CS156 (Introduction to AI), Spring 2022

Final Term Project

Roster Name: Austin Rivard

Preferred Name (if different):

Student ID: 015044445

Email address: austin.rivard@sjsu.edu

Project description/introduction text (the background information)

This project was created with the goal of generating new images of doodles from seven different classes: airplane, ant, axe, bear, bicycle, bird, and bread.

Doodles are a very interesting domain of images to work with in computer vision because they are the human brain's attempt to represent an actual object (airplane, bicycle, bread) to other humans using very few features. When tasked with quickly drawing a doodle of an object from memory, a person is forced to think about the most efficient way to convey meaning that another person would likely understand. For this reason, doodles typically include the most distinguishing features of an object and could be a sort of window into how the human brain represents real-life objects internally.

Intrigued by this idea, I wanted to investigate how a machine learning model could learn to doodle. I chose these categories because they are so different from one another that the model would need to learn to draw a wide variety of objects as opposed to if I were to, say, choose just cats and dogs, although that could be an interesting dataset to to use to see the distinguishing features that it learns from each.

Machine learning algorithm selected for this project

The model used is a conditional deep convolutional GAN, or cDCGAN, with some additional modifications like spectral normalization added to help convergence of the model and improve the final quality of generated images. It took me many iterations to find hyperparameters that wouldn't result in the network falling into a failure mode during training. The most common failure mode was characterized by discriminator loss going to zero and generator loss climbing to around ten shortly after training began. I performed many experiments by tweaking the learning rate, ADAM momentum, latent space dimensionality, batch size, and also added spectral normalization to both the generator and discriminator and added dropout to just the generator. The hyperparameters used below are what I found success with through experimentation, but there are likely many changes that could be made to speed up training, improve image quality, or both.

Dataset source

Quick, Draw! - Google

https://quickdraw.withgoogle.com/data

https://github.com/googlecreativelab/quickdraw-dataset

References and sources

Code examples used from:

Class reference notebook GAN.MNIST.ipynb for Dr. Yulia Newton's CS156 class

https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

https://github.com/soumith/ganhacks

https://arxiv.org/abs/1611.07004

https://arxiv.org/abs/1809.11096

https://arxiv.org/abs/1807.04720

https://arxiv.org/abs/1802.05957

Solution

Load libraries and set random number generator seed

In [1]:
```
import tensorflow as tf
import tensorflow_addons as tfa
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.utils import plot_model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import Model
```

In [2]:
```
np.random.seed(42)
```

In [3]:
```
from tensorflow.python.keras import backend as K
K._get_available_gpus()
```

Out[3]:
```
['/device:GPU:0']
```

Code the solution

In [4]:
```
ls data
```

```
 Volume in drive C is Windows
 Volume Serial Number is 9CA3-5D2E

 Directory of c:\Users\itsau\Code\School\CS156\Term Project\data

05/13/2022  06:46 PM    <DIR>          .
05/14/2022  07:52 PM    <DIR>          ..
05/13/2022  06:44 PM       118,872,512 full_numpy_bitmap_airplane.npy
05/13/2022  06:44 PM        97,695,888 full_numpy_bitmap_ant.npy
05/13/2022  06:44 PM        97,311,728 full_numpy_bitmap_axe.npy
05/13/2022  06:44 PM       105,653,488 full_numpy_bitmap_bear.npy
05/13/2022  06:45 PM        99,197,248 full_numpy_bitmap_bicycle.npy
05/13/2022  06:45 PM       104,720,528 full_numpy_bitmap_bird.npy
05/13/2022  06:45 PM        94,526,960 full_numpy_bitmap_bread.npy
               7 File(s)    717,978,352 bytes
               2 Dir(s)  533,265,866,752 bytes free
```

