In [1]:
```
from platform import python_version
print('Notebook was originally ran with Python version:')
print(python_version())
```

```
Notebook was originally ran with Python version:
3.6.6
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from DataPreparation.dataset_preparation import get_SVHN_dataset
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

A. Load the Dataset

Set the data directory to the path where the following files exist: train_32x32.mat, test_32x32.mat

In [3]:
```
data_dir = 'Dataset/SVHN/'
validation_split = 0.2
seed = 6135
```

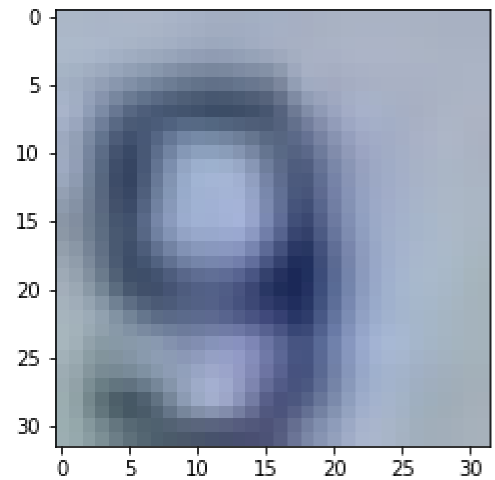
In [4]:
```
X_train, y_train, X_val, y_val, X_test, y_test, X_train_max = get_SVHN_dataset(da

scale = lambda img: X_train_max * ((img / 2) + 0.5)
print('X_train shape: ', X_train.shape)
print('y_train shape: ', y_train.shape)
print('X_val shape: ', X_test.shape)
print('y_val shape: ', y_test.shape)
print('X_test shape: ', X_test.shape)
print('y_test shape: ', y_test.shape)
```

```
Generating numpy dataset from existing .mat files in Dataset/SVHN/
X_train shape:  (58606, 3, 32, 32)
y_train shape:  (58606,)
X_val shape:  (26032, 3, 32, 32)
y_val shape:  (26032,)
X_test shape:  (26032, 3, 32, 32)
y_test shape:  (26032,)
```

Visualization sanity check

In [5]:
```
plt.imshow(np.transpose(scale(X_test[250]), (1, 2, 0)).astype('uint8'))
plt.show()
```



In [6]:
```
batch_size = 256
X_train_ = TensorDataset(torch.from_numpy(X_train))
loader_train  = DataLoader(X_train_, batch_size=batch_size, shuffle=True)

X_val_ = TensorDataset(torch.from_numpy(X_val))
loader_val = DataLoader(X_val_, batch_size=batch_size, shuffle=False)

X_test_ = TensorDataset(torch.from_numpy(X_test))
loader_test = DataLoader(X_test_, batch_size=batch_size, shuffle=False)
```

B. Train the Model

Select device

In [7]:
```
# CUDA for PyTorch
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
print('Using device=GPU') if use_cuda else print('Using device=CPU')
```

```
Using device=GPU
```

VAE

In [8]:
```
from models.vae import VAE
num_latent = 100
model = VAE(num_latent).to(device)
```

In [9]:
```
# Hyperparameters
learning_rate = 3e-4
num_epochs = 100
```

In [10]:
```
from utils.train_eval_utils import train_model
print('~~~ Training with GPU ~~~') if use_cuda else print('~~~ Training with CPU ~
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print('Model has %.2fM trainable parameters.\n' % (num_params/1e6))

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_history = train_model(model, optimizer, loader_train,
                            loader_val, num_epochs,
                            device)
```

