Lab5 - Let's Play GAN

311581005 智能碩一 吳佳豪

1 INTRODUCTION

GAN 主要包含生成器 generator 與判別器 discriminator 兩個部分，如以圖片為例，生成器主要學習真實圖片的分佈後將輸入低維向量轉換為高維向量的真實圖片，而判斷器主要需要對輸入的圖片判別是否為真實的圖片，兩個互相對抗來提升網路能力。

這次作業主要實作多加了 condition 的 CGAN，可以使生成器依照我們想要的條件生成真實圖片。而目標主要為給定多個特定形狀與顏色的 labels 生成相對應的圖片，以下實驗主要是使用 DCGAN 與 ACGAN 架構實作，實驗結果 ACGAN 可以在 test 與 new_test 的 JSON 測試檔中分別達到 **80.56%與 82.14%**的準確度。

2 IMPLEMENTATION DETAILS

2.1 DESCRIBE HOW YOU IMPLEMENT YOUR MODEL, INCLUDING YOUR CHOICE OF CGAN, MODEL ARCHITECTURES, AND LOSS FUNCTIONS.

以下主要為實作 DCGAN 與 ACGAN 架構。

2.1.1 DCGAN

DCGAN 是將 CNN 與 GAN 做結合的深度卷積生成對抗網路，主要生成器使用 deconvolutions 而判別器使用 stride convolutions 替代 pooling layer，並在生成器與判別器都使用 batchnorm，可以有效避免模型不穩定。在生成器中除了輸出層用 tanh 激活函數，其餘層都使用 ReLU，而在判別器中所有層都使用 LeakyReLU 激活函數。因為主要判斷圖片是真還是為假，只有兩種可能性，所以選擇 Binary Cross Entropy 當此架構的 loss function。

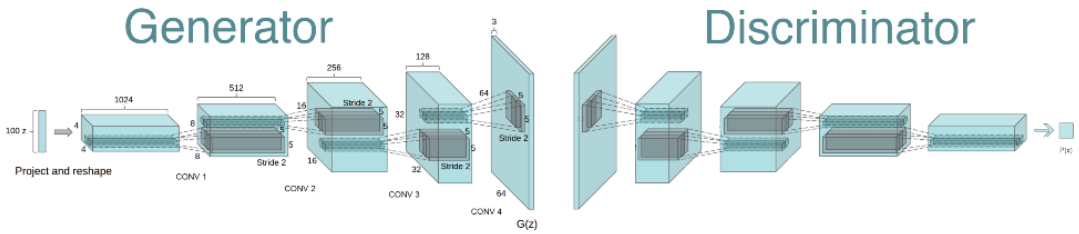

Figure 1. DCGAN conception

```python
class dc_netG(nn.Module):
    def __init__(self, nz):
        super(dc_netG, self).__init__()
        self.nz = nz

        # Transposed Convolution 1
        self.tconv1 = nn.Sequential(
            nn.ConvTranspose2d(self.nz, 512, 4, 1, 0, bias=Fals
            nn.BatchNorm2d(512),
            nn.ReLU(True),
        )
        # Transposed Convolution 2
        self.tconv2 = nn.Sequential(
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
        )
        # Transposed Convolution 3
        self.tconv3 = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
        )
        # Transposed Convolution 4
        self.tconv4 = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
        )

        # Transposed Convolution 5
        self.tconv5 = nn.Sequential(
            nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),
            nn.Tanh(),
        )

    def forward(self, input):
        input = input.view(-1, self.nz, 1, 1)
        tconv1 = self.tconv1(input)
        tconv2 = self.tconv2(tconv1)
        tconv3 = self.tconv3(tconv2)
        tconv4 = self.tconv4(tconv3)
        tconv5 = self.tconv5(tconv4)
        output = tconv5
        return output
```

```python
class dc_netD(nn.Module):
    def __init__(self):
        super(dc_netD, self).__init__()

        self.addConditionInfo=nn.Sequential(
            nn.Linear(24, 64 * 64),
            nn.LeakyReLU(0.2, inplace=True)
        )
        # Convolution 1
        self.conv1 = nn.Sequential(
            nn.Conv2d(3 + 1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
        )
        # Convolution 2
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
        )
        # Convolution 3
        self.conv3 = nn.Sequential(
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
        )
        # Convolution 4
        self.conv4 = nn.Sequential(
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
        )
        # Convolution 5
        self.conv5 = nn.Conv2d(512, 1, 4, 1, 0, bias=False)

        self.sigmoid = nn.Sigmoid()

    def forward(self, input,c):
        c = self.addConditionInfo(c).view(-1,1,64,64)
        input = torch.cat((input,c),dim=1)
        conv1 = self.conv1(input)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)
        conv4 = self.conv4(conv3)
        conv5 = self.conv5(conv4)
        realfake = self.sigmoid(conv5).view(-1, 1).squeeze(1)
        return realfake
```

