|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198351-810198577** |
| تاریخ ارسال گزارش | **1402.09.17** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین ششم** | | |

**فهرست**

[**پاسخ 1** –  **Control VAE** 3](#_Toc156318905)

[۱-۱.مقدمه 3](#_Toc156318906)

[۱-۱-۱.دیتاست 5](#_Toc156318907)

[۲-۱. پیاده سازی VAE 5](#_Toc156318908)

[۲-۱-۱. افزودن کتابخانه ها 6](#_Toc156318909)

[۲-۱-۲ .بارگزاری دادگان 7](#_Toc156318910)

[۲-۱-۳. ساخت دیتالودر 7](#_Toc156318911)

[۲-۱-۴. پیاده سازی مدل Beta VAE 8](#_Toc156318912)

[۲-۱-۵. آموزش مدل Beta VAE 9](#_Toc156318913)

[۳-۱. ارزیابی مدل 11](#_Toc156318914)

[۴-۱. پیاده سازی Control VAE 13](#_Toc156318915)

[**پاسخ** **۲**  - 18](#_Toc156318916)

**شکل‌ها**

شکل 1. عنوان تصویر نمونه 4

# **پاسخ 1** – **Control VAE**

## ۱-۱.مقدمه

مقاله:

[ControlVAE: Controllable Variational Autoencoder](https://arxiv.org/abs/2004.05988)

در مورد ایده کلی این مقاله توضیح دهید که چگونه این مقاله می تواند در نهایت به مدل VAE برسد که هم کیفیت عکس های تولیدی و هم تنوع آن ها مناسب باشد؟

VAE از دو بخش اصلی به اسم های log likelihood و KL divergence تشکیل شده است. اولین اسم تلاش میکند که input data را پیاده سازی کند در حالی که KL divergence باغث هم کیفیت خروجی ساخته شده می‌شود و هم باعث diversity ساخته ها می‌شود.

برای جبران KL vanishing مدل Beta VAE عرضه شده که در این تمرین هم استفاده شده است. در مقاله دو چالش با تغییر اندازه این Beta در مدل های پیشین مطرح شده که دقیقا این مقدار چگونه تعریف بشود. که چالش اصلی در مدل های مطرح شده مربوط tune کردن وزن در KL-divergence هست. به همین دلیل در این مقاله از مدل control feedback استفاده می‌شود.

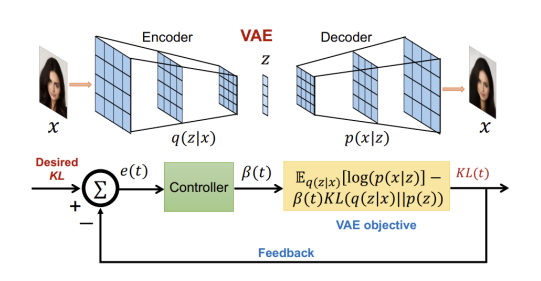
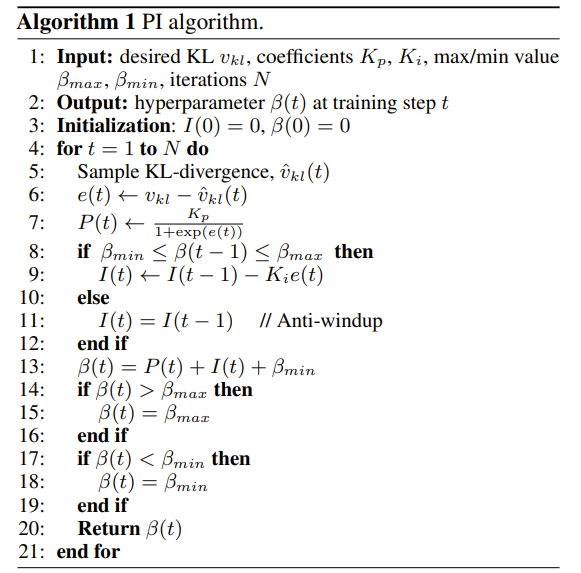


Figure 1شکل 1- مدل control VAE

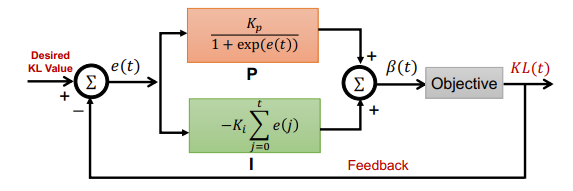
به این صورت که در مدل Control VAE مدل VAE با Controller ترکیب می‌شود تا Stabilize KL divergence رخ بدهد به وسیله مقدار دهی اتوماتیک وزن پارامتر بتا.