```
Epoch 96:
Train: elbo -71.9778, likelihood -39.4060, KL 32.5719
Validation: elbo -79.8376, likelihood -46.6083, KL 33.2293
-----------
Epoch 97:
Train: elbo -71.5037, likelihood -38.9638, KL 32.5399
Validation: elbo -80.5824, likelihood -47.8544, KL 32.7280
-----------
Epoch 98:
Train: elbo -71.4471, likelihood -38.8846, KL 32.5625
Validation: elbo -81.2226, likelihood -48.6840, KL 32.5386
-----------
Epoch 99:
Train: elbo -71.7573, likelihood -39.1462, KL 32.6111
Validation: elbo -80.3220, likelihood -46.8925, KL 33.4295
-----------
Epoch 100:
Train: elbo -71.4943, likelihood -38.9519, KL 32.5424
Validation: elbo -80.3651, likelihood -47.8037, KL 32.5614
-----------
```

In [11]:
```
PATH = 'vae100.pt'
torch.save(model, PATH)
```

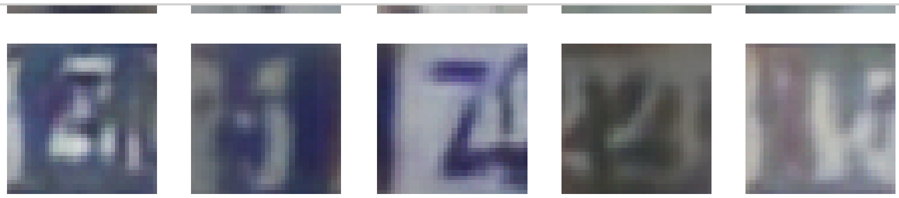
GAN

In [12]:
```
from models.gan import Generator, Discriminator
num_latent = 100
generator = Generator(num_latent).to(device)
discriminator = Discriminator().to(device)
```

In [13]:
```
# Hyperparameters
learning_rate = 1e-4
b1 = 0.5
b2 = 0.999
num_iterations = 20000
d_iterations = 5
```

In [14]:
```
from utils.train_eval_utils_gan import train_model
print('~~~ Training with GPU ~~~') if use_cuda else print('~~~ Training with CPU ~
num_params = sum(p.numel() for p in generator.parameters() if p.requires_grad)
print('Generator has %.2fM trainable parameters.' % (num_params/1e6))
num_params = sum(p.numel() for p in discriminator.parameters() if p.requires_grad
print('Discriminator has %.2fM trainable parameters.\n' % (num_params/1e6))

optimizer_d = torch.optim.Adam(discriminator.parameters(), lr=learning_rate, beta
optimizer_g = torch.optim.Adam(generator.parameters(), lr=learning_rate, betas=(b
train_model(discriminator, generator, optimizer_d, optimizer_g,
            loader_train, loader_val, num_iterations, d_iterations, device, use_c
```



GAN SVHN generations

In [15]:
```
PATH = 'generator%d.pt' % num_iterations
torch.save(generator, PATH)
PATH = 'discriminator%d.pt' % num_iterations
torch.save(discriminator, PATH)
```

C. Qualitative Evaluation

C.1. Generate Samples

In [16]:
```
num_generations = 64
```

VAE

In [17]:
```
z = torch.randn(num_generations, num_latent).to(device)
model.eval()
with torch.no_grad():
    vae_generations = model.sample(z).cpu().numpy()
vae_generations = np.transpose(scale(vae_generations), (0, 2, 3, 1)).astype('uint
```

In [18]:
```
from utils.plotter import plot_and_save_images
plot_and_save_images(vae_generations, 'VAE')
```

VAE SVHN generations

GAN

In [19]:
```
z = torch.randn(num_generations, num_latent).to(device)
generator.eval()
with torch.no_grad():
    gan_generations = generator.sample(z).cpu().numpy()
gan_generations = np.transpose(scale(gan_generations), (0, 2, 3, 1)).astype('uint
```

In [20]:
```
from utils.plotter import plot_and_save_images
plot_and_save_images(gan_generations, 'GAN_')
```

GAN_ SVHN generations

C.2. Disentanglement

VAE

In [188]:
```
epsilon = 1.5
top_n = 15
num_interp = 9
z = torch.randn((1, num_latent)).to(device)
```

In [189]:
```
# Find dimensions that result in the most change:
vae_disentanglement = np.zeros((num_latent, 3, 32, 32))

