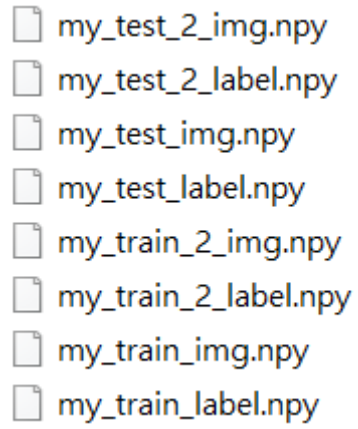
Data preprocess:

這次功課的前處理我先將 binary 檔裡的資料使用 data_prerocess.py, data_prerocess2.py，將 img, label 轉成 npy 檔。因為我發現如果直接在 dataloader 裡取用 binary 檔的資料會使 cpu 在檔案讀取方面拖太久的時間，下圖是兩者的比較(上未轉 npy，下轉 npy)可以發現兩者在讀取 64*16 筆資料時時間相差至 13 倍。

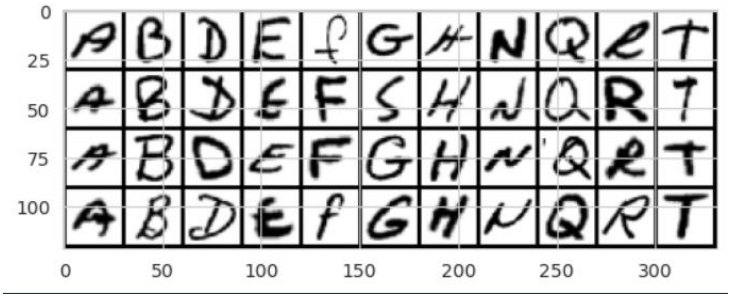
```
print(len(train_laoder))
start = time.time()
for i, data in enumerate(train_laoder):
    a = 1
    #print(data.shape)
    #imshow(torchvision.utils.make_grid(data, padding=3))
print(time.time() - start)

16
1.3291065692901611

print(len(dataloader))
start = time.time()
for i, data in enumerate(dataloader):
    # print(data.shape)
    a = 1
    # imshow(torchvision.utils.make_grid(data, padding=3))
print(time.time() - start)

16
0.10868287086486816
```

兩種 dataloader 比較圖



Bonus dataset 是有_2 的檔案

65 classes dataset



Bonus 11 classes dataset

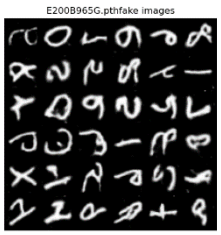
Generate model choise:

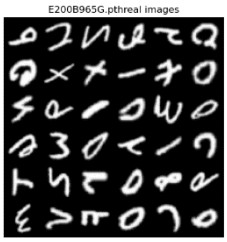
在 generate model 方面我選擇的 3 個 GAN 分別為:

1. DCGAN (助教給的範例)
2. CycleGAN (聽說 cycleGAN 就能生產出很好的結果)
3. Conditional Diffusion model (另外找到的輕量化 diffusion model)

1. DCGAN

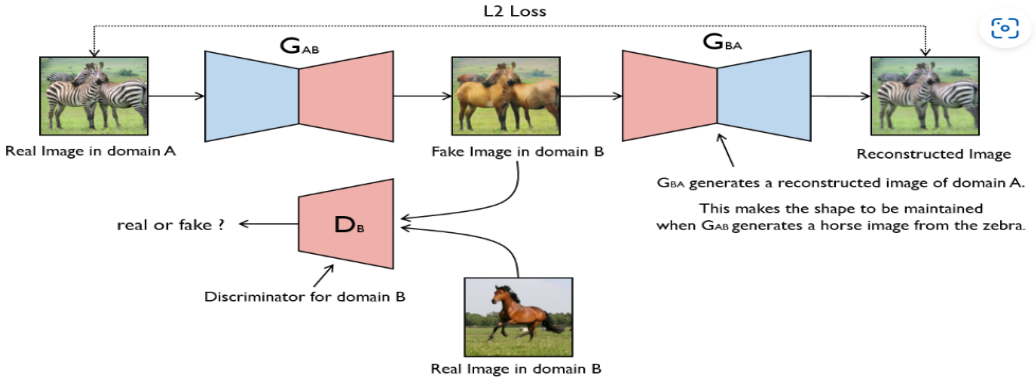
dcGAN 的話就是直接沿用助教的設定下去訓練，結果如下二圖。





2. CycleGAN

因為 cycleGAN 都是 IMG2IMG，風格轉換上的任務，像是馬轉斑馬，



通常需要一張 real A 圖，目標是轉成 real B 圖，這樣的前提下我以兩種想法下去做訓練:

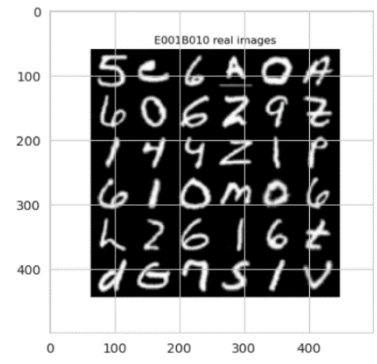
I. Real A: a random sample imgs, Real B: shuffle Real A imgs

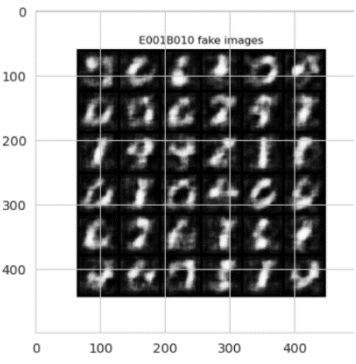
甲、Train 2 個 epoch 就已經有很好的結果了，結果比較傾向 autoencoder

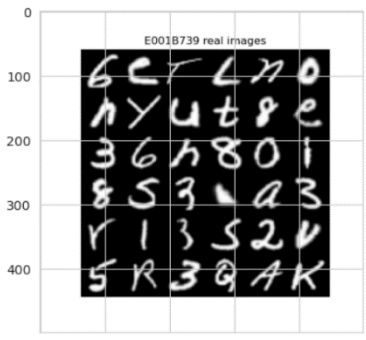
II. Real A: a noise sample, Real B: a random sample imgs

