Machine Learning II Project

Generate Dog Images with Generative Adversarial Networks (GANs)

Xi Zhang

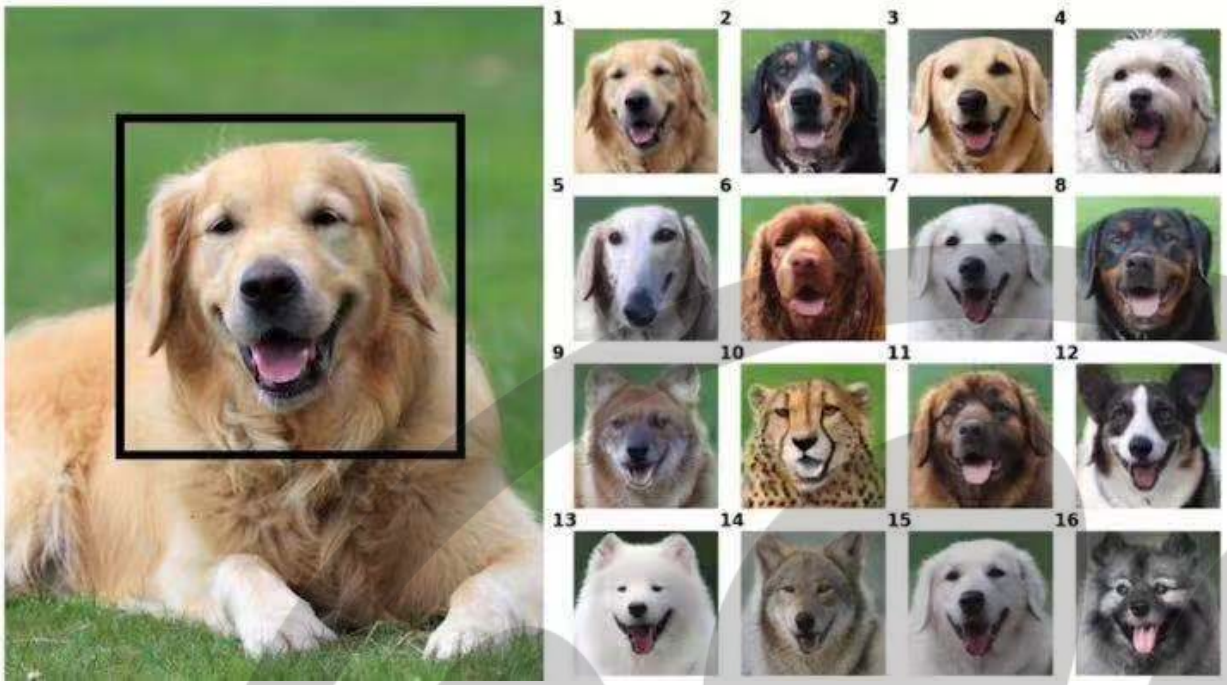
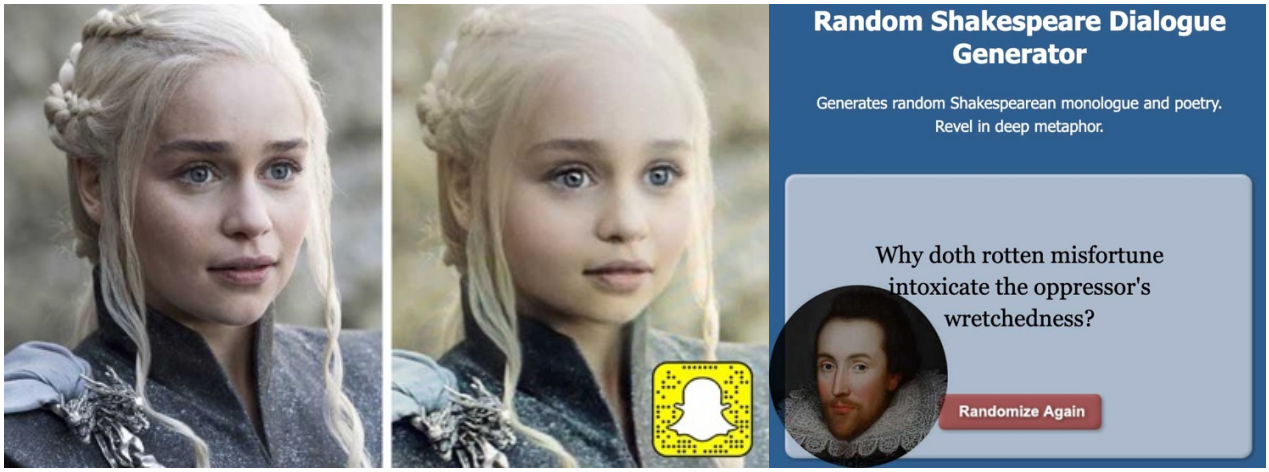


Table of Contents

1, Introduction

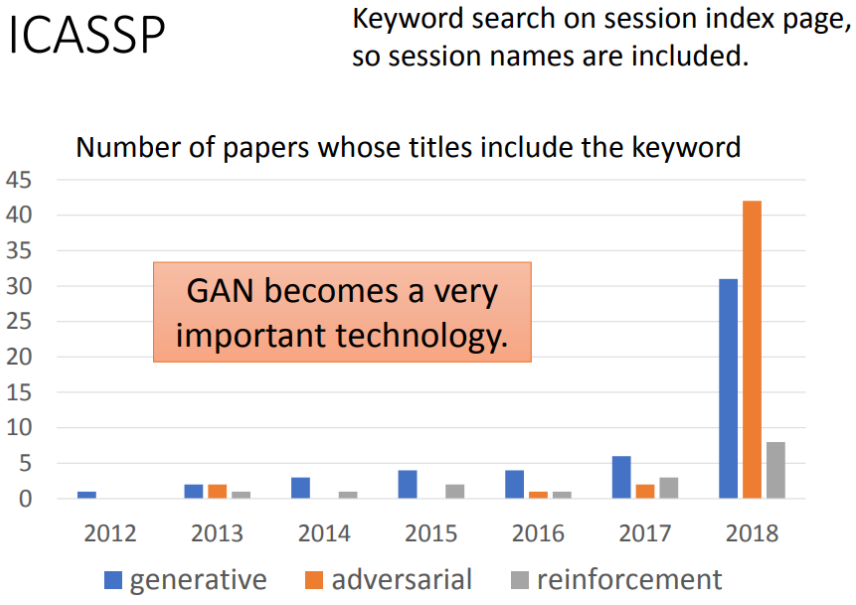
Combined with the knowledge of neural network learned in Machine Learning II, we hope to learn more in-depth and interesting knowledge on this basis. After investigation, Generative Adversarial Networks completely meets our requirements. As a new technology, GAN has been developing vigorously in recent two years. It is based on our familiar neural network such as multilayer perceptron (MLP) and convolutional neural network (CNN), but brings new vitality to machine learning.



Snapchat babyface filter and Shakespearean poetry generator, which only became popular in the last two years, also benefited from GAN's development.

1. 1, Background of GAN

Generative Adversarial Networks (GAN) is a cutting-edge technique of deep neural networks, which was first come up by Ian Goodfellow in 2014. In 2016, Yann LeCun, who is one of the leading scientists in AI, described GAN as “the coolest idea in machine learning in the last twenty years.”



GAN is a very new stuff and has a promising future. Especially in last year (2018), GAN was developed with an exponential increment. In other words, it is almost an infant technology. Although it is really new, there are bunch of models named with the suffix __GAN, such as Conditional GAN, DCGAN, Cycle GAN, Stack GAN. Fortunately, we can catch up the development of GAN now. Actually, we've learned every single component in GAN if I break down it.

1.2, Outline of Our Project

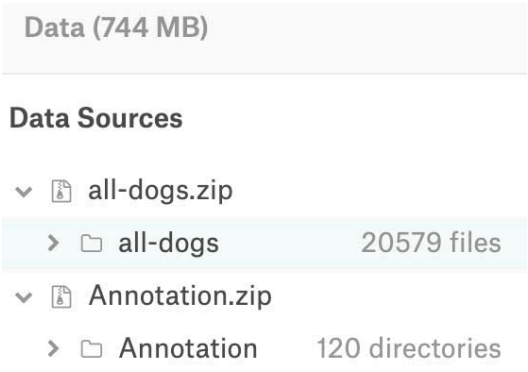
We will first introduce the data, then concepts, algorithms, and structures of our models. We hope to give you a macro understanding of what theories and materials we are based on to complete this project. Then we'll explain each step of implement, the problem we encountered, and how to solve it. Finally, we will explain the reasons for the deficiency and the ideas for future improvement.

2, Description of the Data Set

2.1, Resource

https://www.kaggle.com/c/generative-dog-images

Our data comes from the Kaggle code competition, which was released in June 2019. Since the game has ended in August, the ranking of the leaderboard also has certain reference value for our own model progress.



The open source data consists of the image archive and the annotation archive. After further study, we believe that the subfolder name of the picture package can fully perform the task of label, so our model only USES the picture compression package.

2.2, Overview of the Data Set

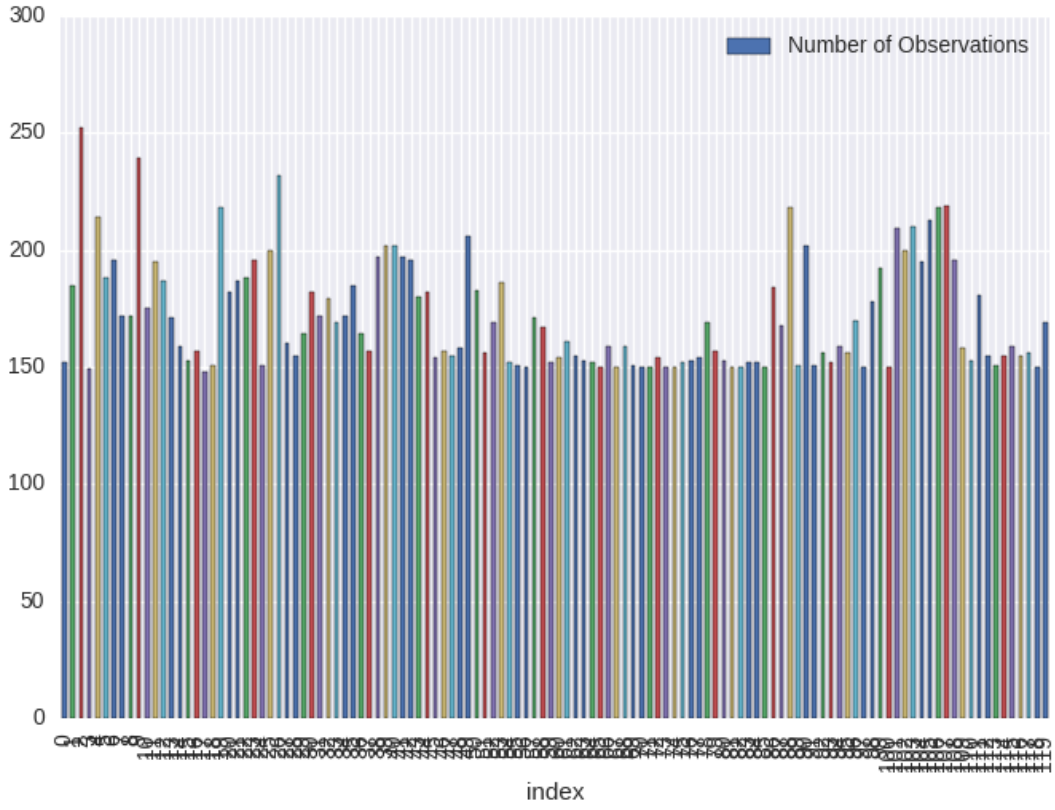
Image/label illustration; shape; balance; resized images

	Dog Breed	Number of Observations
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
.	.	.
.	.	.
.	.	.
116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150
119	n02116738-African_hunting_dog	169

Our dataset contains 20,579 images belonging to 120 dog breeds.

All images are colored with RGB channels, but the size is not fixed, and the proportion of dogs (main features) in the picture is not even. We can see that there are many pictures in which the complex background is the main body. And all we want to do is generate pictures of dogs. A large number of interference features will be the challenge of our study. While the amount of data is not too small, there are too many labels, and the data for each category is not significant when evenly distributed.

To visualize the data balance, we drew a histogram to show the data distribution.



You can see that in the distribution of 120 types of data, the maximum and minimum differences are generally within 100. We do not need to worry about over-sampling or under-sampling resulting in imbalanced data.

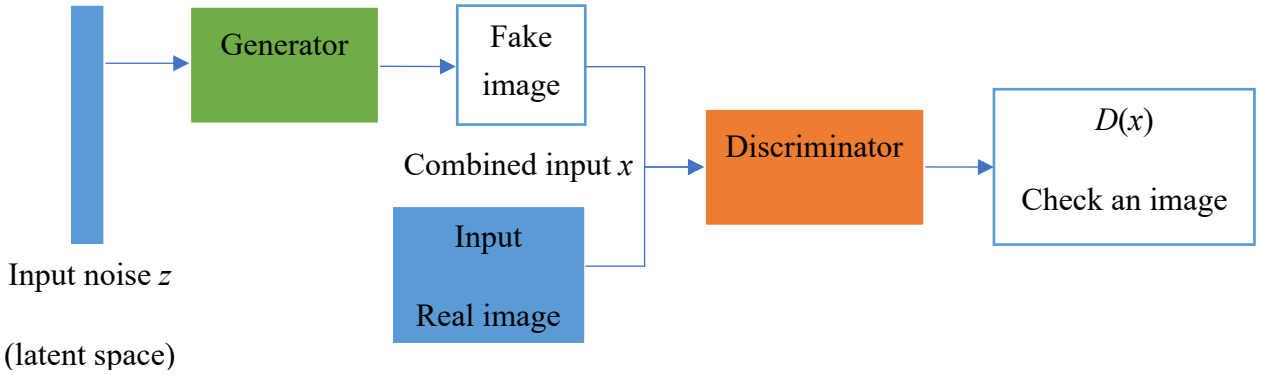
3, Description of Models

We implement GAN as our main network and try CGAN for promoting the results.

3.1, GAN

3.1.1, Structure of GAN

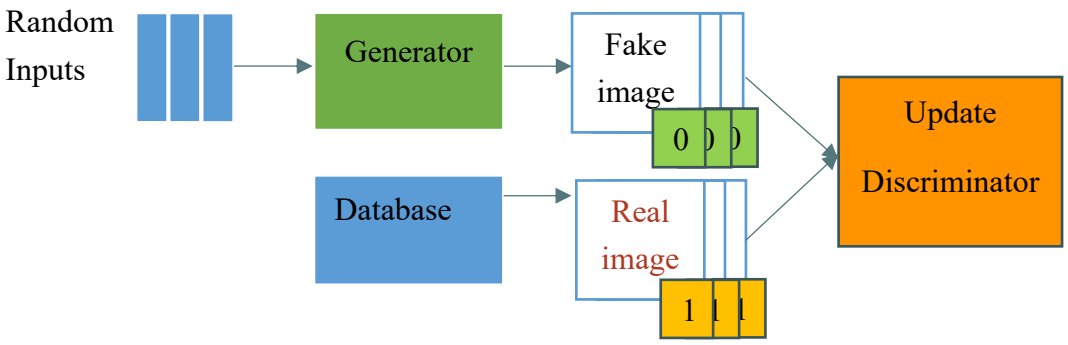
$$min_D max_G V(G,D) = \mathbb{E}_x \log\big(D(x)\big) + \mathbb{E}_z \log\Big(D\big(G(z)\big)\Big)$$



GAN is a minimax problem, which is one of zero-sum non-cooperative games. Generator wants to maximize its performance, which works to generate images as real as possible to confuse the Discriminator. Discriminator wants to distinguish a mixture of original and generated images whether real or fake. In this game, zero-sum means if Generator is improved, then there must be an increased loss of Discriminator. Our aim is to find the lowest aggregate loss of them, where there is a Nash Equilibrium.

3.1.2, Algorithm

- Initialize the Generator and Discriminator.
- Single train iteration:
    1. Fix the generator, then input the random vectors into the generator to get the generated images. In order to train the generator to generate the specific images we want; we need to pull the sample images from the database. Then update the discriminator.
    2. To update the discriminator, we have to adjust the parameter of it. For example, we can label the real images as 1, and generated images as 0. After updating, the discriminator is supposed to classify the real and fake images well. We can regard this problem as a classification problem and training the discriminator as training a classifier.



    3. After training discriminator, we fix it, and adjust the parameters of generator. The goal of generator training is to generate the image that discriminator can grade it close to 1. This step is similar to the optimizer in the neural network that we learned about, but it's a gradient ascent process.

    4. When we put these two processes together, it's a whole big network. As what we showed at the structure introduction. We put in the vector, and we end up with a number. But if we extract the output from the hidden layer in the middle, we get a complete image. Details will be shown in an implement below.

3.1.3, Algorithms for Mathematical Contents

Initialize parameter θ_d for discriminator and parameter θ_g for generator.

In single training iteration:

Training Discriminator:

- Sample n images from database: $\{\{x^1, x^2, \dots, x^n\}\}$. 'n' is the batch size in the model.
- Sample n noise samples from a distribution: $\{\{z^1, z^2, \dots, z^n\}\}$.
- Obtain generated data $\{\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^n\}\}$, $\tilde{x}^i = G(z^i)$.
- Update discriminator parameters θ_d

  $\tilde{V} = \frac{1}{n}\sum_{i=1}^{n} logD(x^i) + \frac{1}{n}\sum_{i=1}^{n} log(1 - D(\tilde{x}^i))$

  In order to maximize the objective function, we need to maximize the $D(x^i)$. It means we need to train the discriminator to give the real image a large value. Meanwhile, we want to minimize the $(D(\tilde{x}^i))$, which means we want the discriminator to give the fake image a small value.

  $\theta_d \leftarrow \theta_d + \eta\tilde{V}(\theta_d)$ * η: $Learning\ Rate$

  We use assent gradient to update the parameter of discriminator.

Training Generator:

- Sample n noise samples from a distribution: $\{\{z^1, z^2, \dots, z^n\}\}$.
- Update discriminator parameters θ_g

  $\tilde{V} = \frac{1}{n}\sum_{i=1}^{n} \log(D(G(z^i)))$

  When we input a random vector into the generator, it generates an image. This result will be graded by the trained discriminator. Our goal is to maximize the objective function.

  $\theta_g \leftarrow \theta_g + \eta\tilde{V}(\theta_g)$

  We use assent gradient to update the parameter of generator.

4, Results

4.1, Generated Images with Dogs

After completed the structure and decided the core of GAN, we still had a hard time on getting clear images. Therefore, we adjusted parameters and tried other labeling method and training methods on discriminator and generator. Unfortunately, we didn't make any improvement. Therefore, we decided to change the dataset for evaluating the model.

4.2, Generated Images with FashionMnist Dataset

This is the generated FashionMnist data by our trained GAN. It looks way better than the images above. Then we got confidence on keeping improving our model. Since the orginal images in FashionMnist are very simple in 1 channel, so we tried to fit the data like our dog dataset.

5, Summary and Conclusions

After this process, we believed the major portion of the bad result is due to our data with too many distractions. We thought we need to work on the images preprocessing. But pretraining was very difficult on our data. Therefore, we decided to keep trying the model itself to promote the result.

References

Avinash H. (2017). *The GAN Zoo*. GitHub. https://github.com/hindupuravinash/the-gan-zoo

Hongyi, L. (2018). *GAN Lecture 1: Introduction*. YouTube. https://www.youtube.com/watch?v=DQNNMiAP5lw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw&index=1

Jason B. (2019). *How to Develop a Conditional GAN (cGAN) From Scratch*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

Jonathan H. (2018). *GAN — Ways to improve GAN performance.* Towards Data Science. https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

Jonathan H. (2018). *GAN — CGAN & InfoGAN (using labels to improve GAN).* Medium. https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

Jonathan H. (2018). *GAN — Why it is so hard to train Generative Adversarial Networks!* Medium. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Jonathan H. (2018). *GAN — DCGAN (Deep convolutional generative adversarial networks)*

. Medium. https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

Jon G. (2019). *Conditional generative adversarial nets for convolutional face generation.* Stanford University.

Kaggle Competition. (2019). *Generative Dog Images*. Kaggle. https://www.kaggle.com/c/generative-dog-images

Naoki S. (2017). *Up-sampling with Transposed Convolution.* Medium. https://medium.com/activating-robotic-minds/up-sampling-with-transposed-convolution-9ae4f2df52d0

Utkarsh D. (2018). *Keep Calm and train a GAN. Pitfalls and Tips on training Generative Adversarial Networks*. Medium. https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9

Appendix

Data

	DOG BREED	NUMBER OF OBSERVATIONS
0	n02085620-Chihuahua	152
1	n02085782-Japanese_spaniel	185
2	n02085936-Maltese_dog	252
3	n02086079-Pekinese	149
4	n02086240-Shih-Tzu	214
5	n02086646-Blenheim_spaniel	188
6	n02086910-papillon	196
7	n02087046-toy_terrier	172
8	n02087394-Rhodesian_ridgeback	172
9	n02088094-Afghan_hound	239
10	n02088238-basset	175
11	n02088364-beagle	195
12	n02088466-bloodhound	187
13	n02088632-bluetick	171
14	n02089078-black-and-tan_coonhound	159
15	n02089867-Walker_hound	153
16	n02089973-English_foxhound	157
17	n02090379-redbone	148
18	n02090622-borzoi	151
19	n02090721-Irish_wolfhound	218
20	n02091032-Italian_greyhound	182
21	n02091134-whippet	187
22	n02091244-Ibizan_hound	188
23	n02091467-Norwegian_elkhound	196
24	n02091635-otterhound	151
25	n02091831-Saluki	200
26	n02092002-Scottish_deerhound	232
27	n02092339-Weimaraner	160
28	n02093256-Staffordshire_bullterrier	155
29	n02093428-American_Staffordshire_terrier	164
30	n02093647-Bedlington_terrier	182
31	n02093754-Border_terrier	172
32	n02093859-Kerry_blue_terrier	179
33	n02093991-Irish_terrier	169

34	n02094114-Norfolk_terrier	172
35	n02094258-Norwich_terrier	185
36	n02094433-Yorkshire_terrier	164
37	n02095314-wire-haired_fox_terrier	157
38	n02095570-Lakeland_terrier	197
39	n02095889-Sealyham_terrier	202
40	n02096051-Airedale	202
41	n02096177-cairn	197
42	n02096294-Australian_terrier	196
43	n02096437-Dandie_Dinmont	180
44	n02096585-Boston_bull	182
45	n02097047-miniature_schnauzer	154
46	n02097130-giant_schnauzer	157
47	n02097209-standard_schnauzer	155
48	n02097298-Scotch_terrier	158
49	n02097474-Tibetan_terrier	206
50	n02097658-silky_terrier	183
51	n02098105-soft-coated_wheaten_terrier	156
52	n02098286-West_Highland_white_terrier	169
53	n02098413-Lhasa	186
54	n02099267-flat-coated_retriever	152
55	n02099429-curly-coated_retriever	151
56	n02099601-golden_retriever	150
57	n02099712-Labrador_retriever	171
58	n02099849-Chesapeake_Bay_retriever	167
59	n02100236-German_short-haired_pointer	152
60	n02100583-vizsla	154
61	n02100735-English_setter	161
62	n02100877-Irish_setter	155
63	n02101006-Gordon_setter	153
64	n02101388-Brittany_spaniel	152
65	n02101556-clumber	150
66	n02102040-English_springer	159
67	n02102177-Welsh_springer_spaniel	150
68	n02102318-cocker_spaniel	159
69	n02102480-Sussex_spaniel	151
70	n02102973-Irish_water_spaniel	150
71	n02104029-kuvasz	150
72	n02104365-schipperke	154
73	n02105056-groenendael	150
74	n02105162-malinois	150

75	n02105251-briard	152
76	n02105412-kelpie	153
77	n02105505-komondor	154
78	n02105641-Old_English_sheepdog	169
79	n02105855-Shetland_sheepdog	157
80	n02106030-collie	153
81	n02106166-Border_collie	150
82	n02106382-Bouvier_des_Flandres	150
83	n02106550-Rottweiler	152
84	n02106662-German_shepherd	152
85	n02107142-Doberman	150
86	n02107312-miniature_pinscher	184
87	n02107574-Greater_Swiss_Mountain_dog	168
88	n02107683-Bernese_mountain_dog	218
89	n02107908-Appenzeller	151
90	n02108000-EntleBucher	202
91	n02108089-boxer	151
92	n02108422-bull_mastiff	156
93	n02108551-Tibetan_mastiff	152
94	n02108915-French_bulldog	159
95	n02109047-Great_Dane	156
96	n02109525-Saint_Bernard	170
97	n02109961-Eskimo_dog	150
98	n02110063-malamute	178
99	n02110185-Siberian_husky	192
100	n02110627-affenpinscher	150
101	n02110806-basenji	209
102	n02110958-pug	200
103	n02111129-Leonberg	210
104	n02111277-Newfoundland	195
105	n02111500-Great_Pyrenees	213
106	n02111889-Samoyed	218
107	n02112018-Pomeranian	219
108	n02112137-chow	196
109	n02112350-keeshond	158
110	n02112706-Brabancon_griffon	153
111	n02113023-Pembroke	181
112	n02113186-Cardigan	155
113	n02113624-toy_poodle	151
114	n02113712-miniature_poodle	155
115	n02113799-standard_poodle	159

116	n02113978-Mexican_hairless	155
117	n02115641-dingo	156
118	n02115913-dhole	150
119	n02116738-African_hunting_dog	169

Codes

Data Preparation

```
#
################################################################################################################
################################
# Import
#
################################################################################################################
################################
import os
import cv2
import torch
import numpy as np
import torchvision
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

#
################################################################################################################
################################
# Load Data
#
################################################################################################################
################################

if "Images" not in os.listdir(os.getcwd()):
    os.system("wget http://vision.stanford.edu/aditya86/ImageNetDogs/images.tar")
    os.system("tar -xvf images.tar")

os.listdir("Images")

DATA_DIR = os.getcwd() + "/Images/"
RESIZE_TO = 100, 100
x, y = [], []
for i in range(len(os.listdir(DATA_DIR))):
    for path in [f for f in os.listdir(DATA_DIR+os.listdir(DATA_DIR)[i])]:
        x.append(cv2.resize(cv2.imread(DATA_DIR + os.listdir(DATA_DIR)[i] + '/' + path),
(RESIZE_TO)))
        label = os.listdir(DATA_DIR)[i]
        y.append(label)

x, y = np.array(x), np.array(y)
y_label = y

#
################################################################################################################
################################
```

```
# One-Hot-Encode
#
********************************************************************************************
*****************************

# integer encode
le = preprocessing.LabelEncoder()
le.fit(os.listdir(DATA_DIR))
list(le.classes_)
y = le.transform(y)
print(x.shape, y.shape)


#
********************************************************************************************
*****************************
# Data Split
#
********************************************************************************************
*****************************

x_train, x_test, y_train, y_test, y_label_train, y_label_test = train_test_split(x, y,
y_label, random_state=1, test_size=0.3)
np.save("x_train.npy", x_train); np.save("y_train.npy", y_train);
np.save("y_label_train.npy", y_label_train);
np.save("x_test.npy", x_test); np.save("y_test.npy", y_test);
np.save("y_label_test.npy", y_label_test)
```

Data Visualization

```
#
********************************************************************************************
*****************************
# Import
#
********************************************************************************************
*****************************
import os
import cv2
import torch
import numpy as np
import pandas as pd
import torchvision
import seaborn as sns
import matplotlib.pyplot as plt

#
********************************************************************************************
*****************************
# Data Prep
#
********************************************************************************************
*****************************

Load_Path = '/home/ubuntu/Deep-Learning/Final_Project/Data_prep/'
x_train, y_train = np.load(Load_Path + "x_train.npy"), np.load(Load_Path +
"y_train.npy")
x_test, y_test = np.load(Load_Path + "x_test.npy"), np.load(Load_Path + "y_test.npy")
```

```
y_label_train, y_label_test = np.load(Load_Path + "y_label_train.npy"),
np.load(Load_Path + "y_label_test.npy")
#
******************************************************************************************
*****************************
# Data Vis
#
******************************************************************************************
*****************************

classes = np.unique(y_label_train).tolist()
#plt.imshow(x_train[5])
#plt.show()
# print labels
print("The dog breeds shown above are:" '\n',
      ', '.join('%5s' % classes[y_train[j]] for j in range(5)))

# check diversity
print("Train set contains " , len(np.unique(y_label_train)) , "categories." '\n'
      "Test  set contains " , len(np.unique(y_label_test)) , "categories.")

#
******************************************************************************************
*****************************
# Data Frame
#
******************************************************************************************
*****************************
# concatenate data
X, Y, Y_label = np.concatenate((x_train, x_test)), np.concatenate((y_train, y_test)),
np.concatenate((y_label_train, y_label_test))

unique, counts = np.unique(Y_label, return_counts=True)
my_dict = dict(zip(unique, counts))

#s = pd.Series(my_dict,index=my_dict.keys())
s = pd.Series(my_dict, name='Number of Observations')
s.index.name = 'Dog Breed'
s = s.reset_index()
print(s)
s.to_csv('Dog.csv')

unique, counts = np.unique(Y, return_counts=True)
my_dict1 = dict(zip(unique, counts))
s1 = pd.Series(my_dict1, name='Number of Observations')
s1.index.name = 'Encoded Dog Breed'
s1 = s.reset_index()
#s.plot(kind='bar',x='Dog Breed',y='Number of Observations')
s1.plot(kind='bar',x='index',y='Number of Observations')
plt.savefig('plot.png')
plt.show()
```

GAN with FashionMNIST data

```
import os
import random
import cv2
# import tensorflow as tf
import numpy as np
```

```
from keras.models import Model, Sequential
from keras.layers import Input, Reshape, Dense, Dropout, \
    Activation, LeakyReLU, Conv2D, Conv2DTranspose, \
    MaxPooling2D, UpSampling2D, Flatten, BatchNormalization
from keras.initializers import glorot_uniform, glorot_normal
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

SEED = 42
os.environ['PYTHONHASHSEED'] = str(SEED)
random.seed(SEED)
np.random.seed(SEED)
# tf.random.set_seed(SEED)
weight_init = glorot_normal(seed=SEED)

LR = 0.0002
from keras.datasets.fashion_mnist import load_data


# real_100 = np.load('x_train.npy')
(real_100,_), (_,_) = load_data()
X = real_100.reshape((-1, 28, 28, 1))
# convert from ints to floats
real_100 = X.astype('float32')
# scale from [0,255] to [-1,1]
# real_100 = (X - 127.5) / 127.5

real = np.ndarray(shape=(real_100.shape[0], 64, 64, 1))
for i in range(real_100.shape[0]):
    real[i] = (cv2.resize(real_100[i], (64, 64))).reshape(64, 64, 1)


img_size = real[0].shape

# latent space of noise
z = (100,)
optimizer = Adam(lr=0.0002, beta_1=0.5)

# Build Generator
def generator_conv():
    noise = Input(shape=z)
    x = Dense(4*4*256)(noise)
    x = LeakyReLU(alpha=0.2)(x)
    x = Reshape((4, 4, 256))(x)
    x = Conv2DTranspose(filters=128,
                        kernel_size=(4, 4),
                        strides=(2, 2),
                        padding='same')(x)
    x = LeakyReLU(0.2)(x)
    # x = BatchNormalization()(x)
    x = Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same')(x)
    x = LeakyReLU(0.2)(x)
    # x = BatchNormalization()(x)
    x = Conv2DTranspose(64, (4, 4), strides=(2, 2), padding='same')(x)
    x = LeakyReLU(0.2)(x)
    # x = BatchNormalization()(x)
    x = Conv2DTranspose(64, (4, 4), strides=(2, 2), padding='same')(x)
    generated = Conv2D(1, (8, 8), padding='same', activation='tanh')(x)

    generator = Model(inputs=noise, outputs=generated)
    return generator
```

```
# gen = generator()
# gen.summary()
# fake = gen.predict(np.random.normal(0, 1, size=(100,)).reshape(1, -1))
# plt.imshow(fake[0])
# plt.show()

# Build Discriminator
def discriminator_conv():
    img = Input(img_size)
    x = Conv2D(128, kernel_size=(3, 3), strides=(2, 2), padding='same')(img)
    x = LeakyReLU(0.2)(x)
    x = Conv2D(128, (3, 3), strides=(2, 2), padding='same')(x)
    x = LeakyReLU(0.2)(x)
    x = Flatten()(x)
    x = Dropout(0.3)(x)
    out = Dense(1, activation='sigmoid')(x)

    discriminator = Model(inputs=img, outputs=out)
    discriminator.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    return discriminator

# discr = discriminator()
# discr.summary()


def generator_fc():
    noise = Input(shape=z)
    x = Dense(256, kernel_initializer=weight_init)(noise)
    x = LeakyReLU(0.2)(x)
    x = BatchNormalization()(x)
    x = Dense(512, kernel_initializer=weight_init)(x)
    x = LeakyReLU(0.2)(x)
    x = BatchNormalization()(x)
    x = Dense(1024, kernel_initializer=weight_init)(x)
    x = LeakyReLU(0.2)(x)
    x = BatchNormalization()(x)
    x = Dense(np.prod(img_size), activation='tanh', kernel_initializer=weight_init)(x)
    generated = Reshape(img_size)(x)
    generator = Model(inputs=noise, outputs=generated)
    return generator

def discriminator_fc():
    img = Input(shape=img_size)
    x = Flatten()(img)
    x = Dense(512, kernel_initializer=weight_init)(x)
    x = LeakyReLU(0.2)(x)
    x = Dense(256, kernel_initializer=weight_init)(x)
    x = LeakyReLU(0.2)(x)
    out = Dense(1, activation='sigmoid', kernel_initializer=weight_init)(x)

    discriminator = Model(inputs=img, outputs=out)
    return discriminator



def generator_trainer(generator, discriminator):

    discriminator.trainable = False

    model = Sequential()
    model.add(generator)
```

```
    model.add(discriminator)
    model.compile(optimizer=optimizer, loss='binary_crossentropy')

    return model

# GAN model compiling
class GAN():
    def __init__(self, model='conv', img_shape=(64, 64, 1), latent_space=(100,)):
        self.img_size = img_shape  # channel_last
        self.z = latent_space
        self.optimizer = Adam(0.0002, 0.5)

        if model == 'conv':
            self.gen = generator_conv()
            self.discr = discriminator_conv()
        else:
            self.gen = generator_fc()
            self.discr = discriminator_fc()

        self.train_gen = generator_trainer(self.gen, self.discr)
        # self.gen.compile(self.optimizer, loss='binary_crossentropy')
        # self.discr.compile(self.optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

        # self.discr.trainable = False
        # noise = Input(self.z)
        # fake = self.gen(noise)
        # out = self.discr(fake)

        # self.train_gen = Model(inputs=noise, outputs=out)
        # self.train_gen.compile(self.optimizer, loss='binary_crossentropy')
        self.loss_D, self.loss_G = [], []

    def Generator(self):

        return

    def Descriminator(self):

        return

    def train(self, imgs, epochs=50, batch_size=128):
        # load data
        imgs = (imgs - 127.5)/127.5
        bs_half = batch_size//2

        for epoch in range(epochs):
            # Get a half batch of random real images
            idx = np.random.randint(0, imgs.shape[0], bs_half)
            real_img = imgs[idx]

            # Generate a half batch of new images
            noise = np.random.normal(0, 1, size=((bs_half,) + self.z))
            fake_img = self.gen.predict(noise)
            # Train the discriminator
            loss_fake = self.discr.train_on_batch(fake_img, np.zeros((bs_half, 1)))
            loss_real = self.discr.train_on_batch(real_img, np.ones((bs_half, 1)))
            self.loss_D.append(0.5 * np.add(loss_fake, loss_real))

            # Train the generator
            noise = np.random.normal(0, 1, size=((batch_size,) + self.z))
            loss_gen = self.train_gen.train_on_batch(noise, np.ones(batch_size))
            self.loss_G.append(loss_gen)
```

```
            if (epoch + 1) * 10 % epochs == 0:
                print('Epoch (%d / %d): [Loss_D_real: %f, Loss_D_fake: %f, acc: %.2f%%]
[Loss_G: %f]' %
                      (epoch+1, epochs, loss_real[0], loss_fake[0], 100*self.loss_D[-1][1],
loss_gen))

        return


def plt_img(gan):
    r, c = 2, 4
    noise = np.random.normal(0, 1, (r * c, 100))
    gen_imgs = gan.gen.predict(noise)
    # Rescale images 0 - 1
    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
    plt.show()
    return
# train GAN
gan = GAN(model='conv')
LEARNING_STEPS = 52
for learning_step in range(LEARNING_STEPS):
    print('LEARNING STEP # ', learning_step+1, '-'*50)
    # iteration = [5, 5]
    # Adjust the learning times to balance G and D at a competitive level.
    # if gan.loss_D is not None:
    #     acc = gan.loss_D[1]
    #     iteration = [int(5 * (1-acc)) + 1, int(5 * acc) + 1]
    gan.train(real, epochs=100, batch_size=128)
    if (learning_step+1)%1 == 0:
        plt_img(gan)
```

Percentage of the Copied Codes: 16%