model.eval()
with torch.no_grad():
    original = model.sample(z).cpu().numpy()
    for i in range(num_latent):
        z_ = z.clone().detach()
        z_[0, i] += epsilon
        vae_disentanglement[i] = model.sample(z_).cpu().numpy()

diff = np.sum((vae_disentanglement - original)**2, axis=(1,2,3))
topn_features = diff.argsort()[-top_n:][::-1]

print('Most effective latent dimensions:')
print(topn_features)
```

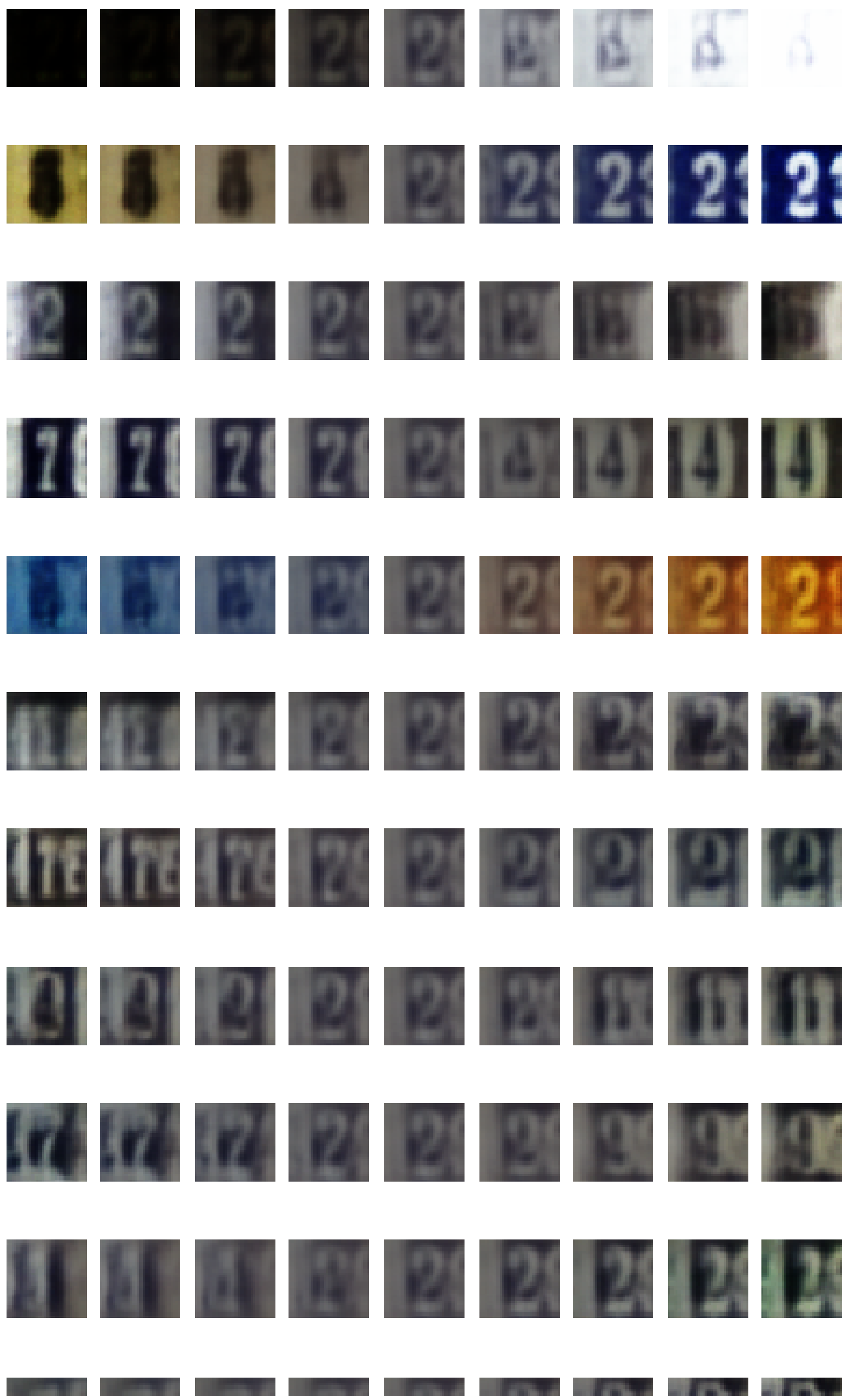
```
Most effective latent dimensions:
[36 48 78 90 32 12 37 56 59 49 47 82 42 35 10]
```

In [190]:
```
# Traverse along those dimensions:
vae_disentanglement = np.zeros((top_n, num_interp, 3, 32, 32))
with torch.no_grad():
    for i, feature in enumerate(topn_features):
        z_ = z.clone().detach()
        z_[0, feature] -= epsilon * np.floor(float(num_interp) / 2)
        for j in range(num_interp):
            if j != 0: z_[0, feature] += epsilon
            vae_disentanglement[i, j] = model.sample(z_).cpu().numpy()

