Open in Colab

# GAN Training

Reference: https://github.com/davidADSP/GDL_code/

License: https://raw.githubusercontent.com/davidADSP/GDL_code/master/LICENSE

# Training

# Preparation

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

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import RandomNormal

import numpy as np
import json
import os
import pickle as pkl
import matplotlib.pyplot as plt


class GAN():
    def __init__(self
        , input_dim
        , discriminator_conv_filters
        , discriminator_conv_kernel_size
        , discriminator_conv_strides
        , discriminator_batch_norm_momentum
        , discriminator_activation
        , discriminator_dropout_rate
        , discriminator_learning_rate
        , generator_initial_dense_layer_size
        , generator_upsample
        , generator_conv_filters
        , generator_conv_kernel_size
        , generator_conv_strides
        , generator_batch_norm_momentum
        , generator_activation
        , generator_dropout_rate
        , generator_learning_rate
```

```
    , optimiser
    , z_dim
    ):

    self.name = 'gan'

    self.input_dim = input_dim
    self.discriminator_conv_filters = discriminator_conv_filters
    self.discriminator_conv_kernel_size = discriminator_conv_kernel_size
    self.discriminator_conv_strides = discriminator_conv_strides
    self.discriminator_batch_norm_momentum = discriminator_batch_norm_momentum
    self.discriminator_activation = discriminator_activation
    self.discriminator_dropout_rate = discriminator_dropout_rate
    self.discriminator_learning_rate = discriminator_learning_rate

    self.generator_initial_dense_layer_size = generator_initial_dense_layer_size
    self.generator_upsample = generator_upsample
    self.generator_conv_filters = generator_conv_filters
    self.generator_conv_kernel_size = generator_conv_kernel_size
    self.generator_conv_strides = generator_conv_strides
    self.generator_batch_norm_momentum = generator_batch_norm_momentum
    self.generator_activation = generator_activation
    self.generator_dropout_rate = generator_dropout_rate
    self.generator_learning_rate = generator_learning_rate

    self.optimiser = optimiser
    self.z_dim = z_dim

    self.n_layers_discriminator = len(discriminator_conv_filters)
    self.n_layers_generator = len(generator_conv_filters)

    self.weight_init = RandomNormal(mean=0., stddev=0.02)

    self.d_losses = []
    self.g_losses = []

    self.epoch = 0

    self._build_discriminator()
    self._build_generator()

    self._build_adversarial()

def get_activation(self, activation):
    if activation == 'leaky_relu':
        layer = LeakyReLU(alpha = 0.2)
    else:
        layer = Activation(activation)
    return layer

def _build_discriminator(self):

    ### THE discriminator
    discriminator_input = Input(shape=self.input_dim, name='discriminator_input')
```

```
        x = discriminator_input

        for i in range(self.n_layers_discriminator):

            x = Conv2D(
                filters = self.discriminator_conv_filters[i]
                , kernel_size = self.discriminator_conv_kernel_size[i]
                , strides = self.discriminator_conv_strides[i]
                , padding = 'same'
                , name = 'discriminator_conv_' + str(i)
                , kernel_initializer = self.weight_init
                )(x)

            if self.discriminator_batch_norm_momentum and i > 0:
                x = BatchNormalization(momentum = self.discriminator_batch_norm_momentum)(x)

            x = self.get_activation(self.discriminator_activation)(x)

            if self.discriminator_dropout_rate:
                x = Dropout(rate = self.discriminator_dropout_rate)(x)

        x = Flatten()(x)

        discriminator_output = Dense(1, activation='sigmoid', kernel_initializer = self.weight_init

        self.discriminator = Model(discriminator_input, discriminator_output)


    def _build_generator(self):