甲、這個模式 train 比較慢，大概 50 個 epoch 可以有不錯的結果

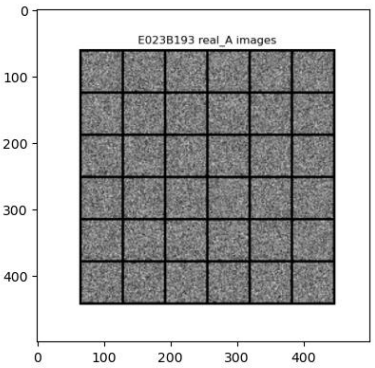
I

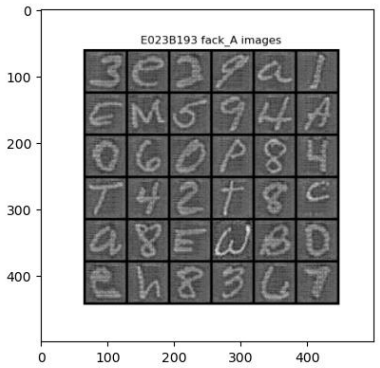


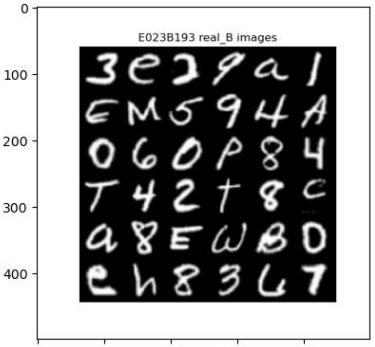


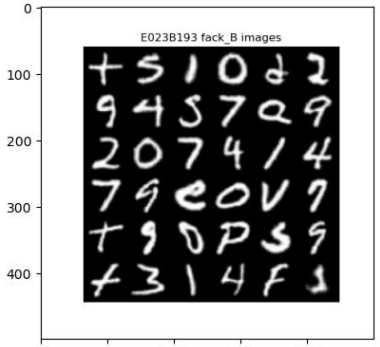


II









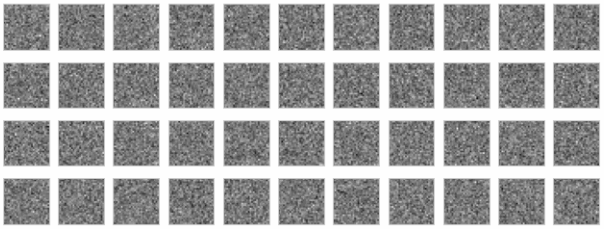
3. Conditional Diffusion model

這是一個加了 conditional 的 stable diffusion model，其中 n_T 代表 diffusion 總共的步數，classes 為 Bonus 的 dataset 數量，在訓練時我們 input 的 label 會先經過 enbeding 再跟圖片一起做訓練。 右圖是 difussion model schedule 的 implement。最後我們也是照著這個 schedule 去跟 generate 出來的圖片去做 L2 loss。

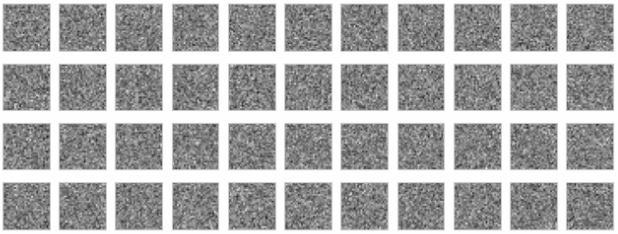
```
self.n_epochs = 100
self.batch_size = 1024
self.n_T = 400
self.n_classes = 11
self.n_feat = 128
self.lr = 0.0002
self.save_model = True
self.ws_test = [0.0, 0.5, 2.0]
```

```
def ddpm_schedules(beta1, beta2, T):
    """
    Returns pre-computed schedules for DDPM sampling, training process.
    """
    assert beta1 < beta2 < 1.0, "beta1 and beta2 must be in (0, 1)"

    beta_t = (beta2 - beta1) * torch.arange(0, T + 1, dtype=torch.float32) / T + beta1
    sqrt_beta_t = torch.sqrt(beta_t)
    alpha_t = 1 - beta_t
    log_alpha_t = torch.log(alpha_t)
    alphabar_t = torch.cumsum(log_alpha_t, dim=0).exp()

    sqrtab = torch.sqrt(alphabar_t)
    oneover_sqrta = 1 / torch.sqrt(alpha_t)

    sqrtmab = torch.sqrt(1 - alphabar_t)
    mab_over_sqrtmab_inv = (1 - alpha_t) / sqrtmab
```

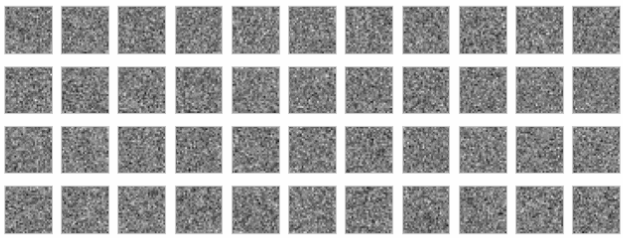
從 W_0~W_2 代表的是 context mask 的程度多寡(conditional 的部分)，W_0 在 inference 的時候完全 mask context，W 越大代表給較多 context 的資訊。產生的圖也就越好。



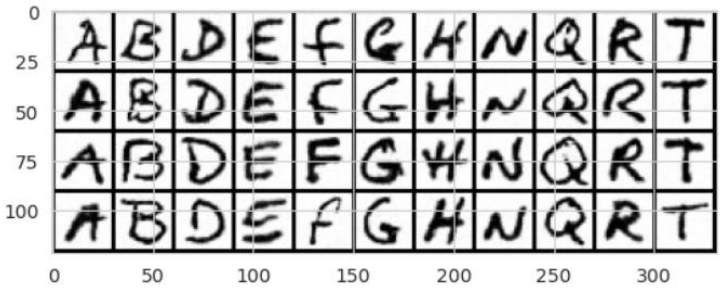
W_0



W_1



W_2



實際上在 inference 的時候我們給定 label，跟一個 nomal nosie，model 負責幫我們生產出跟 label 相關的圖片 。