بر این اساس طبق الگوریتم شماره 1 مقاله:



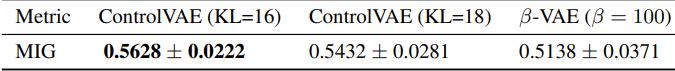
‏0—1الگوریتم شماره 1 مقاله

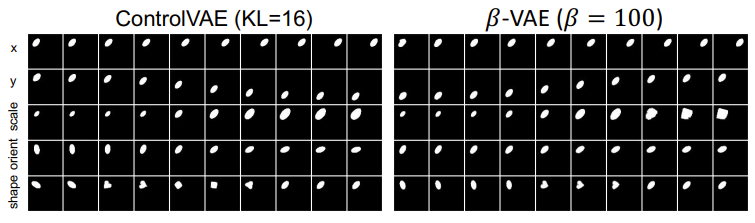
کنترلر که با نام PI شناخته می‌شود مطابق شکل زیردر step t مقدار Beta را محاسبه و Tune کرده و Feedback می‌دهد.



‏0—2شکل 3. شمای تغییر پارامتر های عرضه شده در الگوریتم در تغییر بتا

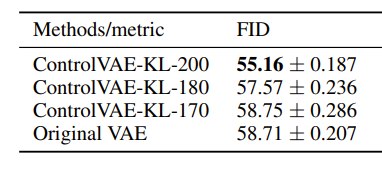
همچنین مقاله بر روی سه دیتاست خروجی گرفته است ولی در اینجا ما از یک دیتاست استفاده میکنیم. جدول و شکل زیر نتایج تفاوت متریک و تصویر ساخته شده با کمک Control VAE را نشان میدهد.





‏0—3شکل 4. نمایش خروجی تفاوت مدل بتا و خروجی تصاویر مدل کنترلر

همچنین این تفاوت برای دو مدل بر اساس شاخص FID در جدول زیر اورده شده است



‏0—4شکل 5. شاخص FID مقاله

### ۱-۱-۱.دیتاست

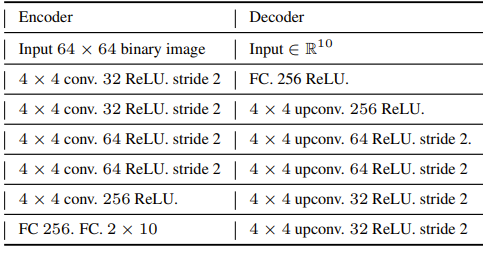
در این تمرین به منظور دیتاست از بخشی از دیتاست [dSprites](https://github.com/google-deepmind/dsprites-dataset) که شامل تصاویر تک کاناله دو بعدی به ابعاد 64\*64 هستند استفاده شده است.



## ۲-۱. پیاده سازی VAE

جدول 1. جدول 1مشخصات پیاده سازی

دقت شود در این مرحله صرفا مدل VAE ساده ساخته نمی‌شود و مدل Beta VAE پیاده سازی می‌شود. معماری مدل بر اساس جدول 6 مقاله اورده شده است:



‏0—5 شکل 6. معماری مدل Beta VAE

### ۲-۱-۱. افزودن کتابخانه ها

!pip install pytorch-fid

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

import os

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

import shutil

from google.colab import drive

from google.colab import files

import requests

import zipfile

import io

from transformers import AutoTokenizer

from transformers import TFAutoModelForSequenceClassification

import tensorflow as tf

import torch

import torch.nn as nn

import torch.nn.functional as F

from sklearn.metrics import confusion\_matrix, classification\_report, f1\_score, recall\_score, accuracy\_score

from sklearn.metrics import accuracy\_score, f1\_score, confusion\_matrix

from transformers.modeling\_outputs import BaseModelOutputWithPastAndCrossAttentions

import torch.optim as optim

import torch

from tensorflow.keras import Sequential, Model

from tensorflow.keras.layers import Input, Dense, Conv2D, Conv2DTranspose, Lambda, Reshape, Flatten, LeakyReLU, Softmax

