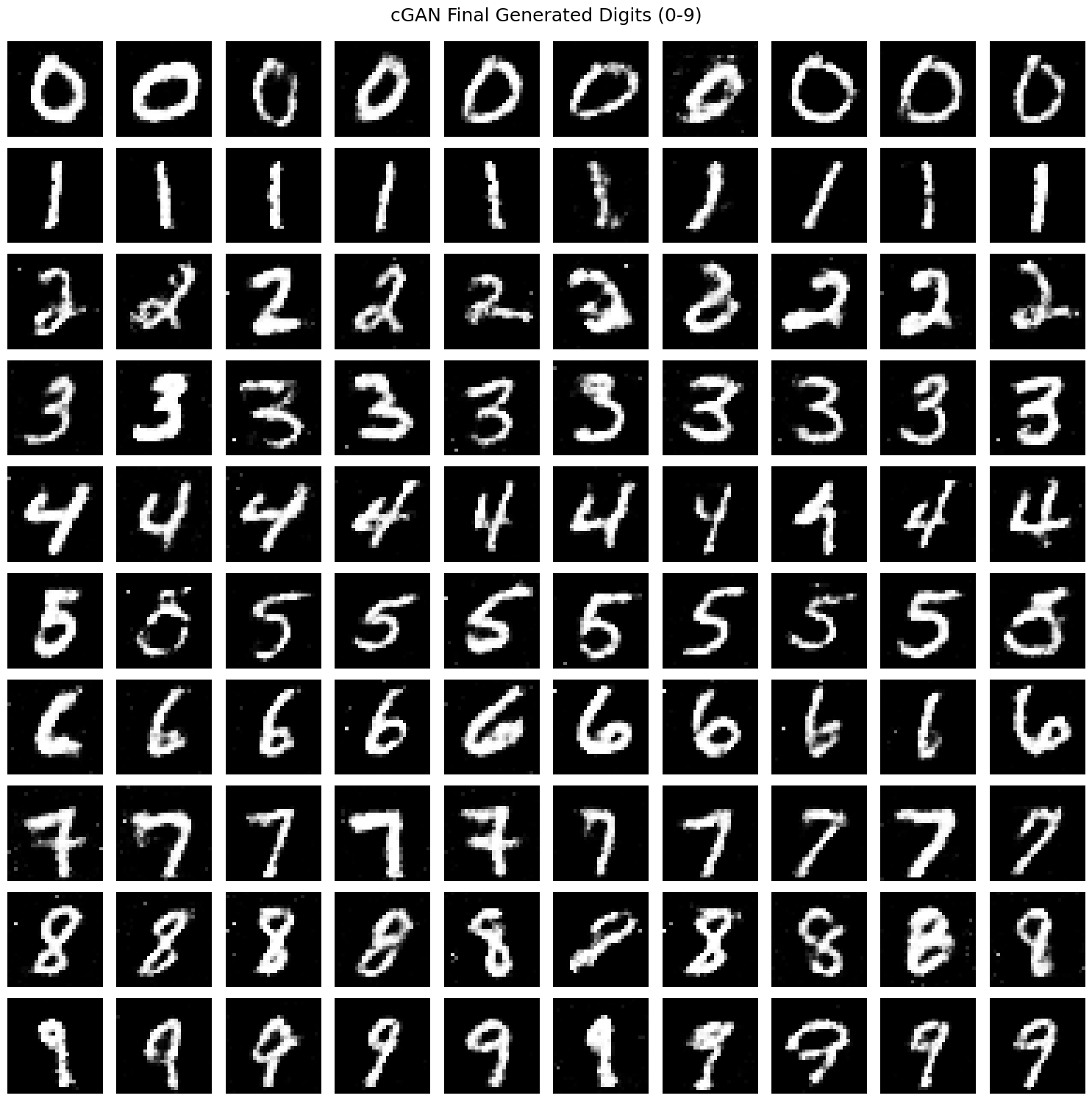
VAE Generated Images:



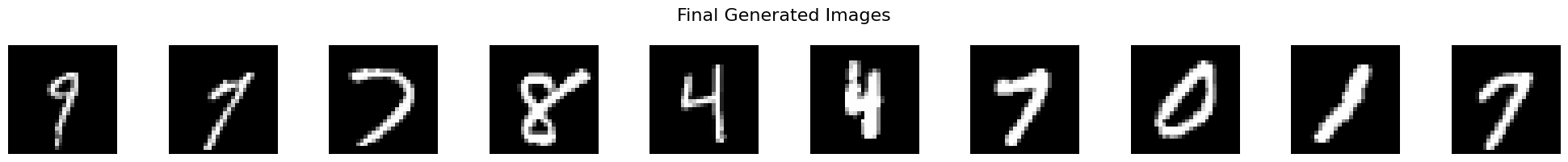
GAN Generated Images:



GAN Generated Images:



GAN Generated Images:



分析四個模型項目

清晰度比較:

**分析：**

* **VAE**：產生較模糊的影像，因為使用 BCE 重建損失和 KL 散度正則化會導致生成分佈過於平滑，傾向產生「平均化」的結果
* **GAN**：產生較清晰的影像，因為判別器會強制生成器產生接近真實資料分佈的樣本，對抗訓練機制鼓勵生成高頻細節
* **cGAN**：清晰度與 GAN 相當，同樣受益於對抗訓練
* **Diffusion**：確實相對清晰，透過逐步去噪過程能捕捉細緻紋理，但手寫數字對於細緻紋理的部分其實並無大量需求，且確實需要大量的訓練和採樣步數（程式碼設定 2000 步），成本考量可能不是最佳解

可控性(是否能指定數字):

**理論分析：**

* **VAE**：**無法控制**。從標準常態分佈採樣，無法指定生成特定數字
* **GAN**：**無法控制**。同樣從隨機噪聲生成，無條件資訊
* **cGAN**：**完全可控**。透過標籤嵌入（nn.Embedding），可明確指定生成 0-9 任一數字
* **Diffusion**：**無法控制**（此程式碼實作）。未實作條件機制，但理論上可擴展為 Conditional Diffusion

訓練/推理效率:

**理論分析：**

**訓練效率（從快到慢）：**

1. **GAN/cGAN**：理論上最快，單次前向傳播即可計算損失，但重點在需要平衡 G 和 D 的訓練
2. **VAE**：訓練相對快速，單次前向+後向傳播，結構簡單（全連接層）
3. **Diffusion**：最慢，非常慢，每個 batch 需要隨機採樣時間步 t 並前向傳播 U-Net

**推理效率（從快到慢）：**

1. **VAE**：最快，單次解碼器前向傳播（幾毫秒）
2. **GAN/cGAN**：非常快，單次生成器前向傳播
3. **Diffusion**：**極慢**，需要 2000 步反向擴散過程（程式碼設定），每步都要執行 U-Net

**實際數據（基於程式碼配置）：**

|  |  |  |  |
| --- | --- | --- | --- |
| **模型** | **訓練 Epochs** | **每 Epoch 大約時間(取決於硬體性能)** | **推理時間（10張）** |
| VAE | 50 | ~30秒 | <1秒 |
| GAN | 100 | ~40秒 | <1秒 |
| cGAN | 50 | ~40秒 | <1秒 |
| Diffusion | 200 | ~2-3分鐘 | **數分鐘**（2000步去噪） |

穩定性(是否出現模糊或mode collapse):

**理論分析：**

**VAE：穩定但模糊**

* **訓練穩定**：損失函數明確（BCE + KLD），不會崩潰
* **固有模糊性**：KL 散度正則化迫使潛在空間接近標準常態分佈，犧牲重建清晰度
* 程式碼改進：使用 β-annealing (0.5→4.0) 和 Free Bits (0.5) 減輕所謂的後驗坍縮

**GAN：訓練途中確實發生過 Mode Collapse 但有成功處理掉**

* **Mode Collapse 風險高**：理論上生成器可能只學會生成少數幾種數字（如只生成 1 和 7）
* **訓練不穩定**：G 和 D 的對抗平衡難以維持，可能出現震盪
* 我的程式碼中的緩解措施：
  + Discriminator 學習率設為 Generator 的 2 倍（D\_lr = 2 \* G\_lr）
  + 使用 Dropout (0.3) 和 LeakyReLU 增強穩定性

**cGAN：比較穩定**

* **相對穩定**：條件資訊提供額外監督信號，減少 Mode Collapse
* **Label Smoothing**：程式碼對真實樣本使用 0.85 而非 1.0（valid = 0.85），減少判別器過度自信

**Diffusion：最穩定但訓練困難**

* **最穩定**：MSE 損失明確，無對抗訓練，無 mode collapse 問題
* **需要長時間訓練**：200 epochs 可能不足以收斂，理論上可能需要 500+ epochs
* **Clipped Reverse Diffusion**：程式碼實作 x\_0\_pred.clamp\_(-1., 1.) 防止數值爆炸

**程式碼中的穩定性機制總結：**

|  |  |  |
| --- | --- | --- |
| **模型** | **穩定性技術** | **程式碼位置** |
| VAE | β-annealing, Free Bits, Dropout | improved\_vae\_loss(), beta\_start=0.5 |
| GAN | Discriminator 學習率加倍, Dropout | D\_lr = 2 \* G\_lr |
| cGAN | Label Smoothing, 條件資訊 | valid = 0.85 |
| Diffusion | EMA, Cosine Schedule, Clipping | ExponentialMovingAverage, clamp\_(-1., 1.) |

其他自行加入之方法:

**VAE 部分**

**1. β-VAE with Annealing Strategy**

python

beta\_start = 0.5

beta\_end = 4.0

warmup\_epochs = 20

* **說明**：逐步增加 KL 散度權重（0.5→4.0），前期專注重建品質，後期強化潛在空間結構
* **效果**：避免後驗坍縮（posterior collapse），改善生成品質

**2. Free Bits 技術**

python

free\_bits = 0.5

KLD\_per\_dim = torch.clamp(KLD\_per\_dim, min=free\_bits)

* **說明**：確保每個潛在維度的 KL 散度至少有最小值，防止某些維度被忽略
* **效果**：提升潛在空間利用率

**3. Logvar Clipping**

python

logvar = torch.clamp(logvar, -10, 10)

* **說明**：限制對數方差範圍，防止數值不穩定
* **效果**：避免梯度爆炸或消失

**4. Learning Rate Scheduler**

python

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5)

* **說明**：當驗證損失停止改善時自動降低學習率
* **效果**：更精細的收斂

**5. Dropout 正則化**

python

self.dropout = nn.Dropout(0.2)

* **說明**：在編碼器/解碼器中加入 Dropout 層
* **效果**：防止過擬合，提升泛化能力

**GAN 部分**

**1. 不對稱學習率（Discriminator 加速）**

python

G\_lr = 2e-4

D\_lr = 2 \* G\_lr *# Discriminator 學習率為 Generator 兩倍*

* **說明**：讓判別器學習更快，避免生成器過早壓制判別器
* **效果**：平衡對抗訓練，減少訓練不穩定

**2. BatchNorm 層**

python

nn.BatchNorm1d(hidden\_dim \* 2)

* **說明**：在 Generator 中使用批次正規化
* **效果**：加速訓練、穩定梯度流

**3. LeakyReLU 激活函數**

python

nn.LeakyReLU(0.2)

* **說明**：使用 LeakyReLU 代替 ReLU，負斜率 0.2
* **效果**：避免神經元死亡問題

**4. Tanh 輸出激活**

python

nn.Tanh() *# 輸出範圍 [-1, 1]*

* **說明**：配合資料歸一化到 [-1, 1]
* **效果**：改善訓練穩定性（標準 GAN 實踐）

**cGAN 部分**

**1. Label Smoothing**

python

valid = torch.ones(batch\_size, 1).to(device) \* 0.85 *# 真實標籤使用 0.85 而非 1.0*

* **說明**：對真實樣本使用軟標籤（0.85），減少判別器過度自信
* **效果**：防止訓練早期判別器過強，改善生成器學習

**2. Embedding 層取代 One-hot**

python

self.label\_emb = nn.Embedding(num\_classes, num\_classes)

* **說明**：使用可學習的嵌入層而非固定的 one-hot 編碼
* **效果**：更高效的記憶體使用、更好的類別表示學習

**3. 不同的 G/D 學習率**

python

G\_lr\_cgan = 2e-4

D\_lr\_cgan = 5e-5 *# Discriminator 學習率較低*

* **說明**：與 GAN 相反，這裡降低判別器學習率
* **效果**：適應條件資訊帶來的額外監督信號

**Diffusion Model 部分**

**1. Exponential Moving Average (EMA)**

python

model\_ema = ExponentialMovingAverage(model, device=device, decay=1.0 - alpha)

* **說明**：維護模型參數的指數移動平均版本用於推理
* **效果**：顯著提升生成品質和穩定性

**2. Cosine Variance Schedule**

python

def \_cosine\_variance\_schedule(self, timesteps, epsilon=0.008):

f\_t = torch.cos(((steps / timesteps + epsilon) / (1.0 + epsilon)) \* math.pi \* 0.5) \*\* 2

* **說明**：使用 Cosine schedule 代替線性 schedule
* **效果**：更平滑的噪聲添加過程，改善生成品質

**3. Clipped Reverse Diffusion**

python

x\_0\_pred = torch.sqrt(1. / alpha\_t\_cumprod) \* x\_t - ...

x\_0\_pred.clamp\_(-1., 1.) *# 裁剪預測值*

* **說明**：在反向擴散過程中裁剪預測的 x₀
* **效果**：防止數值爆炸，提升採樣穩定性

**4. OneCycleLR Scheduler**

python

scheduler = OneCycleLR(optimizer, lr, total\_steps=..., pct\_start=0.25, anneal\_strategy='cos')

* **說明**：使用單週期學習率策略，先增後減
* **效果**：加速收斂、提升最終性能

**5. ShuffleNet V2 輕量化架構**

python

class ResidualBottleneck(nn.Module): *# ShuffleNet V2 基本單元*

self.channel\_shuffle = ChannelShuffle(2)

* **說明**：使用 ShuffleNet V2 組件構建 U-Net，而非標準卷積
* **效果**：大幅減少參數量和計算量（適合資源受限環境）

**6. 高採樣步數**

python

timesteps = 2000 *# 擴散步數*

* **說明**：使用 2000 步而非常見的 1000 步
* **效果**：更精細的去噪過程，理論上可達更高品質

**總結表格**

|  |  |  |
| --- | --- | --- |
| **模型** | **額外技術數量** | **核心亮點** |
| **VAE** | 5 項 | β-annealing + Free Bits |
| **GAN** | 4 項 | 不對稱學習率 |
| **cGAN** | 3 項 | Label Smoothing + Embedding |
| **Diffusion** | 6 項 | EMA + Cosine Schedule + 輕量化架構 |

檢討:  
1. 若時間足夠Fine tuning 可以做得更徹底，四種模型都可能可以有更好的結果

2. 調整被作業訂為固定的那些方法或參數，可能可以有更好的結果

3. 不限制只能用colab，可能可以有更好的硬體資源輔助運算，提升效率