vae_disentanglement = np.transpose(scale(vae_disentanglement), (0, 1, 3, 4, 2)).a
```

In [191]:
```
# Plot and save results
from utils.plotter import plot_and_save_disentanglement
plot_and_save_disentanglement(vae_disentanglement, num_rows=top_n, num_cols=num_i
```

VAE SVHN disentanglement



GAN

In [115]:
```
epsilon = 1.5
top_n = 15
num_interp = 9
z = torch.randn((1, num_latent)).to(device)
```

In [116]:
```
# Find dimensions that result in the most change:
gan_disentanglement = np.zeros((num_latent, 3, 32, 32))

generator.eval()
with torch.no_grad():
    original = generator.sample(z).cpu().numpy()
    for i in range(num_latent):
        z_ = z.clone().detach()
        z_[0, i] += epsilon
        gan_disentanglement[i] = generator.sample(z_).cpu().numpy()

diff = np.sum((gan_disentanglement - original)**2, axis=(1,2,3))
topn_features = diff.argsort()[-top_n:][::-1]

print('Most effective latent dimensions:')
print(topn_features)
```

```
Most effective latent dimensions:
[32 48 90 82 14  8 53  9 42 18 78  3 80 85 16]
```

In [117]:
```
# Traverse along those dimensions:
gan_disentanglement = np.zeros((top_n, num_interp, 3, 32, 32))
with torch.no_grad():
    for i, feature in enumerate(topn_features):
        z_ = z.clone().detach()
        z_[0, feature] -= epsilon * np.floor(float(num_interp) / 2)
        for j in range(num_interp):
            if j != 0: z_[0, feature] += epsilon
            gan_disentanglement[i, j] = generator.sample(z_).cpu().numpy()

gan_disentanglement = np.transpose(scale(gan_disentanglement), (0, 1, 3, 4, 2)).a
```

In [118]:
```
# Plot and save results
from utils.plotter import plot_and_save_disentanglement
plot_and_save_disentanglement(gan_disentanglement, num_rows=top_n, num_cols=num_i
```

GAN SVHN disentanglement

C.3. Interpolation

VAE latent

In [86]:
```
num_interp = 11

z0 = torch.randn((1, num_latent)).to(device)
z1 = torch.randn((1, num_latent)).to(device)
```

In [87]:
```
scales = np.linspace(0, 1, num_interp)
vae_latent = np.zeros((num_interp, 3, 32, 32))
with torch.no_grad():
    for i, s in enumerate(scales):
        z = (s * z0) + ((1 - s) * z1)
        vae_latent[i] = model.sample(z).cpu().numpy()

vae_latent = np.transpose(scale(vae_latent), (0, 2, 3, 1)).astype('uint8')
```

In [88]:
```
from utils.plotter import plot_and_save_interpolation
plot_and_save_interpolation(vae_latent, num_cols=num_interp, spacename='latent', 
```

VAE SVHN latent interpolation

VAE data

In [89]:
```
scales = np.linspace(0, 1, num_interp)
vae_data = np.zeros((num_interp, 3, 32, 32))
with torch.no_grad():
    data0 = model.sample(z0).cpu().numpy()
    data1 = model.sample(z1).cpu().numpy()