Figure 2. DCGAN Implement

2.1.2 ACGAN

ACGAN 主要是在 CGAN 架構上進一步拓展，使用輔助分類器(Auxiliary Classifier)，使得可以使用 condition 生成指定的 class 圖片。在此架構中，因為多增加類別標籤，所以損失函數為判別真偽損失和分類損失兩個部分，因此前面損失為二分類選擇使用 Binary Cross Entropy 而後面為多分類則使用 Cross Entropy 的 loss function。

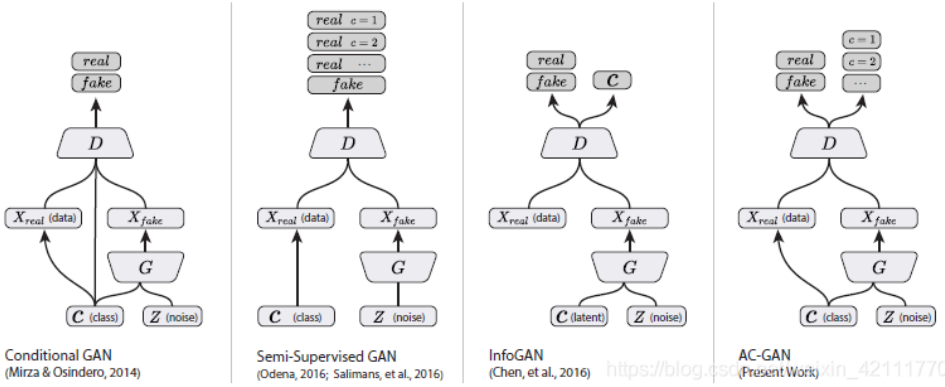

Figure 3. ACGAN conception

```python
class _netG(nn.Module):
    def __init__(self, nz):
        super(_netG, self).__init__()
        self.nz = nz

        # first linear layer
        self.fc1 = nn.Linear(self.nz, 384)
        # Transposed Convolution 2
        self.tconv2 = nn.Sequential(
            nn.ConvTranspose2d(384, 192, 4, 1, 0, bias=False),
            nn.BatchNorm2d(192),
            nn.ReLU(True),
        )
        # Transposed Convolution 3
        self.tconv3 = nn.Sequential(
            nn.ConvTranspose2d(192, 96, 4, 2, 1, bias=False),
            nn.BatchNorm2d(96),
            nn.ReLU(True),
        )
        # Transposed Convolution 4
        self.tconv4 = nn.Sequential(
            nn.ConvTranspose2d(96, 48, 4, 2, 1, bias=False),
            nn.BatchNorm2d(48),
            nn.ReLU(True),
        )
        # Transposed Convolution 5
        self.tconv5 = nn.Sequential(
            nn.ConvTranspose2d(48, 24, 4, 2, 1, bias=False),
            nn.BatchNorm2d(24),
            nn.ReLU(True),
        )

        # Transposed Convolution 6
        self.tconv6 = nn.Sequential(
            nn.ConvTranspose2d(24, 3, 4, 2, 1, bias=False),
            nn.Tanh(),
        )

    def forward(self, input):
        input = input.view(-1, self.nz)
        fc1 = self.fc1(input)
        fc1 = fc1.view(-1, 384, 1, 1)
        tconv2 = self.tconv2(fc1)
        tconv3 = self.tconv3(tconv2)
        tconv4 = self.tconv4(tconv3)
        tconv5 = self.tconv5(tconv4)
        tconv6 = self.tconv6(tconv5)
        output = tconv6
        return output
```

```python
class _netD(nn.Module):
    def __init__(self, num_classes=24):
        super(_netD, self).__init__()
        # Con (variable) conv1: Sequential