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import Progbar

from tensorflow.keras import backend as K

import matplotlib.pyplot as plt

import cv2

import random

import click

### ۲-۱-۲ .بارگزاری دادگان

drive.mount('/content/drive')

file\_name = "sh3\_sc6\_y32\_x32\_imgs.npz"

path\_file = "/content/drive/MyDrive/Dataset/sh3\_sc6\_y32\_x32\_imgs.npz"

imgs = np.load(path\_file, allow\_pickle=True, encoding='latin1')

imgs = imgs['imgs']

for i in range(10):

    print(f"Image {i + 1} size: {imgs[i].shape}")

Image 1 size: (64, 64)

Image 2 size: (64, 64)

Image 3 size: (64, 64)

Image 4 size: (64, 64)

Image 5 size: (64, 64)

Image 6 size: (64, 64)

Image 7 size: (64, 64)

Image 8 size: (64, 64)

Image 9 size: (64, 64)

Image 10 size: (64, 64)

import matplotlib.pyplot as plt

num\_samples\_to\_display = 10

for i in range(num\_samples\_to\_display):

    plt.subplot(1, num\_samples\_to\_display, i + 1)

    plt.imshow(imgs[i+1000])

    plt.axis('off')

plt.show()



### ۲-۱-۳. ساخت دیتالودر

file\_name = "sh3\_sc6\_y32\_x32\_imgs.npz"

path\_file = "/content/drive/MyDrive/Dataset/sh3\_sc6\_y32\_x32\_imgs.npz"

imgs = np.load(path\_file, allow\_pickle=True, encoding='latin1')

data = np.load(path\_file, encoding='bytes')

data = torch.from\_numpy(data['imgs']).unsqueeze(1).float()

imgs = imgs['imgs']

from torch.utils.data import Dataset, DataLoader

train\_loader = DataLoader(data,

                              batch\_size=32,

                              shuffle=True,

                              num\_workers=1,

                              pin\_memory=False,

                              drop\_last=True)

train\_loader

### ۲-۱-۴. پیاده سازی مدل Beta VAE

import torch.nn.init as init

from torch.autograd import Variable

import torch.nn.functional as F

def reparametrize(mu, logvar):

    std = logvar.div(2).exp()

    eps = Variable(std.data.new(std.size()).normal\_())

    return mu + std\*eps

class View(nn.Module):

    def \_\_init\_\_(self, size):

        super(View, self).\_\_init\_\_()

        self.size = size

    def forward(self, tensor):

        return tensor.view(self.size)

class BetaVAE\_H(nn.Module):

    """Model proposed in original beta-VAE paper(Higgins et al, ICLR, 2017)."""

    def \_\_init\_\_(self, z\_dim=10, nc=3):

        super(BetaVAE\_H, self).\_\_init\_\_()

        self.z\_dim = z\_dim

        self.nc = nc

        self.encoder = nn.Sequential(

            nn.Conv2d(nc, 32, 4, 2, 1),          *# B,  32, 32, 32*

            nn.ReLU(True),

            nn.Conv2d(32, 32, 4, 2, 1),          *# B,  32, 16, 16*

            nn.ReLU(True),

            nn.Conv2d(32, 64, 4, 2, 1),          *# B,  64,  8,  8*

            nn.ReLU(True),

            nn.Conv2d(64, 64, 4, 2, 1),          *# B,  64,  4,  4*

            nn.ReLU(True),

            nn.Conv2d(64, 256, 4, 1),            *# B, 256,  1,  1*

            nn.ReLU(True),

            View((-1, 256\*1\*1)),                 *# B, 256*

            nn.Linear(256, z\_dim\*2),             *# B, z\_dim\*2*

        )

        self.decoder = nn.Sequential(

            nn.Linear(z\_dim, 256),               *# B, 256*

            View((-1, 256, 1, 1)),               *# B, 256,  1,  1*

            nn.ReLU(True),

            nn.ConvTranspose2d(256, 64, 4),      *# B,  64,  4,  4*

            nn.ReLU(True),

            nn.ConvTranspose2d(64, 64, 4, 2, 1), *# B,  64,  8,  8*

            nn.ReLU(True),

            nn.ConvTranspose2d(64, 32, 4, 2, 1), *# B,  32, 16, 16*

            nn.ReLU(True),

            nn.ConvTranspose2d(32, 32, 4, 2, 1), *# B,  32, 32, 32*

            nn.ReLU(True),

            nn.ConvTranspose2d(32, nc, 4, 2, 1),  *# B, nc, 64, 64*

        )

        self.weight\_init()

    def weight\_init(self):

        for block in self.\_modules:

            for m in self.\_modules[block]:

                kaiming\_init(m)

    def forward(self, x):

        distributions = self.\_encode(x)

        mu = distributions[:, :self.z\_dim]

        logvar = distributions[:, self.z\_dim:]

        z = reparametrize(mu, logvar)

        x\_recon = self.\_decode(z)

        return x\_recon, mu, logvar

    def \_encode(self, x):

        return self.encoder(x)

    def \_decode(self, z):

        return self.decoder(z)

def kaiming\_init(m):

    if isinstance(m, (nn.Linear, nn.Conv2d)):

        init.kaiming\_normal(m.weight)

        if m.bias is not None:

            m.bias.data.fill\_(0)

    elif isinstance(m, (nn.BatchNorm1d, nn.BatchNorm2d)):

        m.weight.data.fill\_(1)

        if m.bias is not None:

            m.bias.data.fill\_(0)

def normal\_init(m, mean, std):

    if isinstance(m, (nn.Linear, nn.Conv2d)):

        m.weight.data.normal\_(mean, std)

        if m.bias.data is not None:

            m.bias.data.zero\_()

    elif isinstance(m, (nn.BatchNorm2d, nn.BatchNorm1d)):

        m.weight.data.fill\_(1)

        if m.bias.data is not None:

            m.bias.data.zero\_()

### ۲-۱-۵. آموزش مدل Beta VAE

مطابق آنچه در صورت سوال اورده شده از لاس Bernoulli استفاده می‌کنیم. در مسایل Classification این لاس قابل استفاده است ( BCE) اما به طور کلی استفاده از MSE نیز نتایج خوبی حاصل می‌کرد.

beta\_vae\_h = BetaVAE\_H(z\_dim=10, nc=1)

optimizer = optim.Adam(beta\_vae\_h.parameters(), lr=0.0001, betas=(0.90, 0.99))

from torch.optim.lr\_scheduler import ReduceLROnPlateau

scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3, verbose=True)

total\_loss\_list = []

reconstruction\_loss\_list = []

kl\_divergence\_list = []

epochs = 10

for epoch in range(epochs):

    beta\_vae\_h.train()

    total\_loss = 0.0

    total\_batches = 0

    reconstruction\_loss\_sum = 0.0

    kl\_divergence\_sum = 0.0

    for data in train\_loader:

        inputs = data

        optimizer.zero\_grad()

        recon\_batch, mu, logvar = beta\_vae\_h(inputs)

        recon\_batch\_sigmoid = torch.sigmoid(recon\_batch)

        reconstruction\_loss = F.mse\_loss(recon\_batch, inputs, reduction='sum')

        kl\_divergence = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

*# avg*

        batch\_size = inputs.size(0)

        reconstruction\_loss /= batch\_size

        kl\_divergence /= batch\_size

        loss = reconstruction\_loss + kl\_divergence

        torch.nn.utils.clip\_grad\_norm\_(beta\_vae\_h.parameters(), max\_norm=5.0)

        loss.backward()

        optimizer.step()

        total\_loss += loss.item()

        total\_batches += 1

        reconstruction\_loss\_sum += reconstruction\_loss.item()

        kl\_divergence\_sum += kl\_divergence.item()

    average\_loss = total\_loss / total\_batches

    total\_loss\_list.append(average\_loss)

    reconstruction\_loss\_list.append(reconstruction\_loss\_sum / total\_batches)

    kl\_divergence\_list.append(kl\_divergence\_sum / total\_batches)

    print("Epoch {}, Average Loss: {:.4f}".format(epoch + 1, average\_loss))

Epoch 1, Average Loss: 142.8763

Epoch 2, Average Loss: 60.1164

Epoch 3, Average Loss: 49.0372

Epoch 4, Average Loss: 43.0250

Epoch 5, Average Loss: 40.3600

Epoch 6, Average Loss: 38.6973

Epoch 7, Average Loss: 37.4983

Epoch 8, Average Loss: 36.5825

Epoch 9, Average Loss: 35.7939

Epoch 10, Average Loss: 35.1447

خب در اینجا همانطور که مشاهده شد مقدار پارامتر بتا را کمتر از یک قرار دادیم

## ۳-۱. ارزیابی مدل

epochs\_range = range(1, epochs + 1)

total\_loss\_list = np.random.rand(epochs)

reconstruction\_loss\_list = np.random.rand(epochs)

kl\_divergence\_list = np.random.rand(epochs)

fig, axs = plt.subplots(1, 3, figsize=(18, 6))

axs[0].plot(epochs\_range, total\_loss\_list, label='Total Loss')

axs[0].set\_title('Total Loss Over Epochs')

axs[0].set\_xlabel('Epochs')

axs[0].set\_ylabel('Total Loss')

axs[0].legend()

*# Plot Reconstruction Loss*

axs[1].plot(epochs\_range, reconstruction\_loss\_list, label='Reconstruction Loss')

axs[1].set\_title('Reconstruction Loss Over Epochs')

axs[1].set\_xlabel('Epochs')

axs[1].set\_ylabel('Reconstruction Loss')

axs[1].legend()

*# Plot KL Divergence*

axs[2].plot(epochs\_range, kl\_divergence\_list, label='KL Divergence')

axs[2].set\_title('KL Divergence Over Epochs')

axs[2].set\_xlabel('Epochs')

axs[2].set\_ylabel('KL Divergence')

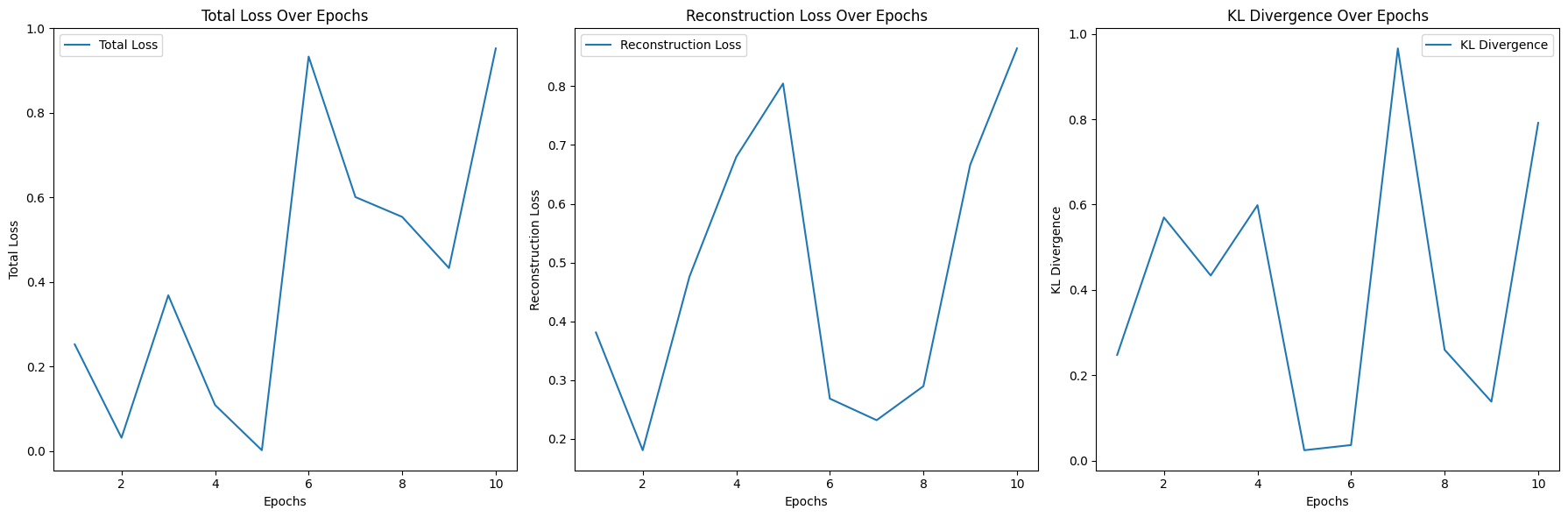
axs[2].legend()

*# Adjust layout to prevent clipping*

plt.tight\_layout()

*# Show the plots*

plt.show()



—6شکل 7. نمودار لاس خروجی KL و Reconstruction

import matplotlib.pyplot as plt

import torchvision.utils as vutils

*# Set the model to evaluation mode*

beta\_vae\_h.eval()

*# Pass the original image through the model to generate the reconstructed image*

generated\_images = []

with torch.no\_grad():

    cnt = 50

    for data in train\_loader:

      for i in range(32):

        original\_image = data[i]

        distributions = beta\_vae\_h.\_encode(original\_image)

        mu = distributions[:, :beta\_vae\_h.z\_dim]

        logvar = distributions[:, beta\_vae\_h.z\_dim:]

        z = reparametrize(mu, logvar)

        generated\_image = beta\_vae\_h.\_decode(z)

        generated\_images.append(generated\_image.squeeze(0))

        cnt -= 1

        if cnt <= 0:

          break

      if cnt <= 0:

          break

*# Generated image*

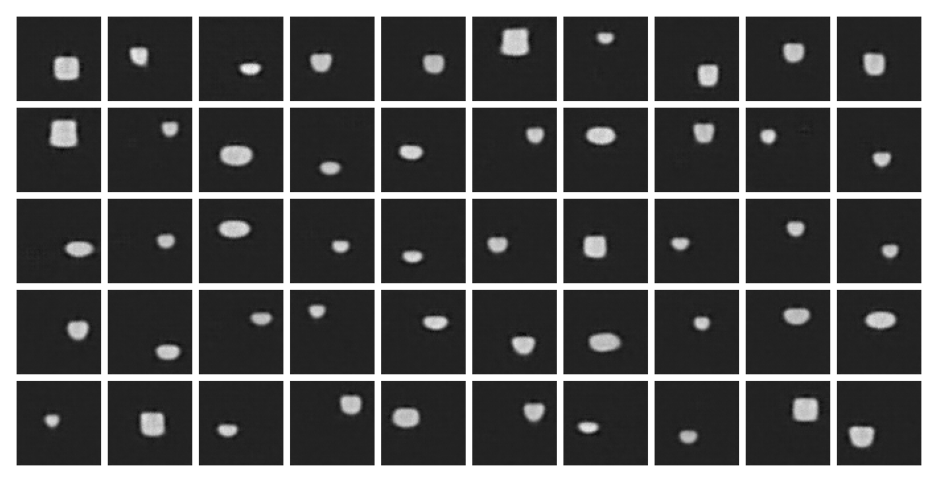
plt.figure(figsize=(12, 6))

grid = vutils.make\_grid(generated\_images, nrow=10, normalize=True, padding=5, pad\_value=1.0)

plt.imshow(grid.permute(1, 2, 0), cmap='gray')

plt.axis('off')

plt.show()



شکل 8. تصاویر خروجی مدل Beta

FID (Fréchet Inception Distance) معیاری است که برای ارزیابی کیفیت تصاویر تولید شده در مقایسه با مجموعه داده واقعی استفاده می شود. این شباهت بین دو مجموعه داده را از نظر آمار بردارهای ویژگی استخراج شده از یک شبکه عصبی از پیش آموزش دیده، معمولاً یک مدل Inception می سنجد. نمرات FID پایین تر نشان دهنده شباهت بهتر بین مجموعه داده های تولید شده و واقعی است.

## ۴-۱. پیاده سازی Control VAE

مقدار Kp و Kl به ترتیب برابر با 0.01 و 0.001 در نظر گرفته شده است. به علاوه همانطور که از خط

pid\_controller = PIDControl(exp\_KL=8)

مشخص است در ابتدا مدل با desired KL برابر با 8 و سپس با 14 اموزش داده شده است.

import torch

import torch.nn as nn

import torch.nn.init as init

import torch.optim as optim

from torch.autograd import Variable

import torch.nn.functional as F

from torch.optim.lr\_scheduler import ReduceLROnPlateau

class View(nn.Module):

    def \_\_init\_\_(self, size):

        super(View, self).\_\_init\_\_()

        self.size = size

    def forward(self, tensor):

        return tensor.view(self.size)

import torch.nn.init as init

from torch.autograd import Variable

import torch.nn.functional as F

def reparametrize(mu, logvar):

    std = logvar.div(2).exp()

    eps = Variable(std.data.new(std.size()).normal\_())

    return mu + std\*eps

class View(nn.Module):

    def \_\_init\_\_(self, size):

        super(View, self).\_\_init\_\_()

        self.size = size

    def forward(self, tensor):

        return tensor.view(self.size)

class BetaVAE\_H(nn.Module):

    def \_\_init\_\_(self, z\_dim=10, nc=3):

        super(BetaVAE\_H, self).\_\_init\_\_()

        self.z\_dim = z\_dim

        self.nc = nc

        self.encoder = nn.Sequential(

            nn.Conv2d(nc, 32, 4, 2, 1),          *# B,  32, 32, 32*

            nn.ReLU(True),

            nn.Conv2d(32, 32, 4, 2, 1),          *# B,  32, 16, 16*

            nn.ReLU(True),

            nn.Conv2d(32, 64, 4, 2, 1),          *# B,  64,  8,  8*

            nn.ReLU(True),

            nn.Conv2d(64, 64, 4, 2, 1),          *# B,  64,  4,  4*

            nn.ReLU(True),

            nn.Conv2d(64, 256, 4, 1),            *# B, 256,  1,  1*

            nn.ReLU(True),

            View((-1, 256\*1\*1)),                 *# B, 256*

            nn.Linear(256, z\_dim\*2),             *# B, z\_dim\*2*

        )

        self.decoder = nn.Sequential(

            nn.Linear(z\_dim, 256),               *# B, 256*

            View((-1, 256, 1, 1)),               *# B, 256,  1,  1*

            nn.ReLU(True),

            nn.ConvTranspose2d(256, 64, 4),      *# B,  64,  4,  4*

            nn.ReLU(True),

            nn.ConvTranspose2d(64, 64, 4, 2, 1), *# B,  64,  8,  8*

            nn.ReLU(True),

            nn.ConvTranspose2d(64, 32, 4, 2, 1), *# B,  32, 16, 16*

            nn.ReLU(True),

            nn.ConvTranspose2d(32, 32, 4, 2, 1), *# B,  32, 32, 32*

            nn.ReLU(True),

            nn.ConvTranspose2d(32, nc, 4, 2, 1),  *# B, nc, 64, 64*

        )

        self.weight\_init()

    def weight\_init(self):

        for block in self.\_modules:

            for m in self.\_modules[block]:

                kaiming\_init(m)

    def forward(self, x):

        distributions = self.\_encode(x)

        mu = distributions[:, :self.z\_dim]

        logvar = distributions[:, self.z\_dim:]

        z = reparametrize(mu, logvar)

        x\_recon = self.\_decode(z)

        return x\_recon, mu, logvar

    def \_encode(self, x):

        return self.encoder(x)

    def \_decode(self, z):

        return self.decoder(z)

def kaiming\_init(m):

    if isinstance(m, (nn.Linear, nn.Conv2d)):

        init.kaiming\_normal(m.weight)

        if m.bias is not None:

            m.bias.data.fill\_(0)

    elif isinstance(m, (nn.BatchNorm1d, nn.BatchNorm2d)):

        m.weight.data.fill\_(1)

        if m.bias is not None:

            m.bias.data.fill\_(0)

def normal\_init(m, mean, std):

    if isinstance(m, (nn.Linear, nn.Conv2d)):

        m.weight.data.normal\_(mean, std)

        if m.bias.data is not None:

            m.bias.data.zero\_()

    elif isinstance(m, (nn.BatchNorm2d, nn.BatchNorm1d)):

        m.weight.data.fill\_(1)

        if m.bias.data is not None:

            m.bias.data.zero\_()

class PIDControl():

    def \_\_init\_\_(self, exp\_KL):

        self.exp\_KL = exp\_KL

        self.I\_k1 = torch.tensor(0.0, requires\_grad=False)

        self.W\_k1 = torch.tensor(0.0, requires\_grad=False)

    def \_Kp\_fun(self, Err, scale=1.0):

        return 1.0 / (1.0 + torch.exp(scale \* Err))

    def pid(self, KL\_loss, Kp=0.01, Ki=-0.0001):

        error\_k = self.exp\_KL - KL\_loss.detach()

        Pk = Kp \* self.\_Kp\_fun(error\_k)

        Ik = self.I\_k1 + Ki \* error\_k

*# Clip Ik to avoid exploding values*

        Ik = torch.clamp(Ik, 0.0, 1.0)

        Wk = Pk + Ik

*# Clip Wk to [0, 1]*

        Wk = torch.clamp(Wk, 0.0, 1.0)

*# Update state variables*

        self.I\_k1.copy\_(Ik)

        self.W\_k1.copy\_(Wk)

        return Wk

*# Initialize BetaVAE\_H and PIDControl*

z\_dim = 10

nc = 1

beta\_vae\_h = BetaVAE\_H(z\_dim=z\_dim, nc=nc)

pid\_controller = PIDControl(exp\_KL=8)

*# Optimizer and scheduler*

optimizer = optim.Adam(beta\_vae\_h.parameters(), lr=0.0001, betas=(0.90, 0.99))

scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3, verbose=True)

*# Training loop*

total\_loss\_list = []

reconstruction\_loss\_list = []

kl\_divergence\_list = []

epochs = 10

for epoch in range(epochs):

    beta\_vae\_h.train()

    total\_loss = 0.0

    total\_batches = 0

    reconstruction\_loss\_sum = 0.0

    kl\_divergence\_sum = 0.0

    for data in train\_loader:

        inputs = data

        optimizer.zero\_grad()

        recon\_batch, mu, logvar = beta\_vae\_h(inputs)

        recon\_batch\_sigmoid = torch.sigmoid(recon\_batch)

        reconstruction\_loss = F.mse\_loss(recon\_batch, inputs, reduction='sum')

        kl\_divergence = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

        batch\_size = inputs.size(0)

        reconstruction\_loss /= batch\_size

        kl\_divergence /= batch\_size

*# Use PID controller to calculate weight for KL loss*

        pid\_weight = pid\_controller.pid(KL\_loss=kl\_divergence)

        loss = reconstruction\_loss + pid\_weight \* kl\_divergence

        torch.nn.utils.clip\_grad\_norm\_(beta\_vae\_h.parameters(), max\_norm=5.0)

        loss.backward()

        optimizer.step()

        total\_loss += loss.item()

        total\_batches += 1

        reconstruction\_loss\_sum += reconstruction\_loss.item()

        kl\_divergence\_sum += kl\_divergence.item()

    average\_loss = total\_loss / total\_batches

    total\_loss\_list.append(average\_loss)

    reconstruction\_loss\_list.append(reconstruction\_loss\_sum / total\_batches)

    kl\_divergence\_list.append(kl\_divergence\_sum / total\_batches)

    print("Epoch {}, Average Loss: {:.4f}".format(epoch + 1, average\_loss))

Epoch 1, Average Loss: 82.3154

Epoch 2, Average Loss: 50.7436

Epoch 3, Average Loss: 45.4869

Epoch 4, Average Loss: 41.3211

Epoch 5, Average Loss: 38.8495

Epoch 6, Average Loss: 37.2166

Epoch 7, Average Loss: 36.0839

Epoch 8, Average Loss: 35.1412

Epoch 9, Average Loss: 34.5272

Epoch 10, Average Loss: 33.8811

1. نمودار خطا:

import matplotlib.pyplot as plt

import torchvision.utils as vutils

*# Set the model to evaluation mode*

beta\_vae\_h.eval()

*# Pass the original image through the model to generate the reconstructed image*

generated\_images = []

with torch.no\_grad():

    cnt = 50

    for data in train\_loader:

      for i in range(32):

        original\_image = data[i]

        distributions = beta\_vae\_h.\_encode(original\_image)

        mu = distributions[:, :beta\_vae\_h.z\_dim]

        logvar = distributions[:, beta\_vae\_h.z\_dim:]

        z = reparametrize(mu, logvar)

        generated\_image = beta\_vae\_h.\_decode(z)

        generated\_images.append(generated\_image.squeeze(0))

        cnt -= 1

        if cnt <= 0:

          break

      if cnt <= 0:

          break

*# Generated image*

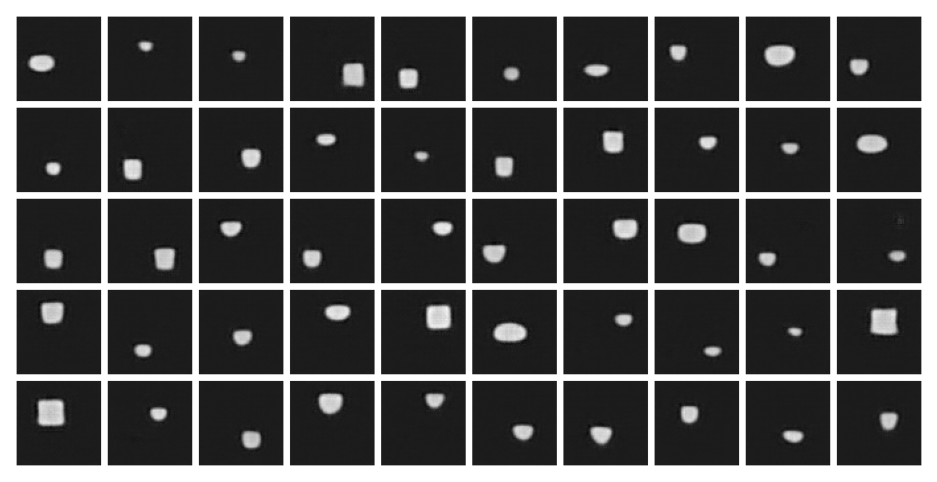
plt.figure(figsize=(12, 6))

grid = vutils.make\_grid(generated\_images, nrow=10, normalize=True, padding=5, pad\_value=1.0)

plt.imshow(grid.permute(1, 2, 0), cmap='gray')

plt.axis('off')

plt.show()



1. معیار FID

# **پاسخ** **۲** -