for i, s in enumerate(scales):
    data = (s * data0) + ((1 - s) * data1)
    vae_data[i] = data

vae_data = np.transpose(scale(vae_data), (0, 2, 3, 1)).astype('uint8')
```

In [90]:
```
plot_and_save_interpolation(vae_data, num_cols=num_interp, spacename='data', mode
```

VAE SVHN data interpolation

GAN latent

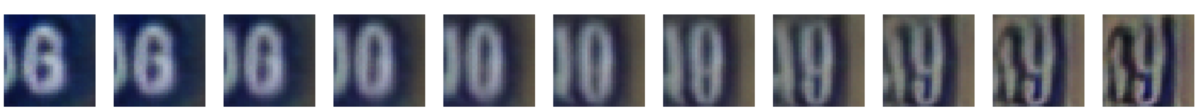
In [94]:
```
z0 = torch.randn((1, num_latent)).to(device)
z1 = torch.randn((1, num_latent)).to(device)
```

In [95]:
```
scales = np.linspace(0, 1, num_interp)
gan_latent = np.zeros((num_interp, 3, 32, 32))
with torch.no_grad():
    for i, s in enumerate(scales):
        z = (s * z0) + ((1 - s) * z1)
        gan_latent[i] = generator.sample(z).cpu().numpy()

gan_latent = np.transpose(scale(gan_latent), (0, 2, 3, 1)).astype('uint8')
```

In [96]:
```
from utils.plotter import plot_and_save_interpolation
plot_and_save_interpolation(gan_latent, num_cols=11, spacename='latent', modelnam
```

GAN SVHN latent interpolation



GAN data

In [97]:
```
scales = np.linspace(0, 1, num_interp)
gan_data = np.zeros((num_interp, 3, 32, 32))
with torch.no_grad():
    data0 = generator.sample(z0).cpu().numpy()
    data1 = generator.sample(z1).cpu().numpy()

for i, s in enumerate(scales):
    data = (s * data0) + ((1 - s) * data1)
    gan_data[i] = data

gan_data = np.transpose(scale(gan_data), (0, 2, 3, 1)).astype('uint8')
```

In [98]:
```
plot_and_save_interpolation(gan_data, num_cols=num_interp, spacename='data', mode
```

GAN SVHN data interpolation

D. Quantitative Evaluations

In [39]:
```
num_generations = 1000
```

VAE Generate & Store Samples

In [40]:
```
z = torch.randn(num_generations, num_latent).to(device)
model.eval()
with torch.no_grad():
    vae_generations = model.sample(z).cpu().numpy()
vae_generations = np.transpose(scale(vae_generations), (0, 2, 3, 1)).astype('uint
```

In [41]:
```
from utils.plotter import sample_saver
sample_saver(vae_generations, path='samples\\VAE\\subfolder')
```

VAE FID score

In [138]:
```
! python score_fid.py --model=svhn_classifier.pt samples\VAE
```

```
Test
Using downloaded and verified file: Dataset\SVHN\test_32x32.mat
Used epsilon: 1.0E-09
FID score: 31648.761138723774
```

GAN Generate & Store Samples

In [43]:
```
z = torch.randn(num_generations, num_latent).to(device)
generator.eval()
with torch.no_grad():
    gan_generations = generator.sample(z).cpu().numpy()
gan_generations = np.transpose(scale(gan_generations), (0, 2, 3, 1)).astype('uint
```

In [44]:
```
from utils.plotter import sample_saver
sample_saver(gan_generations, path='samples\\GAN\\subfolder')
```

GAN FID score

In [139]:
```
! python score_fid.py --model=svhn_classifier.pt samples\GAN
```

```
Test
Using downloaded and verified file: Dataset\SVHN\test_32x32.mat
Used epsilon: 1.0E+00
FID score: 8525.787152984225
```

In [ ]: