CSC413: Programming Assignment 4

Tianyu Du (1003801647)

2020/03/26 at 18:58:33

1 Part 1: Deep Convolutional GAN (DCGAN) [4pt]

1.1 Generator

Implementation

```
1
```

1.2 Training Loop

Implementation

```
1
```

1.3 Experiments

2 Part 2: CycleGAN [3pt]

2.1 CycleGAN Experiments

2.1.1 Question 1

Answer Default configurations at 200 iterations:

Figure 2.1: Iteration 200, left: $X \rightarrow Y$, right: $Y \rightarrow X$

Figure 2.2: Iteration 5000, left: $X \to Y$, right: $Y \to X$

2.1.2 Question 2

Answer Changed the random seed to `SEED = 42` (originally 4). With the new random seed, some images in the final $Y \to X$ translations are more clear and with less background noise. Such as boot (row 5, item 4), mountain (row 3, item 5), and donut (row 2, item 5).

Explain differences

Figure 2.3: Iteration 200, left: $X \to Y$, right: $Y \to X$

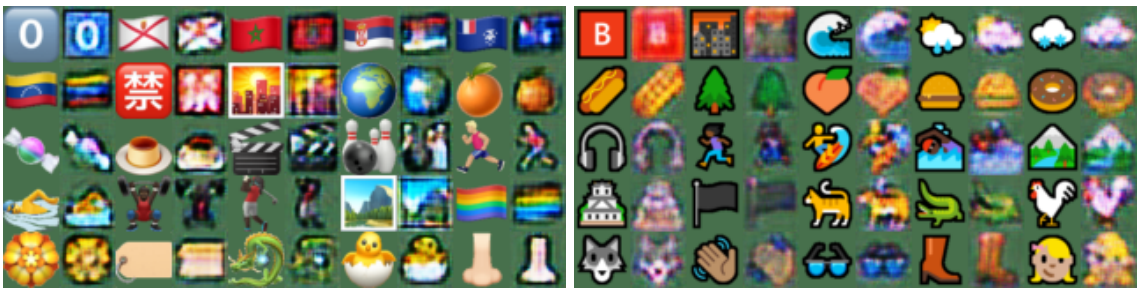


Figure 2.4: Iteration 5000, left: $X \to Y$, right: $Y \to X$

2.1.3 Question 3

Add comment

Comments I tried three different level of lambda values for the cycle consistency, translation results are attached below:

Configuration 1 `lambda_cycle = 0`, all other hyper-parameters are default, including the random seed.

Figure 2.5: `lambda_cycle = 0`, iteration 200, left: $X \to Y$, right: $Y \to X$

Figure 2.6: `lambda_cycle = 0`, iteration 5000, left: $X \to Y$, right: $Y \to X$

Configuration 2 `lambda_cycle = 0.03`, all other hyper-parameters are default, including the random seed.

Figure 2.7: `lambda_cycle = 0.03`, iteration 200, left: $X \to Y$, right: $Y \to X$

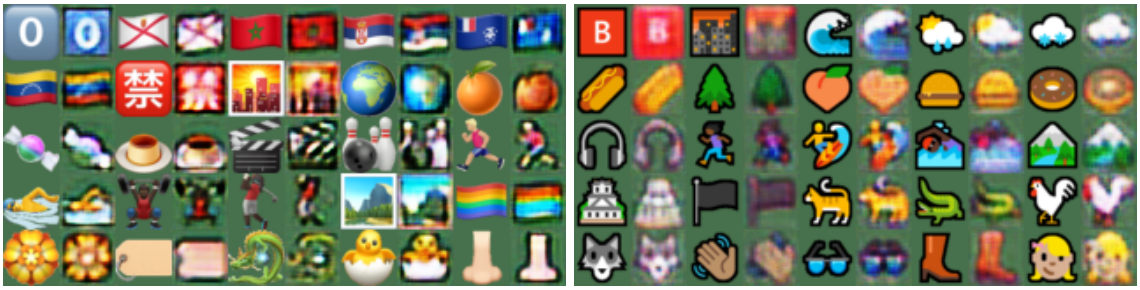
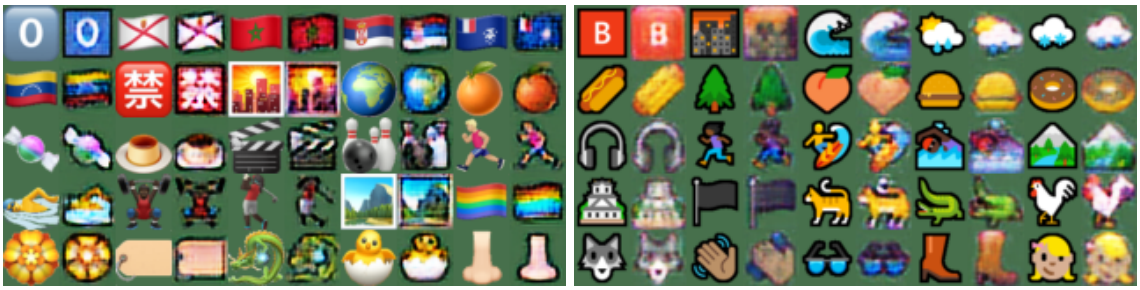


Figure 2.8: `lambda_cycle = 0.03`, iteration 5000, left: $X \to Y$, right: $Y \to X$



Configuration 3 `lambda_cycle = 0.3`, all other hyper-parameters are default, including the random seed.

Figure 2.9: `lambda_cycle = 0.3`, iteration 200, left: $X \to Y$, right: $Y \to X$

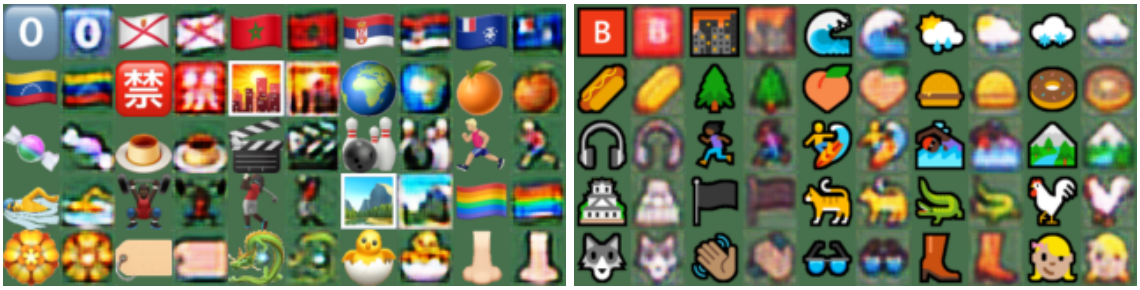
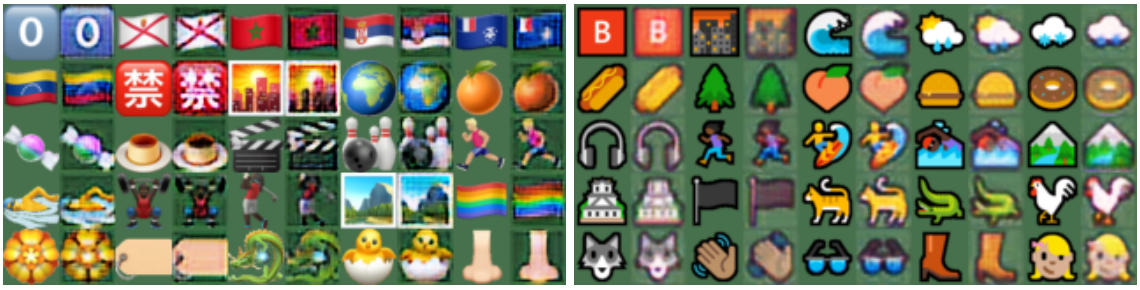
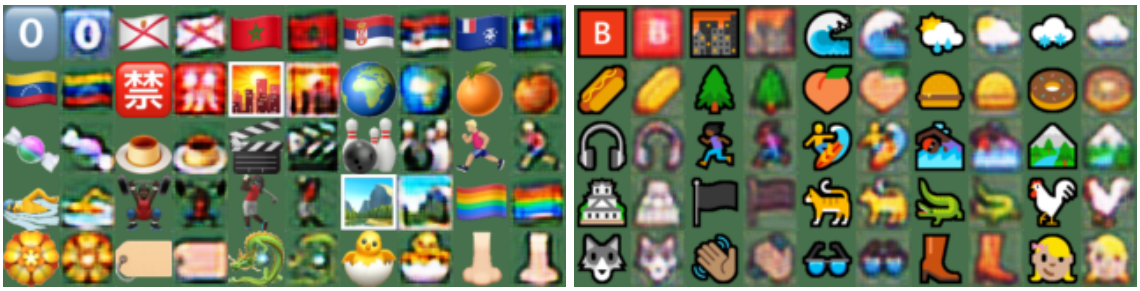


Figure 2.10: `lambda_cycle = 0.3`, iteration 5000, left: $X \to Y$, right: $Y \to X$



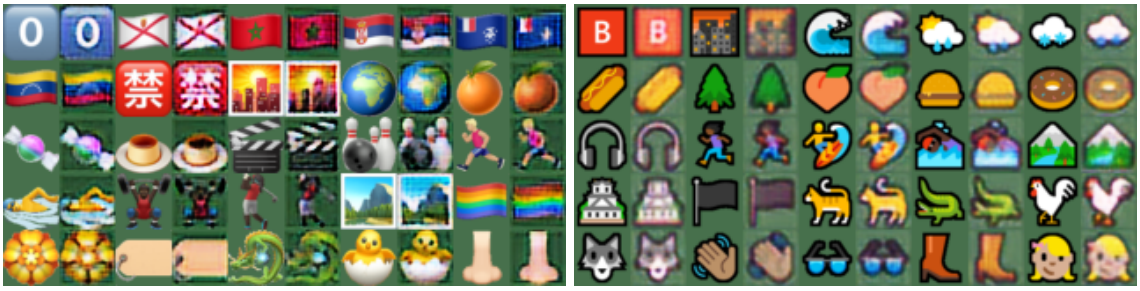
Configuration 4 `lambda_cycle = 1.0`, all other hyper-parameters are default, including the random seed.

Figure 2.11: `lambda_cycle = 1.0`, iteration 200, left: $X \to Y$, right: $Y \to X$



t

Figure 2.12: `lambda_cycle = 1.0`, iteration 5000, left: $X \to Y$, right: $Y \to X$



3 Part 3: BigGAN [2pt]

3.1 BigGAN Experiments

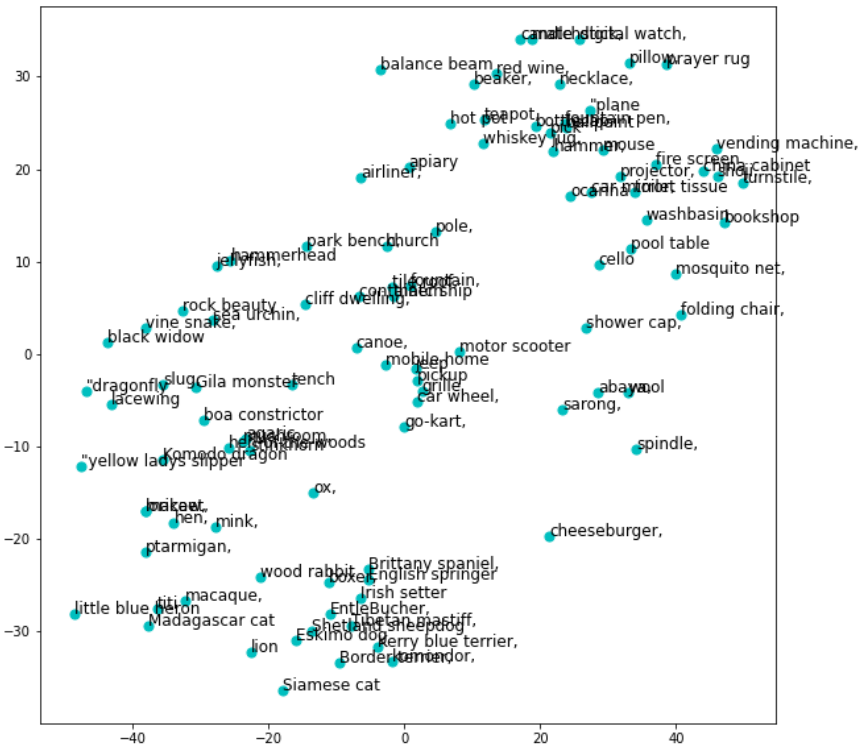
3.1.1 Question 1

Method Given two classes with embeddings $\mathbf{r}_{class1}$ and $\mathbf{r}_{class2}$ in T-SNE space, to decide whether they are good candidate for linear interpolation, I looked at convex combinations of two classes' embeddings in figure 3.1. Let Φ denote the set of convex combinations:

$$\Phi = \{\alpha \mathbf{r}_{class1} + (1-\alpha)\mathbf{r}_{class2} : \alpha \in [0,1]\} \tag{3.1}$$

Linear interpolations between these two classes are basically taking points in set Φ and visualizing them. If there are many other classes' embeddings on or near the set Φ, points in Φ are likely to be associated with meaningful visualizations. Therefore, linear interpolations between these two classes are likely to generate meaningful results. Otherwise, these two classes are not good candidates for linear interpolation.

Figure 3.1: T-SNE Plot



Good Candidate Match 1 There are many classes near the segment between `folding chair` and `vending machine` (right-up corner), so linear interpolation should be meaningful.

Good Candidate Match 2 Similarly, there are many classes on the line between `bookshop` and `hot pot` (right-up corner), so they are ideal for linear interpolation.

Bad Candidate Match 1 There are no other classes' embeddings are in between embeddings of `Go-Kart` and `Chessburger` in T-SNE plot, so they are not good match.

Bad Candidate Pair 2 Similarly, `Ox` and `Chessburger` are not good match.

3.1.2 Question 2

Implementation

```
#Linear interpolation between two class embedings
def generate_linear_interpolate_sample(G, batch_size, class_label1, class_label2, alpha):
    G.eval()
    G.to(DEVICE)
    with torch.no_grad():
        z = torch.randn(batch_size, G.dim_z).to(DEVICE)
        class1_emb = G.shared(torch.tensor(class_label1).to(DEVICE)*torch.ones((batch_size,)
    ).to(DEVICE).long())
        class2_emb = G.shared(torch.tensor(class_label2).to(DEVICE)*torch.ones((batch_size,)
    ).to(DEVICE).long())

        ########################################
        ##   FILL THIS IN: CREATE NEW EMBEDDING  ##
        ########################################
        new_emb = alpha * class1_emb + (1 - alpha) * class2_emb

        images = G(z, new_emb)
    return images

```

Linear Interpolation Results

Figure 3.2: Good Match 1: Folding Chair and Vending Machine

Figure 3.3: Good Match 2 Hot Pot and Bookshop

Figure 3.4: Bad Match 1 Go-Kart and Chessburger

Figure 3.5: Bad Match 2 Ox and Chessburger