In [5]:

```
LR = 0.0002
beta_1 = 0.5
beta_2 = 0.999

def define_discriminator(in_shape=(28,28,1), n_classes=7):
        # label input
        in_label = Input(shape=(1,))
        # embedding for categorical input
        li = Embedding(n_classes, 50)(in_label)
        # scale up to image dimensions with linear activation
        n_nodes = in_shape[0] * in_shape[1]
        li = Dense(n_nodes)(li)
        # reshape to additional channel
        li = Reshape((in_shape[0], in_shape[1], 1))(li)
        # image input
        in_image = Input(shape=in_shape)
        # concat label as a channel
        merge = Concatenate()([in_image, li])
        # downsample
        fe = Conv2D(128, (3,3), strides=(2,2), padding='same')(merge)
        fe = LeakyReLU(alpha=0.2)(fe)
        # downsample
        fe = tfa.layers.SpectralNormalization(Conv2D(128, (3,3), strides=(2,2), padding='same'))(fe)
        fe = LeakyReLU(alpha=0.2)(fe)
        # flatten feature maps
        fe = Flatten()(fe)
        # dropout
        fe = Dropout(0.4)(fe)
        # output
        out_layer = Dense(1, activation='sigmoid')(fe)
        # define model
        model = Model([in_image, in_label], out_layer)
        # compile model
        opt = Adam(lr=LR, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
        return model

# define the standalone generator model
def define_generator(latent_dim, n_classes=7):
        # label input
        in_label = Input(shape=(1,))
        # embedding for categorical input
        li = Embedding(n_classes, 50)(in_label)
        # linear multiplication
        n_nodes = 7 * 7
        li = Dense(n_nodes)(li)
        # reshape to additional channel
        li = Reshape((7, 7, 1))(li)
        # image generator input
        in_lat = Input(shape=(latent_dim,))
        # foundation for 7x7 image
        n_nodes = 128 * 7 * 7
        gen = Dense(n_nodes)(in_lat)
        gen = LeakyReLU(alpha=0.2)(gen)
        gen = Reshape((7, 7, 128))(gen)
        # merge image gen and label input
        merge = Concatenate()([gen, li])
        # upsample to 14x14
        gen = Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(merge)
        gen = Dropout(0.5)(gen)
        gen = LeakyReLU(alpha=0.2)(gen)
        # upsample to 28x28
        gen = tfa.layers.SpectralNormalization(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))(gen)
        gen = LeakyReLU(alpha=0.2)(gen)
        # output
        out_layer = Conv2D(1, (7,7), activation='tanh', padding='same')(gen)
        # define model
        model = Model([in_lat, in_label], out_layer)
        return model

# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
        # make weights in the discriminator not trainable
        d_model.trainable = False
        # get noise and label inputs from generator model
        gen_noise, gen_label = g_model.input
        # get image output from the generator model
        gen_output = g_model.output
        # connect image output and label input from generator as inputs to discriminator
        gan_output = d_model([gen_output, gen_label])
        # define gan model as taking noise and label and outputting a classification
        model = Model([gen_noise, gen_label], gan_output)
        # compile model
        opt = Adam(lr=LR, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt)
        return model

import os
label_map = {
    'airplane': 0,
    'ant': 1,
    'axe': 2,
    'bear': 3,
    'bicycle': 4,
    'bird': 5,
    'bread': 6
}
```

```
def load_real_samples(dir='data'):
    X, Y = [], []
    for filename in os.listdir(dir):
        label = filename.split('_')[-1].split('.')[0]
        data = np.load(dir + '/' + filename)
        data = data.reshape(data.shape[0], 28, 28, 1)
        y = np.full(data.shape[0], label_map[label])
        X.append(data)
        Y.append(y)
    X = np.concatenate(X, axis=0)
    # X = np.expand_dims(X, axis=-1)
    X = X.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5
    Y = np.concatenate(Y, axis=0)
    # synchronously shuffle X and Y
    randomize = np.arange(len(X))
    np.random.shuffle(randomize)
    X = X[randomize]
    Y = Y[randomize]
    return [X, Y]

# select real samples
def generate_real_samples(dataset, n_samples):
        # split into images and labels
        images, labels = dataset
        # choose random instances
        ix = np.random.randint(0, images.shape[0], n_samples)
        # select images and labels
        X, labels = images[ix], labels[ix]
        # generate class labels
        y = np.ones((n_samples, 1))
        return [X, labels], y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples, n_classes=7):
        # generate points in the latent space
        x_input = np.random.randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
        z_input = x_input.reshape(n_samples, latent_dim)
        # generate labels
        labels = np.random.randint(0, n_classes, n_samples)
        return [z_input, labels]

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n_samples):
        # generate points in latent space
        z_input, labels_input = generate_latent_points(latent_dim, n_samples)
        # predict outputs
        images = generator.predict([z_input, labels_input])
        # create class labels
        y = np.zeros((n_samples, 1))
        return [images, labels_input], y

# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, latent_dim, n_samples=70):
        # prepare fake examples
        latent_points, labels = generate_latent_points(latent_dim, n_samples)
        # specify labels
        labels = np.asarray([x for _ in range(10) for x in range(7)])
        # generate images
        X  = g_model.predict([latent_points, labels])
        # scale from [-1,1] to [0,1]
        X = (X + 1) / 2.0
        # plot images
        for i in range(7 * 10):
                # define subplot
                plt.subplot(10, 7, 1 + i)
                # turn off axis
                plt.axis('off')
                # plot raw pixel data
                plt.imshow(X[i, :, :, 0], cmap='gray_r')
        # save plot to file
        plt.savefig('results_baseline/generated_plot_%03d.png' % (step+1))
        plt.close()
        # save the generator model
        g_model.save('results_baseline/model_%03d.h5' % (step+1))

# create a line plot of loss for the gan and save to file
def plot_history(d1_hist, d2_hist, g_hist, a1_hist, a2_hist):
        # plot loss
        plt.subplot(2, 1, 1)
        plt.plot(d1_hist, label='d-real')
        plt.plot(d2_hist, label='d-fake')
        plt.plot(g_hist, label='gen')
        plt.legend()
        # plot discriminator accuracy
        plt.subplot(2, 1, 2)
        plt.plot(a1_hist, label='acc-real')
        plt.plot(a2_hist, label='acc-fake')
        plt.legend()
        # save plot to file
        plt.savefig('results_baseline/plot_line_plot_loss.png')
        plt.close()

# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10, n_batch=128):
```

```
bat_per_epo = int(dataset[0].shape[0] / n_batch)
half_batch = int(n_batch / 2)
d1_hist, d2_hist, g_hist, a1_hist, a2_hist = list(), list(), list(), list(), list()
# manually enumerate epochs
for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
                # get randomly selected 'real' samples
                [X_real, labels_real], y_real = generate_real_samples(dataset, half_batch)
                # update discriminator model weights
                d_loss1, d_acc1 = d_model.train_on_batch([X_real, labels_real], y_real)
                # generate 'fake' examples
                [X_fake, labels], y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
                # update discriminator model weights
                d_loss2, d_acc2 = d_model.train_on_batch([X_fake, labels], y_fake)
                # prepare points in latent space as input for the generator
                [z_input, labels_input] = generate_latent_points(latent_dim, n_batch)
                # create inverted labels for the fake samples
                y_gan = np.ones((n_batch, 1))
                # update the generator via the discriminator's error
                g_loss = gan_model.train_on_batch([z_input, labels_input], y_gan)
                # summarize loss on this batch
                print('>%d, %d/%d, d1=%.3f, d2=%.3f, g=%.3f, a1=%d, a2=%d' %
                        (i+1, j+1, bat_per_epo, d_loss1, d_loss2, g_loss, int(100*d_acc1), int(100*d_acc2)))
                                # record history
                d1_hist.append(d_loss1)
                d2_hist.append(d_loss2)
                g_hist.append(g_loss)
                a1_hist.append(d_acc1)
                a2_hist.append(d_acc2)
        # evaluate the model performance every epoch
        summarize_performance(i, g_model, latent_dim)
plot_history(d1_hist, d2_hist, g_hist, a1_hist, a2_hist)
        # save the generator model
        g_model.save('cgan_generator.h5')
```

In [6]:

```
# size of the latent space
latent_dim = 128
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
```

```
c:\Users\itsau\anaconda3\envs\tf_gpu\lib\site-packages\keras\optimizer_v2\optimizer_v2.py:355: UserWarning: The `lr` argument is deprecate
d, use `learning_rate` instead.
  warnings.warn(
```

In [7]:

```
# train model
train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10, n_batch=2048)
```

```
>1, 1/447, d1=0.709, d2=0.695, g=0.691, a1=36, a2=1
>1, 2/447, d1=0.615, d2=0.701, g=0.685, a1=98, a2=0
>1, 3/447, d1=0.542, d2=0.710, g=0.674, a1=100, a2=0
>1, 4/447, d1=0.477, d2=0.729, g=0.656, a1=100, a2=0
>1, 5/447, d1=0.419, d2=0.759, g=0.629, a1=100, a2=0
>1, 6/447, d1=0.366, d2=0.805, g=0.594, a1=100, a2=0
>1, 7/447, d1=0.323, d2=0.865, g=0.558, a1=100, a2=0
>1, 8/447, d1=0.303, d2=0.923, g=0.534, a1=100, a2=0
>1, 9/447, d1=0.298, d2=0.954, g=0.534, a1=100, a2=0
>1, 10/447, d1=0.309, d2=0.936, g=0.570, a1=100, a2=0
>1, 11/447, d1=0.330, d2=0.862, g=0.634, a1=100, a2=0
>1, 12/447, d1=0.360, d2=0.778, g=0.721, a1=100, a2=4
>1, 13/447, d1=0.388, d2=0.681, g=0.812, a1=99, a2=60
>1, 14/447, d1=0.408, d2=0.600, g=0.905, a1=99, a2=96
>1, 15/447, d1=0.429, d2=0.540, g=0.963, a1=98, a2=100
>1, 16/447, d1=0.450, d2=0.514, g=0.981, a1=96, a2=100
>1, 17/447, d1=0.467, d2=0.505, g=1.002, a1=95, a2=100
>1, 18/447, d1=0.469, d2=0.503, g=0.999, a1=94, a2=100
>1, 19/447, d1=0.451, d2=0.533, g=0.956, a1=96, a2=100
>1, 20/447, d1=0.453, d2=0.606, g=0.860, a1=95, a2=100
>1, 21/447, d1=0.447, d2=0.719, g=0.765, a1=93, a2=5
>1, 22/447, d1=0.425, d2=0.823, g=0.701, a1=95, a2=0
>1, 23/447, d1=0.392, d2=0.800, g=0.729, a1=98, a2=0
>1, 24/447, d1=0.343, d2=0.688, g=0.831, a1=99, a2=62
>1, 25/447, d1=0.299, d2=0.585, g=0.934, a1=100, a2=100
>1, 26/447, d1=0.248, d2=0.537, g=0.980, a1=100, a2=100
>1, 27/447, d1=0.200, d2=0.538, g=0.962, a1=100, a2=100
>1, 28/447, d1=0.154, d2=0.564, g=0.905, a1=100, a2=100
>1, 29/447, d1=0.120, d2=0.598, g=0.837, a1=100, a2=100
>1, 30/447, d1=0.093, d2=0.634, g=0.779, a1=100, a2=100
>1, 31/447, d1=0.075, d2=0.667, g=0.731, a1=100, a2=100
>1, 32/447, d1=0.063, d2=0.712, g=0.679, a1=100, a2=0
>1, 33/447, d1=0.052, d2=0.779, g=0.616, a1=100, a2=0
>1, 34/447, d1=0.047, d2=0.865, g=0.555, a1=100, a2=0
>1, 35/447, d1=0.046, d2=0.970, g=0.496, a1=100, a2=0
>1, 36/447, d1=0.048, d2=1.089, g=0.447, a1=100, a2=0
>1, 37/447, d1=0.056, d2=1.183, g=0.422, a1=100, a2=0
>1, 38/447, d1=0.072, d2=1.200, g=0.434, a1=100, a2=0
>1, 39/447, d1=0.096, d2=1.135, g=0.482, a1=100, a2=0
>1, 40/447, d1=0.133, d2=1.007, g=0.562, a1=100, a2=0
>1, 41/447, d1=0.184, d2=0.867, g=0.658, a1=100, a2=0
```

```
>1, 42/447, d1=0.229, d2=0.746, g=0.756, a1=100, a2=9
>1, 43/447, d1=0.266, d2=0.652, g=0.854, a1=99, a2=82
>1, 44/447, d1=0.300, d2=0.574, g=0.939, a1=99, a2=99
>1, 45/447, d1=0.321, d2=0.524, g=1.009, a1=98, a2=100
>1, 46/447, d1=0.338, d2=0.488, g=1.070, a1=98, a2=100
>1, 47/447, d1=0.344, d2=0.461, g=1.121, a1=97, a2=100
>1, 48/447, d1=0.351, d2=0.439, g=1.166, a1=97, a2=100
>1, 49/447, d1=0.359, d2=0.420, g=1.196, a1=96, a2=100
>1, 50/447, d1=0.367, d2=0.414, g=1.205, a1=95, a2=100
>1, 51/447, d1=0.369, d2=0.412, g=1.203, a1=94, a2=100
>1, 52/447, d1=0.376, d2=0.427, g=1.171, a1=94, a2=100
>1, 53/447, d1=0.372, d2=0.454, g=1.104, a1=94, a2=100
>1, 54/447, d1=0.362, d2=0.501, g=1.011, a1=95, a2=100
>1, 55/447, d1=0.372, d2=0.566, g=0.906, a1=94, a2=99
>1, 56/447, d1=0.350, d2=0.630, g=0.816, a1=95, a2=96
>1, 57/447, d1=0.343, d2=0.672, g=0.765, a1=95, a2=78
>1, 58/447, d1=0.322, d2=0.685, g=0.752, a1=97, a2=64
>1, 59/447, d1=0.293, d2=0.673, g=0.760, a1=98, a2=84
>1, 60/447, d1=0.262, d2=0.650, g=0.781, a1=98, a2=98
>1, 61/447, d1=0.231, d2=0.624, g=0.810, a1=99, a2=100
>1, 62/447, d1=0.211, d2=0.602, g=0.830, a1=99, a2=100
>1, 63/447, d1=0.180, d2=0.596, g=0.833, a1=100, a2=100
>1, 64/447, d1=0.157, d2=0.604, g=0.816, a1=100, a2=100
>1, 65/447, d1=0.137, d2=0.628, g=0.785, a1=100, a2=100
>1, 66/447, d1=0.125, d2=0.666, g=0.745, a1=100, a2=93
>1, 67/447, d1=0.117, d2=0.719, g=0.700, a1=100, a2=8
>1, 68/447, d1=0.120, d2=0.784, g=0.660, a1=100, a2=0
>1, 69/447, d1=0.128, d2=0.846, g=0.641, a1=100, a2=0
>1, 70/447, d1=0.152, d2=0.878, g=0.652, a1=100, a2=0
>1, 71/447, d1=0.190, d2=0.856, g=0.700, a1=100, a2=0
>1, 72/447, d1=0.248, d2=0.784, g=0.771, a1=100, a2=0
>1, 73/447, d1=0.281, d2=0.699, g=0.853, a1=100, a2=43
>1, 74/447, d1=0.290, d2=0.608, g=0.939, a1=100, a2=100
>1, 75/447, d1=0.269, d2=0.536, g=1.001, a1=100, a2=100
>1, 76/447, d1=0.228, d2=0.498, g=1.033, a1=100, a2=100
>1, 77/447, d1=0.187, d2=0.495, g=1.027, a1=100, a2=100
>1, 78/447, d1=0.151, d2=0.519, g=0.987, a1=100, a2=100
>1, 79/447, d1=0.129, d2=0.568, g=0.923, a1=100, a2=100
>1, 80/447, d1=0.122, d2=0.637, g=0.860, a1=100, a2=95
>1, 81/447, d1=0.129, d2=0.700, g=0.834, a1=100, a2=44
>1, 82/447, d1=0.153, d2=0.722, g=0.863, a1=100, a2=29
>1, 83/447, d1=0.209, d2=0.702, g=0.928, a1=100, a2=44
>1, 84/447, d1=0.278, d2=0.683, g=0.998, a1=100, a2=60
>1, 85/447, d1=0.360, d2=0.674, g=1.039, a1=100, a2=66
>1, 86/447, d1=0.425, d2=0.654, g=1.077, a1=99, a2=79
>1, 87/447, d1=0.450, d2=0.597, g=1.133, a1=98, a2=98
>1, 88/447, d1=0.448, d2=0.514, g=1.205, a1=97, a2=100
>1, 89/447, d1=0.417, d2=0.441, g=1.282, a1=98, a2=100
>1, 90/447, d1=0.357, d2=0.392, g=1.347, a1=99, a2=100
>1, 91/447, d1=0.294, d2=0.373, g=1.389, a1=100, a2=100
>1, 92/447, d1=0.272, d2=0.375, g=1.384, a1=100, a2=100
>1, 93/447, d1=0.264, d2=0.405, g=1.342, a1=100, a2=100
>1, 94/447, d1=0.288, d2=0.435, g=1.300, a1=100, a2=100
>1, 95/447, d1=0.314, d2=0.442, g=1.274, a1=100, a2=100
>1, 96/447, d1=0.350, d2=0.447, g=1.233, a1=98, a2=100
>1, 97/447, d1=0.372, d2=0.463, g=1.174, a1=97, a2=100
>1, 98/447, d1=0.352, d2=0.478, g=1.123, a1=98, a2=100
>1, 99/447, d1=0.346, d2=0.494, g=1.075, a1=98, a2=100
>1, 100/447, d1=0.325, d2=0.508, g=1.035, a1=98, a2=100
>1, 101/447, d1=0.287, d2=0.514, g=1.019, a1=99, a2=100
>1, 102/447, d1=0.262, d2=0.514, g=1.015, a1=99, a2=100
>1, 103/447, d1=0.259, d2=0.516, g=1.003, a1=99, a2=100
>1, 104/447, d1=0.243, d2=0.522, g=1.000, a1=99, a2=100
>1, 105/447, d1=0.222, d2=0.517, g=1.014, a1=100, a2=100
>1, 106/447, d1=0.202, d2=0.500, g=1.045, a1=100, a2=100
>1, 107/447, d1=0.180, d2=0.469, g=1.094, a1=100, a2=100
>1, 108/447, d1=0.156, d2=0.434, g=1.147, a1=100, a2=100
>1, 109/447, d1=0.135, d2=0.412, g=1.181, a1=100, a2=100
>1, 110/447, d1=0.110, d2=0.412, g=1.177, a1=100, a2=100
>1, 111/447, d1=0.091, d2=0.437, g=1.135, a1=100, a2=100
>1, 112/447, d1=0.074, d2=0.476, g=1.092, a1=100, a2=100
>1, 113/447, d1=0.059, d2=0.497, g=1.099, a1=100, a2=99
>1, 114/447, d1=0.049, d2=0.462, g=1.178, a1=100, a2=99
>1, 115/447, d1=0.042, d2=0.404, g=1.275, a1=100, a2=99
>1, 116/447, d1=0.039, d2=0.395, g=1.277, a1=100, a2=100
>1, 117/447, d1=0.040, d2=0.457, g=1.166, a1=99, a2=98
>1, 118/447, d1=0.047, d2=0.614, g=0.983, a1=99, a2=77
>1, 119/447, d1=0.064, d2=0.855, g=0.866, a1=99, a2=22
>1, 120/447, d1=0.126, d2=1.054, g=0.979, a1=99, a2=5
>1, 121/447, d1=0.373, d2=0.900, g=1.307, a1=97, a2=13
>1, 122/447, d1=0.791, d2=0.661, g=1.437, a1=30, a2=60
>1, 123/447, d1=0.856, d2=0.548, g=1.444, a1=19, a2=96
>1, 124/447, d1=0.737, d2=0.475, g=1.444, a1=43, a2=99
>1, 125/447, d1=0.650, d2=0.483, g=1.358, a1=61, a2=95
>1, 126/447, d1=0.588, d2=0.549, g=1.237, a1=71, a2=85
>1, 127/447, d1=0.554, d2=0.565, g=1.188, a1=76, a2=85
>1, 128/447, d1=0.537, d2=0.509, g=1.246, a1=78, a2=99
>1, 129/447, d1=0.484, d2=0.418, g=1.350, a1=83, a2=100
>1, 130/447, d1=0.417, d2=0.370, g=1.429, a1=90, a2=100
>1, 131/447, d1=0.356, d2=0.380, g=1.422, a1=94, a2=99
>1, 132/447, d1=0.309, d2=0.452, g=1.349, a1=97, a2=93
>1, 133/447, d1=0.299, d2=0.486, g=1.341, a1=98, a2=90
>1, 134/447, d1=0.302, d2=0.421, g=1.409, a1=99, a2=100
>1, 135/447, d1=0.296, d2=0.382, g=1.414, a1=98, a2=100
>1, 136/447, d1=0.291, d2=0.430, g=1.319, a1=98, a2=100
>1, 137/447, d1=0.268, d2=0.604, g=1.146, a1=99, a2=76
>1, 138/447, d1=0.266, d2=0.646, g=1.220, a1=98, a2=65
```

```
>10, 366/447, d1=0.551, d2=0.559, g=1.143, a1=75, a2=77
>10, 367/447, d1=0.551, d2=0.594, g=1.126, a1=74, a2=74
>10, 368/447, d1=0.561, d2=0.513, g=1.240, a1=75, a2=89
>10, 369/447, d1=0.556, d2=0.496, g=1.218, a1=74, a2=88
>10, 370/447, d1=0.558, d2=0.583, g=1.082, a1=72, a2=76
>10, 371/447, d1=0.550, d2=0.595, g=1.199, a1=74, a2=75
>10, 372/447, d1=0.566, d2=0.452, g=1.419, a1=75, a2=93
>10, 373/447, d1=0.598, d2=0.514, g=1.209, a1=69, a2=80
>10, 374/447, d1=0.530, d2=0.568, g=1.168, a1=76, a2=74
>10, 375/447, d1=0.538, d2=0.523, g=1.260, a1=76, a2=84
>10, 376/447, d1=0.549, d2=0.461, g=1.275, a1=74, a2=93
>10, 377/447, d1=0.549, d2=0.540, g=1.132, a1=73, a2=81
>10, 378/447, d1=0.529, d2=0.615, g=1.073, a1=76, a2=70
>10, 379/447, d1=0.532, d2=0.529, g=1.291, a1=78, a2=86
>10, 380/447, d1=0.542, d2=0.496, g=1.260, a1=76, a2=87
>10, 381/447, d1=0.550, d2=0.611, g=1.114, a1=76, a2=67
>10, 382/447, d1=0.546, d2=0.554, g=1.217, a1=75, a2=81
>10, 383/447, d1=0.542, d2=0.437, g=1.324, a1=77, a2=96
>10, 384/447, d1=0.565, d2=0.514, g=1.160, a1=72, a2=84
>10, 385/447, d1=0.516, d2=0.616, g=1.079, a1=77, a2=68
>10, 386/447, d1=0.524, d2=0.529, g=1.305, a1=78, a2=86
>10, 387/447, d1=0.539, d2=0.522, g=1.253, a1=76, a2=81
>10, 388/447, d1=0.563, d2=0.602, g=1.115, a1=72, a2=67
>10, 389/447, d1=0.550, d2=0.548, g=1.212, a1=76, a2=80
>10, 390/447, d1=0.559, d2=0.470, g=1.275, a1=74, a2=92
>10, 391/447, d1=0.565, d2=0.505, g=1.194, a1=70, a2=85
>10, 392/447, d1=0.543, d2=0.593, g=1.119, a1=75, a2=74
>10, 393/447, d1=0.527, d2=0.530, g=1.282, a1=79, a2=88
>10, 394/447, d1=0.565, d2=0.455, g=1.352, a1=73, a2=90
>10, 395/447, d1=0.546, d2=0.561, g=1.148, a1=74, a2=73
>10, 396/447, d1=0.528, d2=0.606, g=1.155, a1=76, a2=68
>10, 397/447, d1=0.528, d2=0.500, g=1.268, a1=78, a2=90
>10, 398/447, d1=0.557, d2=0.490, g=1.228, a1=73, a2=89
>10, 399/447, d1=0.544, d2=0.580, g=1.115, a1=74, a2=75
>10, 400/447, d1=0.544, d2=0.533, g=1.279, a1=75, a2=85
>10, 401/447, d1=0.530, d2=0.462, g=1.335, a1=80, a2=90
>10, 402/447, d1=0.538, d2=0.582, g=1.166, a1=76, a2=71
>10, 403/447, d1=0.533, d2=0.560, g=1.167, a1=75, a2=78
>10, 404/447, d1=0.533, d2=0.496, g=1.245, a1=80, a2=90
>10, 405/447, d1=0.554, d2=0.529, g=1.162, a1=72, a2=84
>10, 406/447, d1=0.550, d2=0.594, g=1.087, a1=75, a2=74
>10, 407/447, d1=0.540, d2=0.561, g=1.181, a1=75, a2=81
>10, 408/447, d1=0.575, d2=0.489, g=1.313, a1=73, a2=88
>10, 409/447, d1=0.567, d2=0.518, g=1.211, a1=71, a2=79
>10, 410/447, d1=0.537, d2=0.576, g=1.136, a1=75, a2=73
>10, 411/447, d1=0.538, d2=0.520, g=1.195, a1=77, a2=85
>10, 412/447, d1=0.535, d2=0.480, g=1.240, a1=76, a2=92
>10, 413/447, d1=0.516, d2=0.511, g=1.174, a1=78, a2=87
>10, 414/447, d1=0.538, d2=0.583, g=1.118, a1=76, a2=78
>10, 415/447, d1=0.522, d2=0.523, g=1.237, a1=80, a2=84
>10, 416/447, d1=0.549, d2=0.528, g=1.230, a1=74, a2=80
>10, 417/447, d1=0.523, d2=0.603, g=1.154, a1=76, a2=70
>10, 418/447, d1=0.567, d2=0.526, g=1.238, a1=73, a2=85
```

In [11]:

```
# train model a second time
train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=50, n_batch=2048)
```

```
>1, 1/447, d1=0.472, d2=0.490, g=1.437, a1=80, a2=84
>1, 2/447, d1=0.500, d2=0.530, g=1.406, a1=77, a2=80
>1, 3/447, d1=0.535, d2=0.556, g=1.415, a1=74, a2=76
>1, 4/447, d1=0.550, d2=0.475, g=1.526, a1=72, a2=86
>1, 5/447, d1=0.564, d2=0.423, g=1.566, a1=71, a2=92
>1, 6/447, d1=0.520, d2=0.453, g=1.481, a1=77, a2=89
>1, 7/447, d1=0.491, d2=0.468, g=1.383, a1=79, a2=88
>1, 8/447, d1=0.524, d2=0.513, g=1.293, a1=74, a2=80
>1, 9/447, d1=0.497, d2=0.545, g=1.334, a1=77, a2=78
>1, 10/447, d1=0.493, d2=0.495, g=1.431, a1=77, a2=83
>1, 11/447, d1=0.486, d2=0.471, g=1.421, a1=79, a2=87
>1, 12/447, d1=0.477, d2=0.493, g=1.404, a1=81, a2=82
>1, 13/447, d1=0.473, d2=0.517, g=1.382, a1=80, a2=81
>1, 14/447, d1=0.525, d2=0.509, g=1.388, a1=75, a2=85
>1, 15/447, d1=0.543, d2=0.489, g=1.448, a1=72, a2=86
>1, 16/447, d1=0.560, d2=0.460, g=1.423, a1=70, a2=90
>1, 17/447, d1=0.532, d2=0.461, g=1.403, a1=76, a2=90
>1, 18/447, d1=0.497, d2=0.472, g=1.378, a1=79, a2=85
>1, 19/447, d1=0.530, d2=0.510, g=1.343, a1=77, a2=81
>1, 20/447, d1=0.515, d2=0.482, g=1.351, a1=76, a2=87
>1, 21/447, d1=0.490, d2=0.465, g=1.379, a1=79, a2=87
>1, 22/447, d1=0.465, d2=0.519, g=1.335, a1=81, a2=79
>1, 23/447, d1=0.480, d2=0.530, g=1.286, a1=79, a2=79
>1, 24/447, d1=0.513, d2=0.553, g=1.385, a1=74, a2=77
>1, 25/447, d1=0.531, d2=0.493, g=1.501, a1=75, a2=83
>1, 26/447, d1=0.559, d2=0.452, g=1.574, a1=71, a2=90
>1, 27/447, d1=0.535, d2=0.461, g=1.472, a1=76, a2=87
>1, 28/447, d1=0.502, d2=0.474, g=1.361, a1=78, a2=85
>1, 29/447, d1=0.513, d2=0.558, g=1.383, a1=75, a2=76
>1, 30/447, d1=0.529, d2=0.597, g=1.426, a1=74, a2=68
>1, 31/447, d1=0.568, d2=0.532, g=1.623, a1=70, a2=78
>1, 32/447, d1=0.551, d2=0.534, g=1.551, a1=73, a2=75
>1, 33/447, d1=0.522, d2=0.475, g=1.408, a1=73, a2=84
>1, 34/447, d1=0.510, d2=0.568, g=1.461, a1=77, a2=74
>1, 35/447, d1=0.549, d2=0.607, g=1.542, a1=71, a2=66
>1, 36/447, d1=0.579, d2=0.468, g=1.665, a1=68, a2=85
>1, 37/447, d1=0.561, d2=0.388, g=1.660, a1=69, a2=92
>1, 38/447, d1=0.511, d2=0.496, g=1.489, a1=76, a2=82
>1, 39/447, d1=0.516, d2=0.516, g=1.425, a1=75, a2=78
>1, 40/447, d1=0.488, d2=0.506, g=1.428, a1=77, a2=78
```

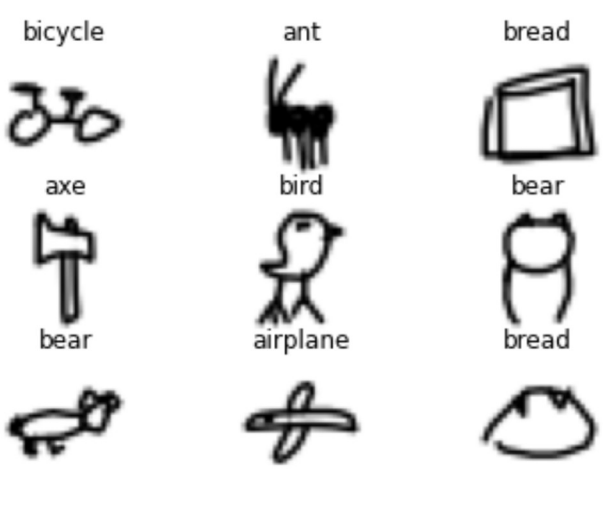
```
>50, 253/447, d1=1.377, d2=1.867, g=7.682, a1=63, a2=66
>50, 254/447, d1=1.867, d2=8.592, g=3.746, a1=74, a2=14
>50, 255/447, d1=3.221, d2=0.957, g=12.496, a1=32, a2=52
>50, 256/447, d1=4.647, d2=0.002, g=10.605, a1=16, a2=100
>50, 257/447, d1=2.517, d2=1.268, g=2.513, a1=49, a2=58
>50, 258/447, d1=0.711, d2=2.167, g=3.311, a1=74, a2=28
>50, 259/447, d1=0.879, d2=0.105, g=4.090, a1=56, a2=97
>50, 260/447, d1=0.883, d2=0.282, g=2.594, a1=58, a2=90
>50, 261/447, d1=0.365, d2=0.403, g=2.761, a1=84, a2=82
>50, 262/447, d1=0.342, d2=0.250, g=2.937, a1=84, a2=92
>50, 263/447, d1=0.393, d2=0.278, g=2.778, a1=82, a2=90
>50, 264/447, d1=0.360, d2=0.309, g=2.661, a1=83, a2=90
>50, 265/447, d1=0.317, d2=0.293, g=2.690, a1=87, a2=91
>50, 266/447, d1=0.357, d2=0.321, g=2.573, a1=85, a2=89
>50, 267/447, d1=0.376, d2=0.368, g=2.622, a1=84, a2=86
>50, 268/447, d1=0.427, d2=0.336, g=2.514, a1=79, a2=89
>50, 269/447, d1=0.388, d2=0.356, g=2.579, a1=82, a2=87
>50, 270/447, d1=0.444, d2=0.356, g=2.478, a1=78, a2=87
>50, 271/447, d1=0.417, d2=0.366, g=2.440, a1=80, a2=87
>50, 272/447, d1=0.401, d2=0.372, g=2.519, a1=81, a2=85
>50, 273/447, d1=0.429, d2=0.357, g=2.359, a1=80, a2=86
>50, 274/447, d1=0.423, d2=0.492, g=2.260, a1=79, a2=78
>50, 275/447, d1=0.466, d2=0.535, g=2.233, a1=76, a2=75
>50, 276/447, d1=0.548, d2=0.531, g=2.098, a1=71, a2=73
>50, 277/447, d1=0.607, d2=0.586, g=2.072, a1=68, a2=68
>50, 278/447, d1=0.572, d2=0.473, g=2.271, a1=70, a2=77
>50, 279/447, d1=0.601, d2=0.422, g=2.297, a1=67, a2=83
>50, 280/447, d1=0.527, d2=0.574, g=1.832, a1=73, a2=71
>50, 281/447, d1=0.538, d2=0.612, g=1.955, a1=73, a2=67
>50, 282/447, d1=0.548, d2=0.398, g=2.528, a1=73, a2=84
>50, 283/447, d1=0.572, d2=0.343, g=2.270, a1=70, a2=91
>50, 284/447, d1=0.467, d2=0.416, g=2.022, a1=76, a2=84
>50, 285/447, d1=0.461, d2=0.537, g=1.816, a1=76, a2=72
```

In [8]:

```
(imgs, lbls), _ = generate_real_samples(dataset, 9)
```

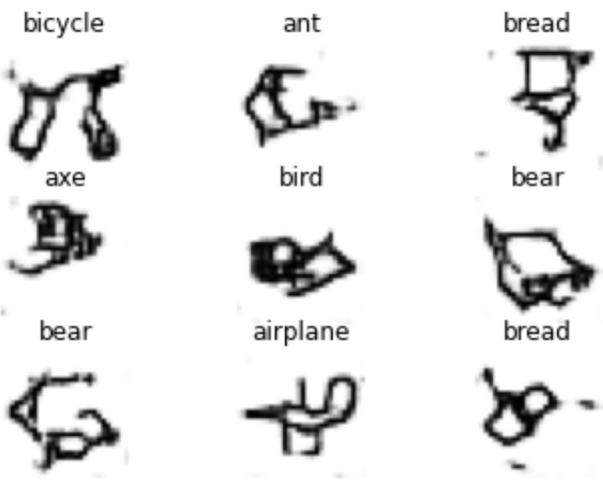
In [13]:

```
num_to_label = {v: k for k, v in label_map.items()}

# create and display a plot of generated images (reversed grayscale)
def display_plot(examples, labels, n):
    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')
        plt.title(num_to_label[labels[i]])
        plt.imshow(examples[i, :, :, 0], cmap='gray_r')
    plt.show()

display_plot(imgs, lbls, 3)
```



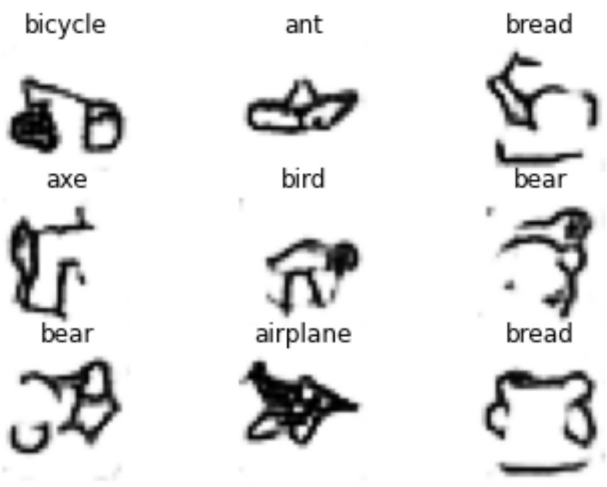
In [16]:

```
latent_points, _ = generate_latent_points(latent_dim, 9)
generated_imgs = g_model.predict([latent_points, lbls])
display_plot(generated_imgs, lbls, 3)
```



In [19]:

```
latent_points, _ = generate_latent_points(latent_dim, 9)
generated_imgs = g_model.predict([latent_points, lbls])
display_plot(generated_imgs, lbls, 3)
```


bicycle
ant
bread
axe
bird
bear
bear
airplane
bread