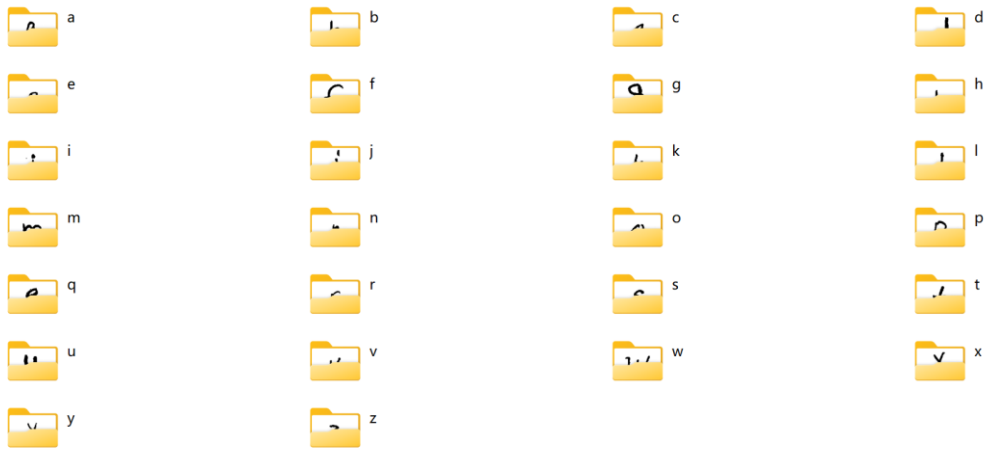
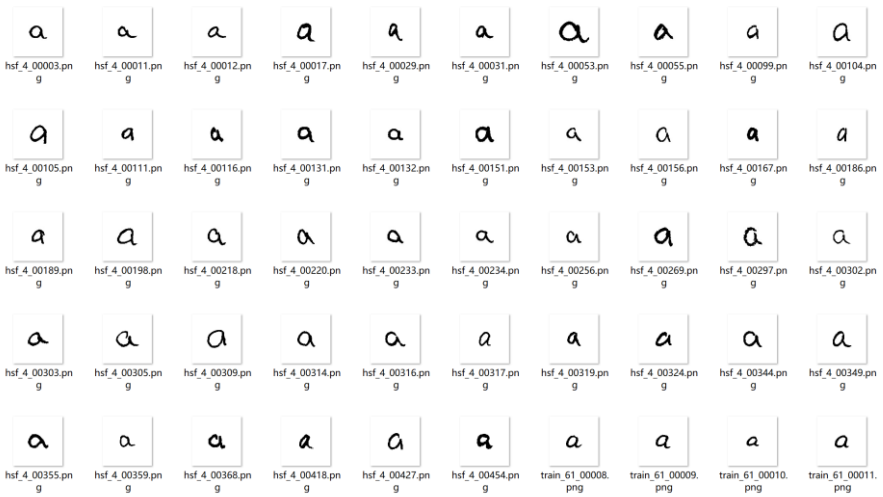
人工智能学习笔记十一——基于 aaegan 的手写英文欺骗样本生成

本文将使用 aaegan 对抗神经网络，来生成手写英文的对抗性欺骗样本。

本文使用的数据集放在了 csdn 上，地址为【免费】手写小写英文字母数据集_英文字母数据集-机器学习文档类资源-CSDN 文库，该数据集包含了 26 个小写的手写英文字母





在介绍 aaegan 之前，首先介绍一下 gan 对抗神经网络。

对抗神经网络（GAN，Generative Adversarial Networks）是一种深度学习方法，它通过两个神经网络（生成器和判别器）之间的博弈过程来生成具有类似于真实数据分布的新数据。GAN 的原理可以概括为以下几个步骤：

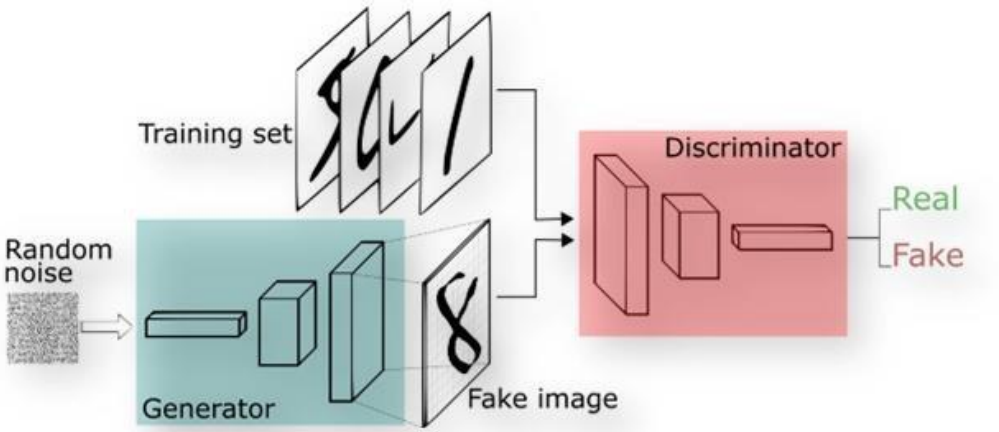
1，生成器（Generator）：生成器是一个随机生成数据的神经网络，它接收一个噪声向量（例如，高斯分布）作为输入，并输出一个生成的样本（例如，图像）。生成器的目标是生成与真实数据相似的样本。

2，判别器（Discriminator）：判别器是一个用于区分真实数据和生成样本的神经网络。它接收一个样本（可以是真实数据或生成数据）作为输入，并输出这个样本来自真实数据的概率。判别器的目标是尽可能准确地区分真实数据和生成数据。

3，训练过程：在训练过程中，生成器和判别器交替进行更新。首先，生成器根据噪声向量生成一个样本，然后判别器尝试区分这个样本是真实数据还是生成数据。接下来，根据判别器的输出，生成器会调整其参数以生成更接近真实数据的样本。这个过程持续进行，直到生成器能够生成足以欺骗判别器的样本。

应用：训练完成后，生成器可以生成与真实数据相似的新数据。这些新数据可以用于扩充训练数据集、生成对抗样本（Adversarial Examples）进行模型安全性评估，或者用于其他图像处理任务（如图像修复、风格迁移等）。

通过这个原理，对抗神经网络可以在不使用真实数据的情况下生成具有类似于真实数据分布的新数据，从而解决了数据不足或数据难以获取的问题。同时，GAN在图像处理、自然语言处理、音频生成等领域取得了显著的成果，成为了深度学习领域的一个热门研究方向。



对抗神经网络（GAN）中的生成器（Generator）通常是一个神经网络，其结构可以有多种选择。生成器的主要目标是生成与真实数据分布相似的样本。在训练过程中，生成器接收一个噪声向量作为输入，并输出一个生成的样本。

生成器的结构可以包括以下几种：

1，多层感知机（MLP）：生成器可以是一个简单的多层感知机，其中包含多个全连接层和激活函数（如 ReLU 或 sigmoid）。通过堆叠多个全连接层，生成器可以学习复杂的函数映射，从而生成具有多样化特征的样本。

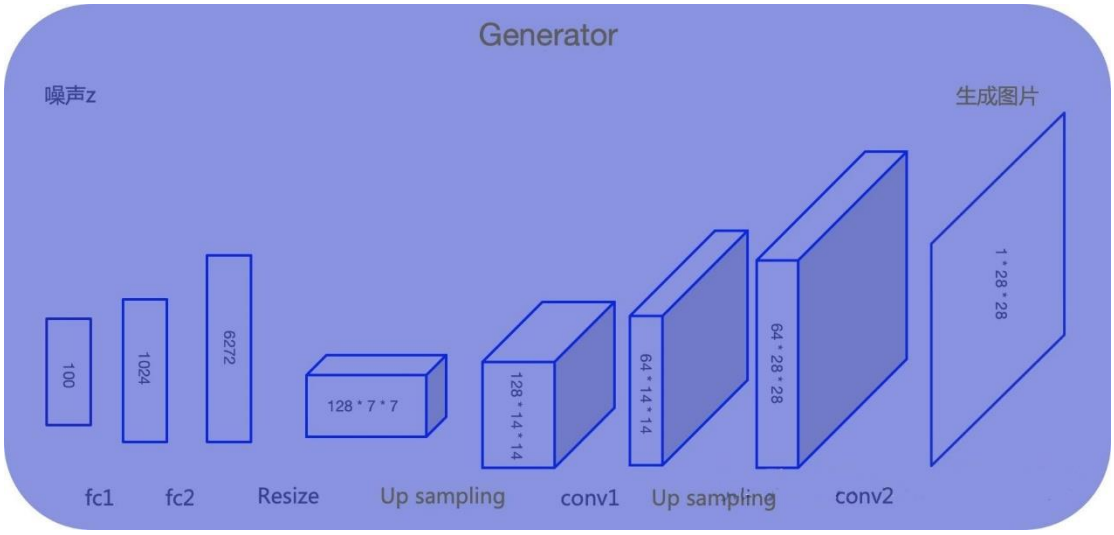
2，卷积神经网络（CNN）：生成器可以是一个卷积神经网络，其中包含多个卷积层、池化层和全连接层。卷积神经网络在图像生成任务中尤为常见，因为它们可以有效地学习图像的局部特征和结构。

3，自编码器（Autoencoder）：生成器可以是一个自编码器，其中包含一个编码器（用于将输入噪声压缩成低维表示）和一个解码器（用于将低维表示重建成生成的样本）。自编码器在生成具有特定特征的样本时表现良好，如生成器可以生成与输入噪声具有相似特征的样本。

4，变分自编码器（Variational Autoencoder，VAE）：生成器可以是一个变分自编码器，它包含一个编码器和一个解码器，以及一个用于生成样本的随机变量。变分自编码器在生成具有类似于真实数据分布的样本时表现良好，因为它们可以学习一个近似的后验分布。

这些结构都可以作为生成器的实现方式。在实际应用中，根据具体任务的需求和数据特点，可以选择合适的生成器结构进行训练。

举个例子，如下图所示就是一个非常简单的生成器：



它由两个全连接层，两个上采样层和两个卷积层组成，从而把一个 100 维的高斯随机噪声便成了大小为 28*28 的图片。

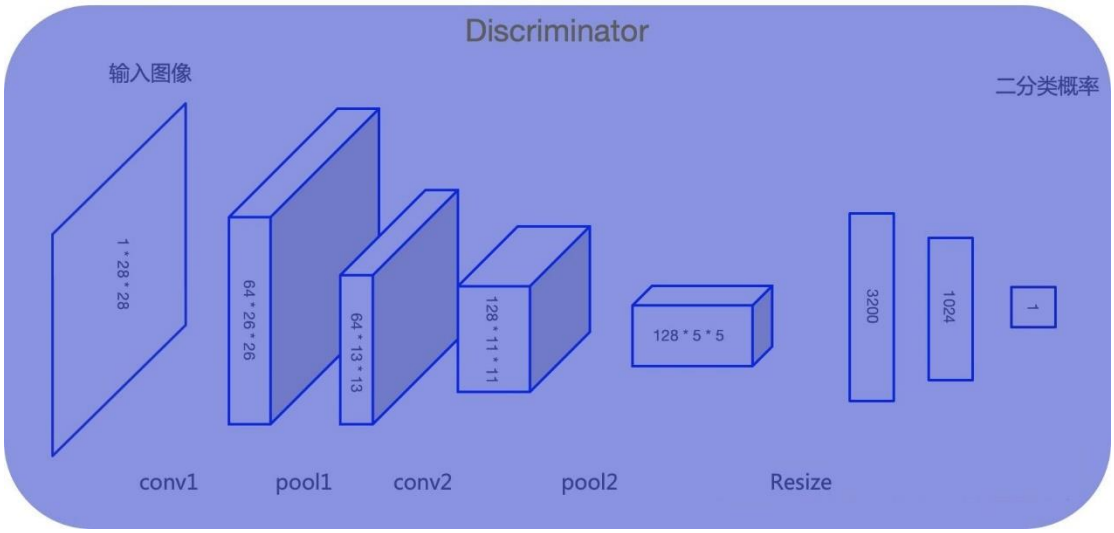
而对抗神经网络（GAN）中，判别器的结构通常是一个神经网络，其主要作用是区分生成器生成的虚假数据和真实数据。判别器在训练过程中与生成器进行对抗，生成器试图生成更逼真的图像以欺骗判别器，而判别器则试图更好地识别生成的图像。

判别器的结构可以根据具体任务和需求进行设计。一般情况下，判别器包括几个

全连接层和激活函数层，用于对输入数据进行特征提取和分类。在判别器中，常用的激活函数有 ReLU（Rectified Linear Unit）、sigmoid 和 tanh 等。

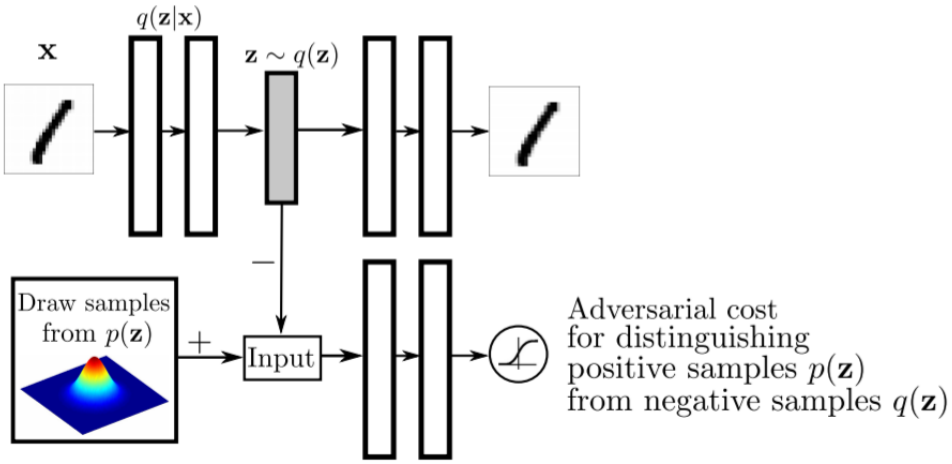
在训练过程中，判别器和生成器通过优化算法（如梯度下降法）不断更新各自的参数，以达到对抗的目的。判别器的性能可以通过其准确性来衡量，即在给定真实数据和生成数据时，判别器能正确识别真实数据的概率。

值得注意的是，判别器在训练过程中可能受到对抗样本攻击的影响。对抗样本是指生成器生成的具有特定属性的图像，这些图像能够欺骗判别器，使其无法正确识别。为了提高判别器的鲁棒性，研究者们提出了许多防御对策，如对抗训练、模型融合等。

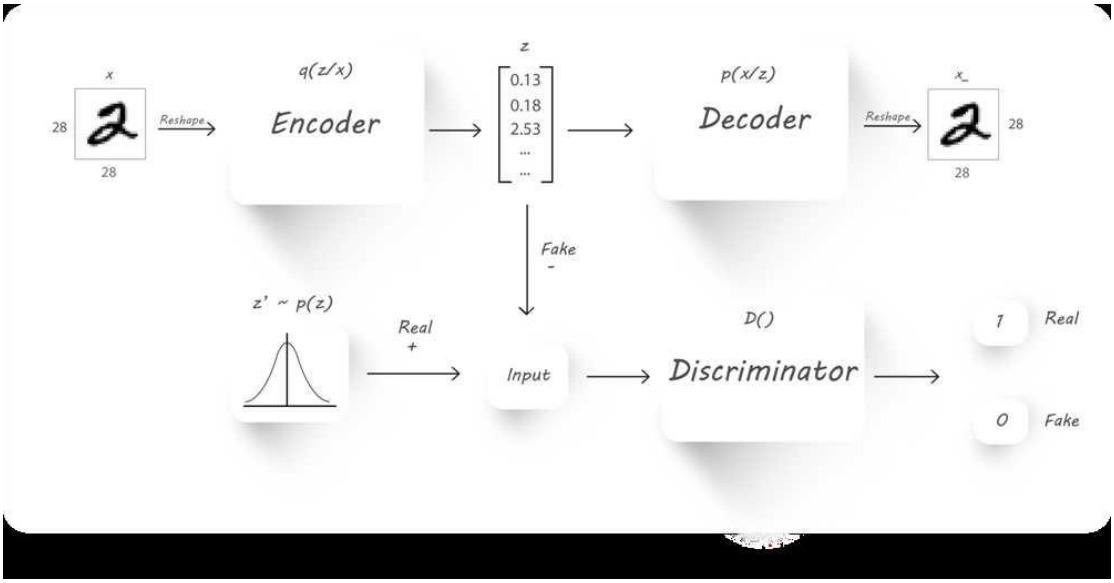


如上图所示就是一个简单的判别器，它的作用是分别真是样本和虚假样本。

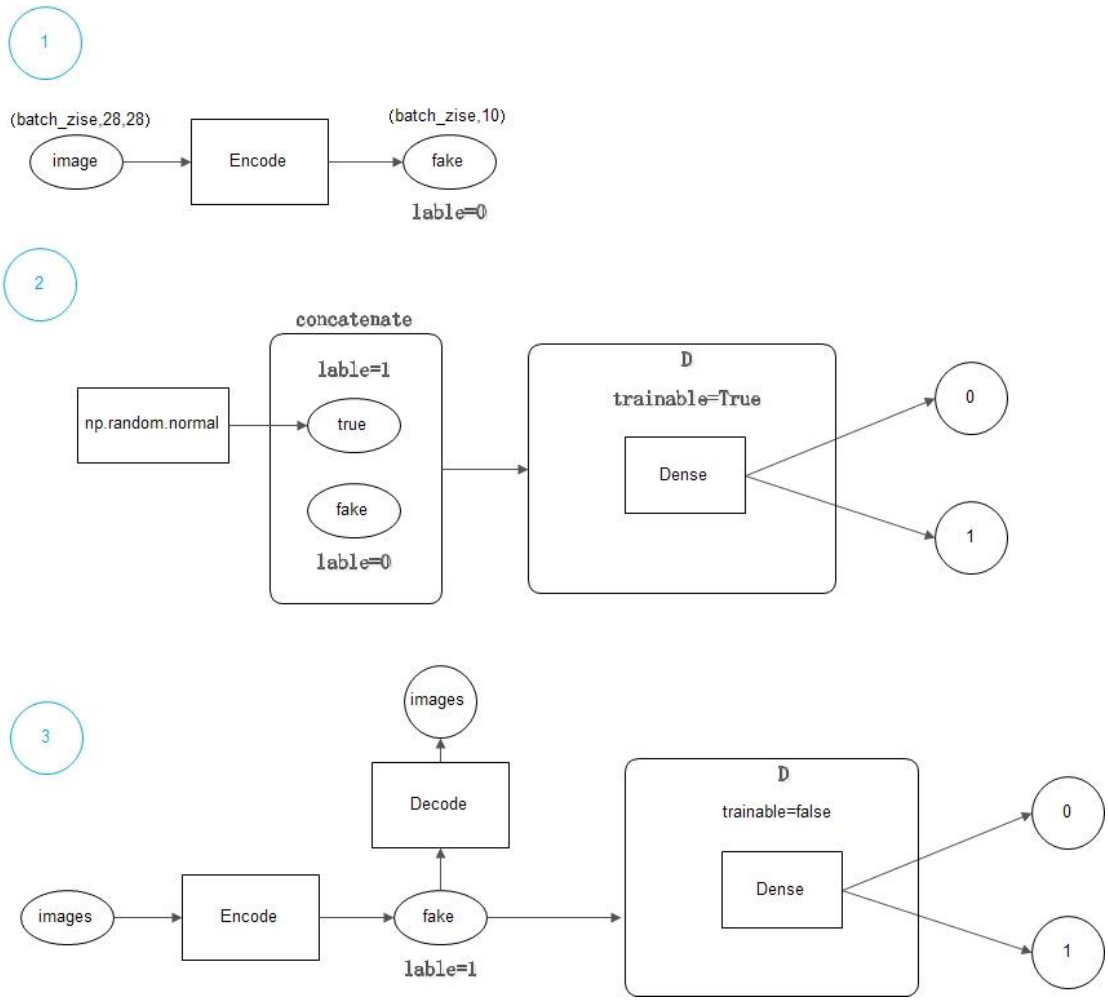
接着来介绍一下 aaegan 网络



下图便是其网络结构：



其按照如下图的 1，2，3 步顺序进行，首先图像通过编码器被转化成单维度数据，接着这个单维度数组和单位的高斯噪声分别做为判别器的正例和负例输入到判别器当中进行训练，最后冻结判别器的参数，训练编码器和解码器，使得编码器最后生成的单维度数组能够让判别器生成错误的答案。



接下来是训练部分，首先要保证以下的文件结构目录：

名称	修改日期	类型	大小
data	2023/8/28 11:01	文件夹	
aae.py	2023/8/28 10:40	Python File	8 KB
MergeLayer.py	2022/11/20 15:09	Python File	1 KB

其中 MergeLayer.py 的代码如下：

```
from keras.layers import Layer
import keras.backend as K

class MergeLayer(Layer):

    def __init__(self, **kwargs):
        super(MergeLayer, self).__init__(**kwargs)
```

```
    def compute_output_shape(self, input_shape):
        return (input_shape[0][0], input_shape[0][1])

    def call(self, x, mask=None):
        final_output = x[0] + K.random_normal(K.shape(x[0])) * K.exp(x[1] / 2)
        return final_output
```

aae.py 的代码如下：

```
#coding:gbk
#摘自网页 https://blog.csdn.net/jing_zhong/article/details/123058344
from __future__ import print_function, division
from MergeLayer import MergeLayer

from keras.layers import Input, Dense, Reshape, Flatten, Dropout, multiply, Gaussia
nNoise
from keras.layers import LeakyReLU
from keras.models import Sequential, Model, load_model
from keras.optimizers import Adam
from PIL import Image
import matplotlib.pyplot as plt
import glob
import numpy as np
import os

class AdversarialAutoencoder():
    def __init__(self):
        self.img_rows = 128
        self.img_cols = 128
        self.channels = 1
        self.img_shape = (self.img_rows, self.img_cols, self.channels)
        self.latent_dim = 8

        optimizer = Adam(0.0001, 0.5)

        # Build and compile the discriminator
        # Build the encoder / decoder

        self.encoder = self.build_encoder()
        self.decoder = self.build_decoder()
        self.discriminator = self.build_discriminator()

        self.discriminator.compile(loss='binary_crossentropy',
```

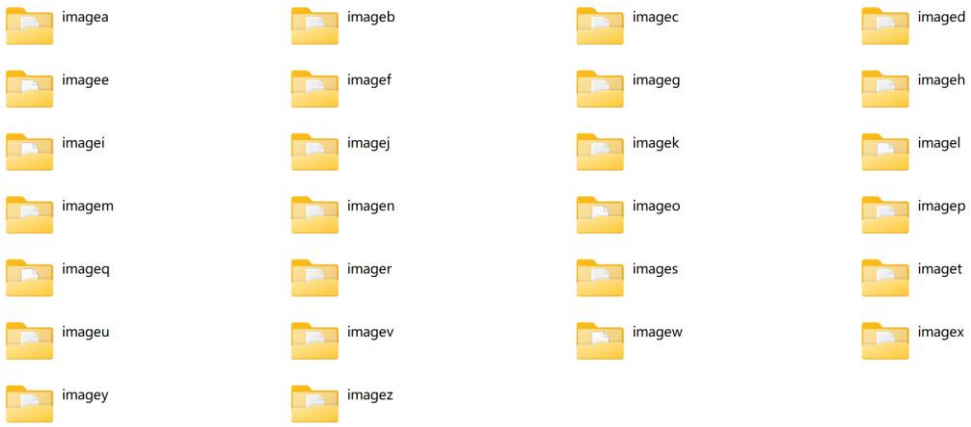
```
            optimizer=optimizer,
            metrics=['accuracy'])
        img = Input(shape=self.img_shape)
        # The generator takes the image, encodes it and reconstructs it
        # from the encoding
        encoded_repr = self.encoder(img)
        reconstructed_img = self.decoder(encoded_repr)

        # For the adversarial_autoencoder model we will only train the generator
        self.discriminator.trainable = False

        # The discriminator determines validity of the encoding
        validity = self.discriminator(encoded_repr)

        # The adversarial_autoencoder model  (stacked generator and discriminator)

        self.adversarial_autoencoder = Model(img, [reconstructed_img, validity])
        self.adversarial_autoencoder.compile(loss=['mse', 'binary_crossentropy'],
            optimizer=optimizer)


    def build_encoder(self):
        # Encoder

        img = Input(shape=self.img_shape)

        h = Flatten()(img)
        h = Dense(512)(h)
        h = LeakyReLU(alpha=0.2)(h)
        h = Dense(512)(h)
        h = LeakyReLU(alpha=0.2)(h)
        mu = Dense(self.latent_dim)(h)
        log_var = Dense(self.latent_dim)(h)

        latent_repr = MergeLayer()([mu, log_var])

        return Model(img, latent_repr)

    def build_decoder(self):

        model = Sequential()

        model.add(Dense(512, input_dim=self.latent_dim))
        model.add(LeakyReLU(alpha=0.2))
```

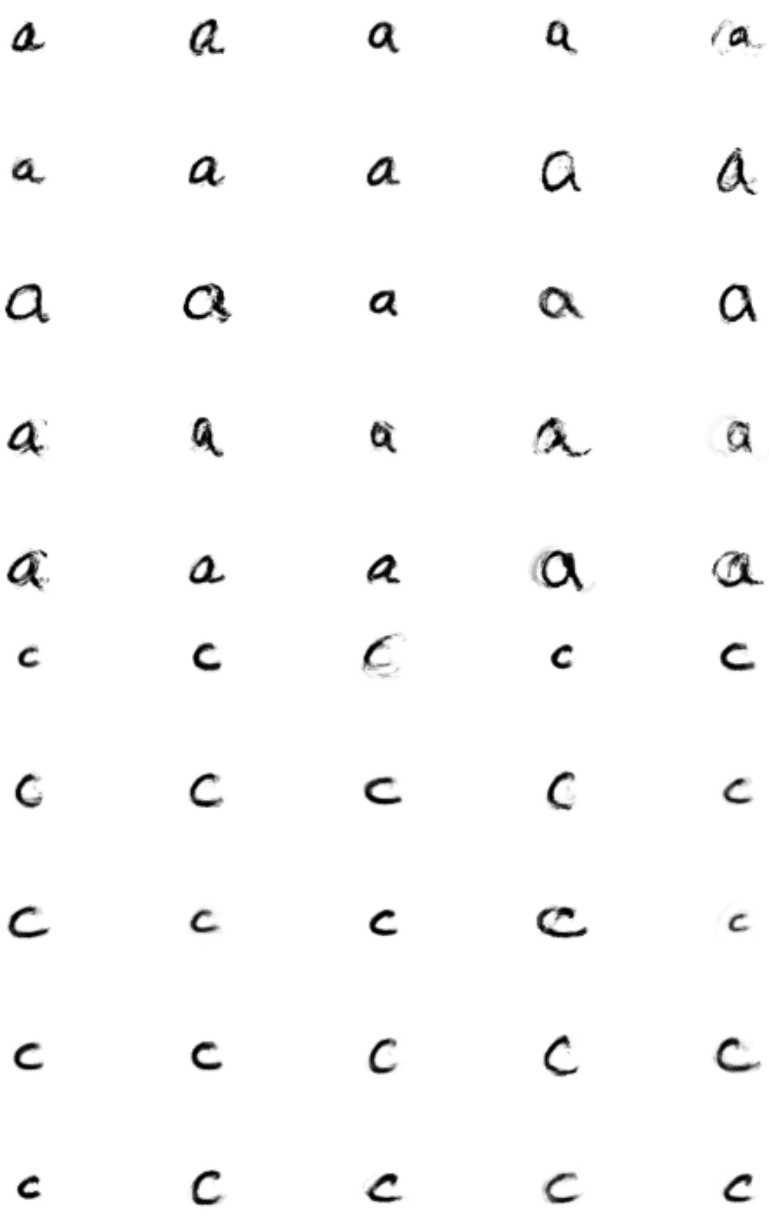
```
        model.add(Dense(512))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dense(np.prod(self.img_shape), activation='tanh'))
        model.add(Reshape(self.img_shape))

        z = Input(shape=(self.latent_dim,))
        img = model(z)

        return Model(z, img)

    def build_discriminator(self):

        model = Sequential()

        model.add(Dense(512, input_dim=self.latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dense(256))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dense(1, activation="sigmoid"))

        encoded_repr = Input(shape=(self.latent_dim, ))
        validity = model(encoded_repr)

        return Model(encoded_repr, validity)
    def load_data(self,sample):
        #dataset = glob.glob("data//*")
            datares = []
        #for i in dataset:
            for k in glob.glob("data//"+sample+"//*.png"):
                imge = Image.open(k,'r')
                imge = imge.convert('L')
                imge = np.asarray(imge)
                datares.append(imge)
            return np.array(datares)
    def train(self, epochs, batch_size=128, sample_interval=50,sample=""):
        folder = os.path.exists("image//image"+sample)
        if not folder:
            os.makedirs("image//image"+sample)
        # Load the dataset
        X_train= self.load_data(sample)

        # Rescale -1 to 1
        X_train = (X_train.astype(np.float32)) / 127.5-1
        X_train = np.expand_dims(X_train, axis=3)
```

```
        # Adversarial ground truths
        valid = np.ones((batch_size, 1))
        fake = np.zeros((batch_size, 1))

        for epoch in range(epochs):

            # ----------------------
            #  Train Discriminator
            # ----------------------

            # Select a random batch of images
            idx = np.random.randint(0, X_train.shape[0], batch_size)
            imgs = X_train[idx]

            latent_fake = self.encoder.predict(imgs,verbose=0)
            latent_real = np.random.normal(size=(batch_size, self.latent_dim))

            # Train the discriminator
            d_loss_real = self.discriminator.train_on_batch(latent_real, valid)
            d_loss_fake = self.discriminator.train_on_batch(latent_fake, fake)
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

            # ----------------------
            #  Train Generator
            # ----------------------

            # Train the generator
            g_loss = self.adversarial_autoencoder.train_on_batch(imgs, [imgs, valid
])

            # Plot the progress


            # If at save interval => save generated image samples
            if epoch % sample_interval == 0 and epoch>0:
                self.sample_images(epoch,sample)
                self.save_model(sample,str(epoch))
            if epoch %250 == 0:
                if epoch==0:
                    f = open("image//image"+sample+"//data.txt","w",encoding="gbk")

                    f.write("%d [D loss: %f, acc: %.2f%%] [G loss: %f, mse: %f]\n"
% (epoch, d_loss[0], 100*d_loss[1], g_loss[0], 100*g_loss[1]))
```

```
                    print ("%d [D loss: %f, acc: %.2f%%] [G loss: %f, mse: %f]" % (epoch, d_loss[0], 100*d_loss[1], g_loss[0], 100*g_loss[1]))
                    f.close()
                else:
                    f = open("image//image"+sample+"//data.txt","a+",encoding="gbk")
                    f.write("%d [D loss: %f, acc: %.2f%%] [G loss: %f, mse: %f]\n" % (epoch, d_loss[0], 100*d_loss[1], g_loss[0], 100*g_loss[1]))
                    print ("%d [D loss: %f, acc: %.2f%%] [G loss: %f, mse: %f]" % (epoch, d_loss[0], 100*d_loss[1], g_loss[0], 100*g_loss[1]))
                    f.close()

    def sample_images(self, epoch,sample):
        r, c = 5, 5

        z = np.random.normal(size=(r*c, self.latent_dim))
        gen_imgs = self.decoder.predict(z)

        gen_imgs = 0.5 * gen_imgs + 0.5

        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt, :,:,0], cmap='gray')
                axs[i,j].axis('off')
                cnt += 1
        fig.savefig("image//image"+sample+"//letter_"+sample+"%d.png" % epoch)
        plt.close()

    def save_model(self,sample,flag):
        self.decoder.save("image//image"+sample+"//aae_decoder"+flag+".h5")


if __name__ == '__main__':
    data = [chr(i) for i in range(ord("a"),ord("z")+1)]
    for i in data:
        aae = AdversarialAutoencoder()
        aae.train(epochs=20001, batch_size=64, sample_interval=1000,sample = i)
```

最后在 image 文件夹下方会生成虚假的 a-z 的小写英文字母：





可以看出，生成的结果是相当不错的。