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 16, 3, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        # Convolution 2
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 3, 1, 0, bias=False),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        # Convolution 3
        self.conv3 = nn.Sequential(
            nn.Conv2d(32, 64, 3, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        # Convolution 4
        self.conv4 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 1, 0, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        # Convolution 5
        self.conv5 = nn.Sequential(
            nn.Conv2d(128, 256, 3, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        # Convolution 6
        self.conv6 = nn.Sequential(
            nn.Conv2d(256, 512, 3, 1, 0, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
        )
        self.fc_dis = nn.Linear(5*5*512, 1)
        self.fc_aux = nn.Linear(5*5*512, num_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input):
        conv1 = self.conv1(input)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)
        conv4 = self.conv4(conv3)
        conv5 = self.conv5(conv4)
        conv6 = self.conv6(conv5)
        flat6 = conv6.view(-1, 5*5*512) #5*5*512
        fc_dis = self.fc_dis(flat6)
        fc_aux = self.fc_aux(flat6)
        classes = self.sigmoid(fc_aux)
        realfake = self.sigmoid(fc_dis).view(-1, 1).squeeze(1)
        return realfake, classes
```

Figure 4. ACGAN Implement

2.2 SPECIFY THE HYPERPARAMETERS

超參數設定:

Table 1. Model hyperparameter settings

Hyperparameter	DCGAN	ACGAN
Epochs	1000	1000
Batch size	64	64
Learning rate	0.0002	0.0002
Z dim	32	64
Random seed	123	123
Generator loop times	5	5

如果 z_dim 調過於大時，發現因 noise 過多所以在訓練上較難訓練，因此訓練後準確度最高只有到 72%左右。而 Generator loop times 為訓練一次 discriminator 訓練幾次 generator，實驗發現如果 1 比 1 時，因為 discriminator 過於強大所以很容易就會使 generator 訓練失敗產生失真有雜訊的圖片，因此在訓練時增加 generator 訓練次數可以得到有效的改善。

3 RESULTS AND DISCUSSION

3.1 SHOW YOUR RESULTS BASED ON THE TESTING DATA

以下實驗結果依照 Table 1. ACGAN 參數進行設定。在 test 與 new_test 的 JSON 測試檔中分別達到 **80.56% 與 82.14%**的準確度。

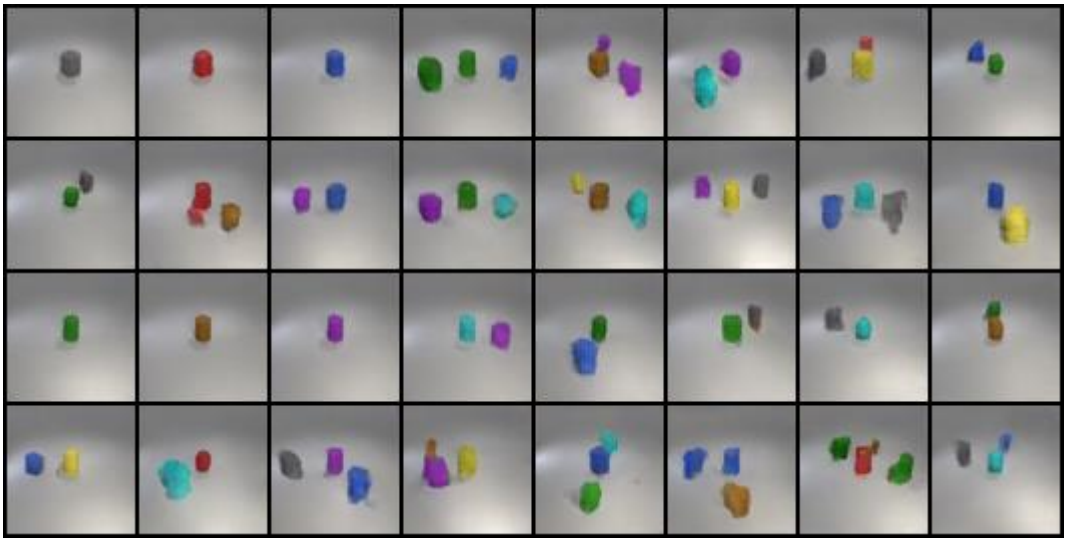

Figure 5. The synthetic images of best accuracy of **0.8056** on test JSON file with ACGAN

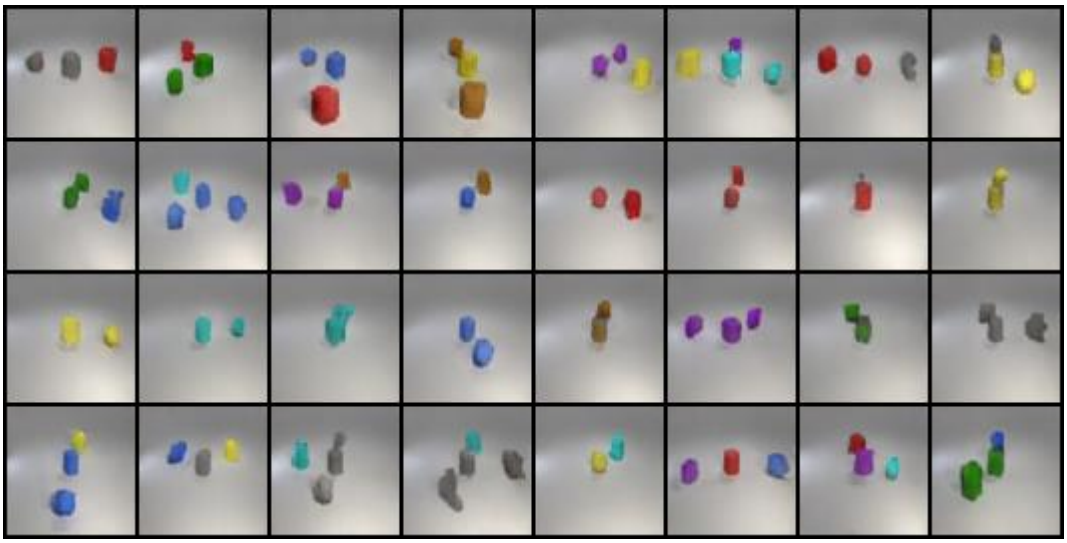

Figure 6. The synthetic images of best accuracy of **0.8214** on new_test JSON file with ACGAN

3.2 DISCUSS THE RESULTS OF DIFFERENT MODELS ARCHITECTURES

一開始選擇使用 DCGAN 架構進行訓練，剛開始訓練時雖然圖片都與真實圖片不會相差太多，但準確度始終都在 20%以下，後來發現因為 discriminator 沒有加入有關類別的 condition 資訊，因此只能產出與真實圖片相差不多的圖片但沒有對應到類別，所以改進成將輸入 24 dim 的 condition 資訊經過一層 Linear layer 變成 1 * h *w 的大小後與原本的 3 * h* w 的圖片 concat，再丟進 convloution layers，發現改進後在 test 與 new_test 測試檔分別可以達到 79.19%與 77.38%的準確度。

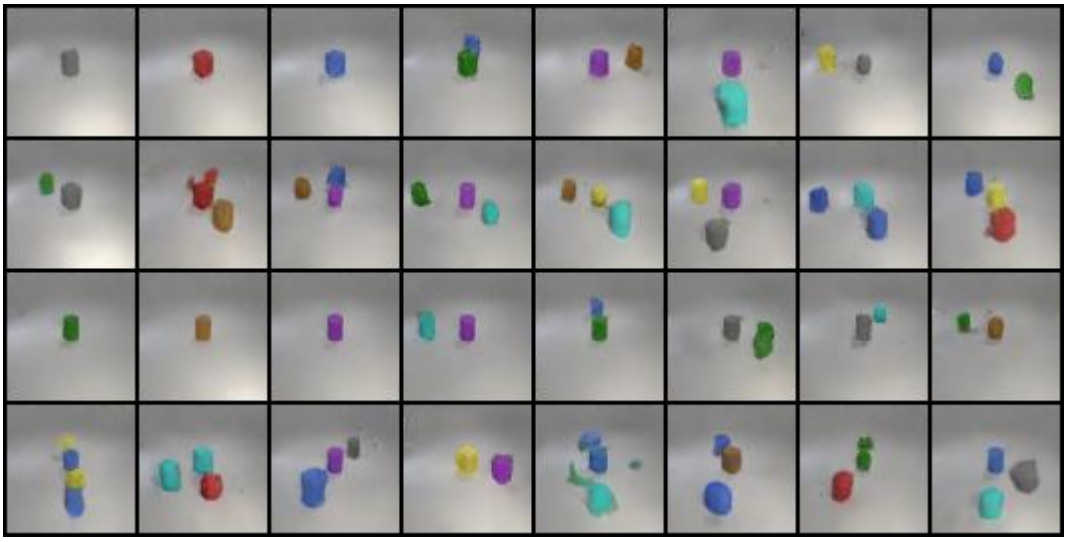

Figure 7. The synthetic images of best accuracy of 0.7917 on test JSON file with DCGAN

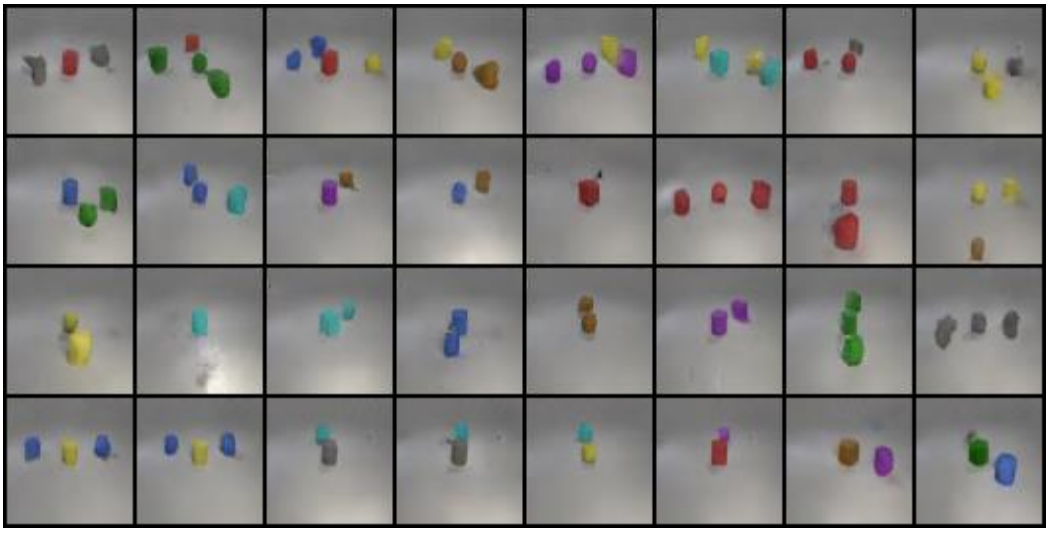

Figure 8. The synthetic images of best accuracy of 0.7738 on new_test JSON file with DCGAN

為了達到 80%以上的準確度，因此後來改使用 ACGAN 來提升分辨類別的能力並測試不同的參數與方法來試著提高精確度，其中有以下幾點：

1. 改變判別真偽損失和分類損失的比例：

   會想要改變的想法是因為目標是需要生成真實且類別正確的圖片，所以測試如果兩個 loss 有不同的權重會不會提升精準度。

   - 判別真偽損失為 0.4 而分類損失為 0.6 的比重，實驗結果為 test 有 77%但 new_test 卻只有 62%的準確度。
   - 判別真偽損失為 0.6 而分類損失為 0.4 的比重，實驗結果為 test 有 68%且 new_test 也只有 70%的準確度。
   - 判別真偽損失和分類損失比重依照 epoch 次數動態變動，實驗結果發現差不多準確度也都在 75%左右。

   經過實驗結果推倒出可能是因為生成真實圖片與生成正確類別都非常重要，因此最後還是設定為各 0.5 的比重。

2. 改變 discriminator 的 kernel size：

   主要想要測試如果 size 大一點看到的範圍大一點，會不會比較提升 discriminator 判別能力，以此提升 generator 的生成能力。因此將 size 由原本 3 * 3 轉變成 5 * 5 的大小，實驗結果準確度最高只到 69%，猜想原因可能是因為 size 較大且圖片解析度也沒有很高，所以對抓取形狀的特徵反而會較弱，所以最後還是改回用 3 * 3 大小的 kernel。

3. 使用 label smoothing :

將 one-hot vector 的 hard label 變成 soft label，因根據 CE-loss 的公式，只會計算為 1 的值其他都會忽略，因此後果可能會使真實標籤與其他標籤的關係被忽略，並且泛化性能變差。但實驗結果發現使用 label smoothing 並 smooth 值設定為 0.1 時，準確度沒有明顯的改變但也沒有變差，因此最後並沒有使用 label smoothing。

4. 其餘超參數變動在標題 2.2中提及。

附: 可以調整的參數

```python
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--model_name', default="DCGan", type=str, help='choice of model name ACGan or DCGan')
    parser.add_argument('--num_classes', default=24, type=int, help='classes number')
    parser.add_argument('--lr', default=0.0002, type=float, help='learning rate')
    parser.add_argument('--weight_decay', default=0, type=float, help='weight decay')
    parser.add_argument('--beta', default=0.5, type=float, help='momentum term for adam')
    parser.add_argument('--batch_size', default=64, type=int, help='batch size')
    parser.add_argument('--log_dir', default='./log/train', help='base directory to save logs')
    parser.add_argument('--model_dir', default='', help='base directory to save logs')
    parser.add_argument('--test_json_name', default='test', help='test json file name')
    parser.add_argument('--dataset_root', default='./dataset', help='root directory for data')
    parser.add_argument('--generator_iter_loop',type=int, default=5, help='every iter generator train times')
    parser.add_argument('--label_smoothing', default=0, type=float, help='label smoothing ratio')
    parser.add_argument('--realfake_loss_ratio', default=0.5, type=float, help='realfake loss ratio(p)')
    parser.add_argument('--classes_loss_ratio', default=0.5, type=float, help='classes loss ratio')
    parser.add_argument('--realfake_loss_decay', default=0, type=float, help='realfake loss decay and classes loss increasing')
    parser.add_argument('--realfake_start_epoch', default=0, type=float, help='realfake loss  over start epochs')
    parser.add_argument('--epochs', type=int, default=1000, help='epoch size')
    parser.add_argument('--seed', default=123, type=int, help='manual seed')
    parser.add_argument('--z_dim', type=int, default=64, help='size of the latent z vector')
    parser.add_argument('--num_workers', type=int, default=4, help='number of data loading threads')
    parser.add_argument('--cuda', default=True, action='store_true')
    parser.add_argument('--cuda_num', type=int, default=0)

    args = parser.parse_args()
    return args
```

Figure 9. All hyperparameters