        ### THE generator

        generator_input = Input(shape=(self.z_dim,), name='generator_input')

        x = generator_input

        x = Dense(np.prod(self.generator_initial_dense_layer_size), kernel_initializer = self.weigh

        if self.generator_batch_norm_momentum:
            x = BatchNormalization(momentum = self.generator_batch_norm_momentum)(x)

        x = self.get_activation(self.generator_activation)(x)

        x = Reshape(self.generator_initial_dense_layer_size)(x)

        if self.generator_dropout_rate:
            x = Dropout(rate = self.generator_dropout_rate)(x)

        for i in range(self.n_layers_generator):

            if self.generator_upsample[i] == 2:
                x = UpSampling2D()(x)
                x = Conv2D(
                    filters = self.generator_conv_filters[i]
                    , kernel_size = self.generator_conv_kernel_size[i]
```

```
                , padding = 'same'
                , name = 'generator_conv_' + str(i)
                , kernel_initializer = self.weight_init
            )(x)
        else:

            x = Conv2DTranspose(
                filters = self.generator_conv_filters[i]
                , kernel_size = self.generator_conv_kernel_size[i]
                , padding = 'same'
                , strides = self.generator_conv_strides[i]
                , name = 'generator_conv_' + str(i)
                , kernel_initializer = self.weight_init
                )(x)

        if i < self.n_layers_generator - 1:

            if self.generator_batch_norm_momentum:
                x = BatchNormalization(momentum = self.generator_batch_norm_momentum)(x)

            x = self.get_activation(self.generator_activation)(x)


        else:

            x = Activation('tanh')(x)


    generator_output = x

    self.generator = Model(generator_input, generator_output)


def get_opti(self, lr):
    if self.optimiser == 'adam':
        opti = Adam(lr=lr, beta_1=0.5)
    elif self.optimiser == 'rmsprop':
        opti = RMSprop(lr=lr)
    else:
        opti = Adam(lr=lr)

    return opti

def set_trainable(self, m, val):
    m.trainable = val
    for l in m.layers:
        l.trainable = val


def _build_adversarial(self):

    ### COMPILE DISCRIMINATOR

    self.discriminator.compile(
    optimizer=self.get_opti(self.discriminator_learning_rate)
```

```
    , loss = 'binary_crossentropy'
    ,  metrics = ['accuracy']
    )

    ### COMPILE THE FULL GAN

    self.set_trainable(self.discriminator, False)

    model_input = Input(shape=(self.z_dim,), name='model_input')
    model_output = self.discriminator(self.generator(model_input))
    self.model = Model(model_input, model_output)

    self.model.compile(optimizer=self.get_opti(self.generator_learning_rate) , loss='binary_cro
    , experimental_run_tf_function=False
    )

    self.set_trainable(self.discriminator, True)




def train_discriminator(self, x_train, batch_size, using_generator):

    valid = np.ones((batch_size,1))
    fake = np.zeros((batch_size,1))

    if using_generator:
        true_imgs = next(x_train)[0]
        if true_imgs.shape[0] != batch_size:
            true_imgs = next(x_train)[0]
    else:
        idx = np.random.randint(0, x_train.shape[0], batch_size)
        true_imgs = x_train[idx]

    noise = np.random.normal(0, 1, (batch_size, self.z_dim))
    gen_imgs = self.generator.predict(noise)

    d_loss_real, d_acc_real =   self.discriminator.train_on_batch(true_imgs, valid)
    d_loss_fake, d_acc_fake =   self.discriminator.train_on_batch(gen_imgs, fake)
    d_loss =  0.5 * (d_loss_real + d_loss_fake)
    d_acc = 0.5 * (d_acc_real + d_acc_fake)

    return [d_loss, d_loss_real, d_loss_fake, d_acc, d_acc_real, d_acc_fake]

def train_generator(self, batch_size):
    valid = np.ones((batch_size,1))
    noise = np.random.normal(0, 1, (batch_size, self.z_dim))
    return self.model.train_on_batch(noise, valid)


def train(self, x_train, batch_size, epochs, run_folder
, print_every_n_batches = 50
, using_generator = False):

    for epoch in range(self.epoch, self.epoch + epochs):
```

```
        d = self.train_discriminator(x_train, batch_size, using_generator)
        g = self.train_generator(batch_size)

        print ("%d [D loss: (%.3f)(R %.3f, F %.3f)] [D acc: (%.3f)(%.3f, %.3f)] [G loss: %.3f]

        self.d_losses.append(d)
        self.g_losses.append(g)

        if epoch % print_every_n_batches == 0:
            self.sample_images(run_folder)
            self.model.save_weights(os.path.join(run_folder, 'weights/weights-%d.h5' % (epoch))
            self.model.save_weights(os.path.join(run_folder, 'weights/weights.h5'))
            self.save_model(run_folder)

        self.epoch += 1


def sample_images(self, run_folder):
    r, c = 5, 5
    noise = np.random.normal(0, 1, (r * c, self.z_dim))
    gen_imgs = self.generator.predict(noise)

    gen_imgs = 0.5 * (gen_imgs + 1)
    gen_imgs = np.clip(gen_imgs, 0, 1)

    fig, axs = plt.subplots(r, c, figsize=(15,15))
    cnt = 0

    for i in range(r):
        for j in range(c):
            axs[i,j].imshow(np.squeeze(gen_imgs[cnt, :,:,:]), cmap = 'gray')
            axs[i,j].axis('off')
            cnt += 1
    fig.savefig(os.path.join(run_folder, "images/sample_%d.png" % self.epoch))
    plt.close()

def plot_model(self, run_folder):
    plot_model(self.model, to_file=os.path.join(run_folder ,'viz/model.png'), show_shapes = Tru
    plot_model(self.discriminator, to_file=os.path.join(run_folder ,'viz/discriminator.png'), s
    plot_model(self.generator, to_file=os.path.join(run_folder ,'viz/generator.png'), show_shap

def save(self, folder):

    with open(os.path.join(folder, 'params.pkl'), 'wb') as f:
        pkl.dump([
            self.input_dim
            , self.discriminator_conv_filters
            , self.discriminator_conv_kernel_size
            , self.discriminator_conv_strides
            , self.discriminator_batch_norm_momentum
            , self.discriminator_activation
            , self.discriminator_dropout_rate
            , self.discriminator_learning_rate
            , self.generator_initial_dense_layer_size
            , self.generator_upsample
```

```
                , self.generator_upsample
                , self.generator_conv_filters
                , self.generator_conv_kernel_size
                , self.generator_conv_strides
                , self.generator_batch_norm_momentum
                , self.generator_activation
                , self.generator_dropout_rate
                , self.generator_learning_rate
                , self.optimiser
                , self.z_dim
                ], f)

        self.plot_model(folder)

    def save_model(self, run_folder):
        self.model.save(os.path.join(run_folder, 'model.h5'))
        self.discriminator.save(os.path.join(run_folder, 'discriminator.h5'))
        self.generator.save(os.path.join(run_folder, 'generator.h5'))

    def load_weights(self, filepath):
        self.model.load_weights(filepath)
```

## imports

```
!mkdir -p run/gan
```

```
# run params
SECTION = 'gan'
RUN_ID = '0001'
DATA_NAME = 'camel'
RUN_FOLDER = 'run/{}/'.format(SECTION)
RUN_FOLDER += '_'.join([RUN_ID, DATA_NAME])

if not os.path.exists(RUN_FOLDER):
    os.mkdir(RUN_FOLDER)
    os.mkdir(os.path.join(RUN_FOLDER, 'viz'))
    os.mkdir(os.path.join(RUN_FOLDER, 'images'))
    os.mkdir(os.path.join(RUN_FOLDER, 'weights'))

mode =  'build' #'load' #
```

## data

download camel dataset from [quickdraw_dataset]

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

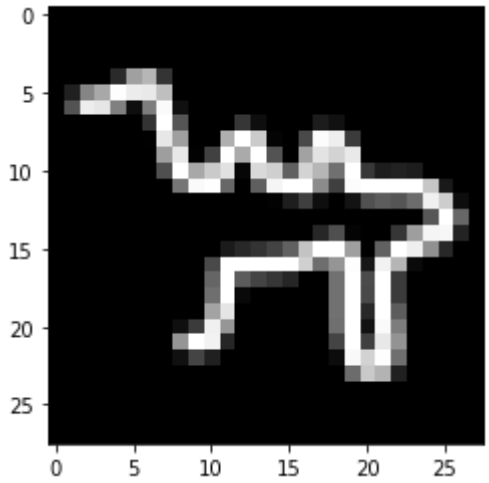
```
DATA_PATH='/content/drive/MyDrive/DeepLearning/book14/local/GDL_code/data/camel'

!ls {DATA_PATH}

      full_numpy_bitmap_camel.npy

!mkdir -p data

!cp {DATA_PATH}/full_numpy_bitmap_camel.npy data

!ls data

      full_numpy_bitmap_camel.npy

from os import walk

def load_safari():

#    mypath = os.path.join("./data", folder)
    mypath = "./data"
    txt_name_list = []
    for (dirpath, dirnames, filenames) in walk(mypath):
        for f in filenames:
            if f != '.DS_Store':
                txt_name_list.append(f)
                break

    slice_train = int(80000/len(txt_name_list))  ###Setting value to be 80000 for the final dataset
    i = 0
    seed = np.random.randint(1, 10e6)

    for txt_name in txt_name_list:
        txt_path = os.path.join(mypath,txt_name)
        x = np.load(txt_path)
        x = (x.astype('float32') - 127.5) / 127.5
        # x = x.astype('float32') / 255.0

        x = x.reshape(x.shape[0], 28, 28, 1)

        y = [i] * len(x)
        np.random.seed(seed)
        np.random.shuffle(x)
        np.random.seed(seed)
        np.random.shuffle(y)
        x = x[:slice_train]
        y = y[:slice_train]
        if i != 0:
            xtotal = np.concatenate((x,xtotal), axis=0)
            ytotal = np.concatenate((y,ytotal), axis=0)
        else:
            xtotal = x
```

```
            xtotal = x
            ytotal = y
        i += 1

    return xtotal, ytotal


(x_train, y_train) = load_safari()


x_train.shape
```

```
(80000, 28, 28, 1)
```

```
plt.imshow(x_train[200,:,:,0], cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7f7c35e20dd0>
```




## architecture

```
gan = GAN(input_dim = (28,28,1)
        , discriminator_conv_filters = [64,64,128,128]
        , discriminator_conv_kernel_size = [5,5,5,5]
        , discriminator_conv_strides = [2,2,2,1]
        , discriminator_batch_norm_momentum = None
        , discriminator_activation = 'relu'
        , discriminator_dropout_rate = 0.4
        , discriminator_learning_rate = 0.0008
        , generator_initial_dense_layer_size = (7, 7, 64)
        , generator_upsample = [2,2, 1, 1]
        , generator_conv_filters = [128,64, 64,1]
        , generator_conv_kernel_size = [5,5,5,5]
        , generator_conv_strides = [1,1, 1, 1]
        , generator_batch_norm_momentum = 0.9
        , generator_activation = 'relu'
        , generator_dropout_rate = None
        , generator_learning_rate = 0.0004
        , optimiser = 'rmsprop'
        , z_dim = 100
        )
```

```
if mode == 'build':
    gan.save(RUN_FOLDER)
else:
    gan.load_weights(os.path.join(RUN_FOLDER, 'weights/weights.h5'))
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: UserWarning: T
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

```
gan.discriminator.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
discriminator_input (InputLa [(None, 28, 28, 1)]       0
_________________________________________________________________
discriminator_conv_0 (Conv2D (None, 14, 14, 64)        1664
_________________________________________________________________
activation (Activation)      (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
discriminator_conv_1 (Conv2D (None, 7, 7, 64)          102464
_________________________________________________________________
activation_1 (Activation)    (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
discriminator_conv_2 (Conv2D (None, 4, 4, 128)         204928
_________________________________________________________________
activation_2 (Activation)    (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
discriminator_conv_3 (Conv2D (None, 4, 4, 128)         409728
_________________________________________________________________
activation_3 (Activation)    (None, 4, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 2049
=================================================================
Total params: 720,833
Trainable params: 720,833
Non-trainable params: 0
_________________________________________________________________
```

```
gan.generator.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
generator_input (InputLayer) [(None, 100)]             0
```

```
dense_1 (Dense)              (None, 3136)              316736
_________________________________________________________________
batch_normalization (BatchNo (None, 3136)              12544
_________________________________________________________________
activation_4 (Activation)    (None, 3136)              0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 64)        0
_________________________________________________________________
generator_conv_0 (Conv2D)    (None, 14, 14, 128)       204928
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
activation_5 (Activation)    (None, 14, 14, 128)       0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
generator_conv_1 (Conv2D)    (None, 28, 28, 64)        204864
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_6 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
generator_conv_2 (Conv2DTran (None, 28, 28, 64)        102464
_________________________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_7 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
generator_conv_3 (Conv2DTran (None, 28, 28, 1)         1601
_________________________________________________________________
activation_8 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 844,161
Trainable params: 837,377
Non-trainable params: 6,784
_________________________________________________________________
```

## training

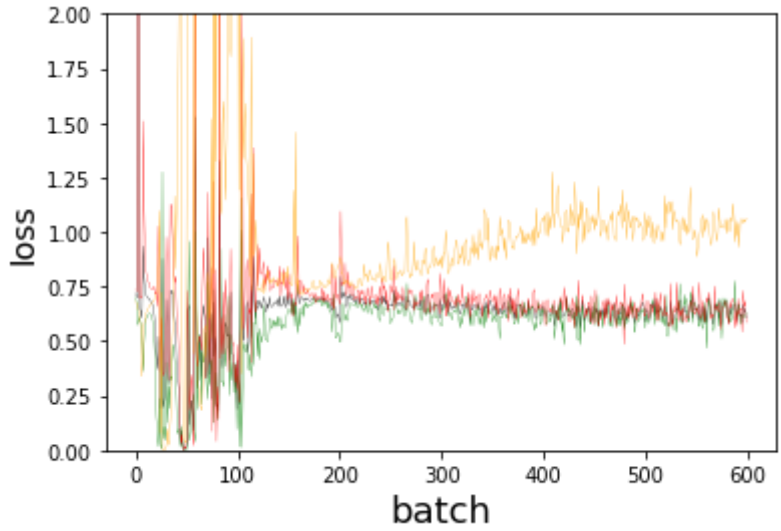
```
BATCH_SIZE = 64
#EPOCHS = 6000
EPOCHS = 600
PRINT_EVERY_N_BATCHES = 5


gan.train(
    x_train
    , batch_size = BATCH_SIZE
    , epochs = EPOCHS
    , run_folder = RUN_FOLDER
    , print_every_n_batches = PRINT_EVERY_N_BATCHES
)
```

```
546 [D loss: (0.607)(R 0.564, F 0.651)] [D acc: (0.664)(0.625, 0.703)] [G loss: 0.985] [G a
547 [D loss: (0.713)(R 0.675, F 0.751)] [D acc: (0.562)(0.578, 0.547)] [G loss: 0.966] [G a
```

```
548 [D loss: (0.646)(R 0.667, F 0.626)] [D acc: (0.664)(0.562, 0.766)] [G loss: 0.979] [G a
549 [D loss: (0.649)(R 0.676, F 0.623)] [D acc: (0.641)(0.531, 0.750)] [G loss: 1.009] [G a
550 [D loss: (0.645)(R 0.649, F 0.641)] [D acc: (0.633)(0.562, 0.703)] [G loss: 0.951] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
551 [D loss: (0.651)(R 0.647, F 0.655)] [D acc: (0.617)(0.562, 0.672)] [G loss: 0.887] [G a
552 [D loss: (0.593)(R 0.568, F 0.618)] [D acc: (0.680)(0.672, 0.688)] [G loss: 0.946] [G a
553 [D loss: (0.669)(R 0.570, F 0.768)] [D acc: (0.609)(0.719, 0.500)] [G loss: 0.981] [G a
554 [D loss: (0.627)(R 0.621, F 0.633)] [D acc: (0.617)(0.547, 0.688)] [G loss: 1.015] [G a
555 [D loss: (0.592)(R 0.581, F 0.603)] [D acc: (0.695)(0.656, 0.734)] [G loss: 1.053] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
556 [D loss: (0.594)(R 0.556, F 0.631)] [D acc: (0.680)(0.672, 0.688)] [G loss: 1.079] [G a
557 [D loss: (0.660)(R 0.636, F 0.684)] [D acc: (0.656)(0.562, 0.750)] [G loss: 0.975] [G a
558 [D loss: (0.585)(R 0.581, F 0.588)] [D acc: (0.719)(0.672, 0.766)] [G loss: 1.062] [G a
559 [D loss: (0.655)(R 0.676, F 0.634)] [D acc: (0.641)(0.578, 0.703)] [G loss: 1.005] [G a
560 [D loss: (0.584)(R 0.468, F 0.700)] [D acc: (0.711)(0.797, 0.625)] [G loss: 1.049] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
561 [D loss: (0.632)(R 0.583, F 0.681)] [D acc: (0.680)(0.703, 0.656)] [G loss: 1.060] [G a
562 [D loss: (0.698)(R 0.690, F 0.705)] [D acc: (0.617)(0.578, 0.656)] [G loss: 1.014] [G a
563 [D loss: (0.698)(R 0.674, F 0.722)] [D acc: (0.531)(0.500, 0.562)] [G loss: 0.998] [G a
564 [D loss: (0.623)(R 0.643, F 0.602)] [D acc: (0.672)(0.547, 0.797)] [G loss: 0.943] [G a
565 [D loss: (0.627)(R 0.635, F 0.619)] [D acc: (0.625)(0.578, 0.672)] [G loss: 0.943] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
566 [D loss: (0.595)(R 0.566, F 0.623)] [D acc: (0.625)(0.594, 0.656)] [G loss: 1.018] [G a
567 [D loss: (0.576)(R 0.546, F 0.607)] [D acc: (0.711)(0.656, 0.766)] [G loss: 1.094] [G a
568 [D loss: (0.672)(R 0.592, F 0.752)] [D acc: (0.602)(0.641, 0.562)] [G loss: 1.051] [G a
569 [D loss: (0.650)(R 0.657, F 0.644)] [D acc: (0.648)(0.594, 0.703)] [G loss: 1.015] [G a
570 [D loss: (0.668)(R 0.674, F 0.661)] [D acc: (0.570)(0.516, 0.625)] [G loss: 0.958] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
571 [D loss: (0.643)(R 0.646, F 0.640)] [D acc: (0.578)(0.547, 0.609)] [G loss: 0.988] [G a
572 [D loss: (0.602)(R 0.569, F 0.635)] [D acc: (0.664)(0.656, 0.672)] [G loss: 1.109] [G a
573 [D loss: (0.687)(R 0.696, F 0.679)] [D acc: (0.555)(0.484, 0.625)] [G loss: 1.059] [G a
574 [D loss: (0.608)(R 0.663, F 0.554)] [D acc: (0.711)(0.609, 0.812)] [G loss: 1.021] [G a
575 [D loss: (0.572)(R 0.511, F 0.633)] [D acc: (0.719)(0.719, 0.719)] [G loss: 1.026] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
576 [D loss: (0.636)(R 0.617, F 0.654)] [D acc: (0.609)(0.562, 0.656)] [G loss: 1.032] [G a
577 [D loss: (0.677)(R 0.698, F 0.657)] [D acc: (0.555)(0.469, 0.641)] [G loss: 1.019] [G a
578 [D loss: (0.666)(R 0.653, F 0.679)] [D acc: (0.609)(0.625, 0.594)] [G loss: 1.018] [G a
579 [D loss: (0.687)(R 0.691, F 0.683)] [D acc: (0.570)(0.516, 0.625)] [G loss: 1.059] [G a
580 [D loss: (0.645)(R 0.684, F 0.605)] [D acc: (0.633)(0.547, 0.719)] [G loss: 0.994] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
581 [D loss: (0.612)(R 0.618, F 0.606)] [D acc: (0.641)(0.609, 0.672)] [G loss: 1.060] [G a
582 [D loss: (0.581)(R 0.587, F 0.576)] [D acc: (0.703)(0.672, 0.734)] [G loss: 1.024] [G a
583 [D loss: (0.664)(R 0.652, F 0.676)] [D acc: (0.625)(0.609, 0.641)] [G loss: 1.001] [G a
584 [D loss: (0.620)(R 0.589, F 0.651)] [D acc: (0.664)(0.641, 0.688)] [G loss: 1.068] [G a
585 [D loss: (0.646)(R 0.600, F 0.691)] [D acc: (0.602)(0.609, 0.594)] [G loss: 1.049] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
586 [D loss: (0.651)(R 0.608, F 0.693)] [D acc: (0.617)(0.609, 0.625)] [G loss: 1.130] [G a
587 [D loss: (0.671)(R 0.666, F 0.676)] [D acc: (0.578)(0.531, 0.625)] [G loss: 1.063] [G a
588 [D loss: (0.669)(R 0.776, F 0.561)] [D acc: (0.609)(0.453, 0.766)] [G loss: 0.954] [G a
589 [D loss: (0.632)(R 0.626, F 0.637)] [D acc: (0.656)(0.641, 0.672)] [G loss: 1.051] [G a
590 [D loss: (0.591)(R 0.609, F 0.572)] [D acc: (0.750)(0.656, 0.844)] [G loss: 0.980] [G a
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built
591 [D loss: (0.673)(R 0.650, F 0.695)] [D acc: (0.562)(0.531, 0.594)] [G loss: 1.042] [G a
592 [D loss: (0.644)(R 0.668, F 0.620)] [D acc: (0.570)(0.516, 0.625)] [G loss: 0.931] [G a
593 [D loss: (0.612)(R 0.578, F 0.647)] [D acc: (0.688)(0.656, 0.719)] [G loss: 1.044] [G a
594 [D loss: (0.627)(R 0.604, F 0.650)] [D acc: (0.688)(0.641, 0.734)] [G loss: 1.057] [G a
595 [D loss: (0.584)(R 0.626, F 0.542)] [D acc: (0.727)(0.641, 0.812)] [G loss: 1.059] [G a
```

```
fig = plt.figure()
plt.plot([x[0] for x in gan_d_losses], color='black', linewidth=0.25)
```

```
plt.plot([x[1] for x in gan.d_losses], color='green', linewidth=0.25)
plt.plot([x[2] for x in gan.d_losses], color='red', linewidth=0.25)
plt.plot([x[0] for x in gan.g_losses], color='orange', linewidth=0.25)

plt.xlabel('batch', fontsize=18)
plt.ylabel('loss', fontsize=16)

#plt.xlim(0, 2000)
plt.ylim(0, 2)

plt.show()
```

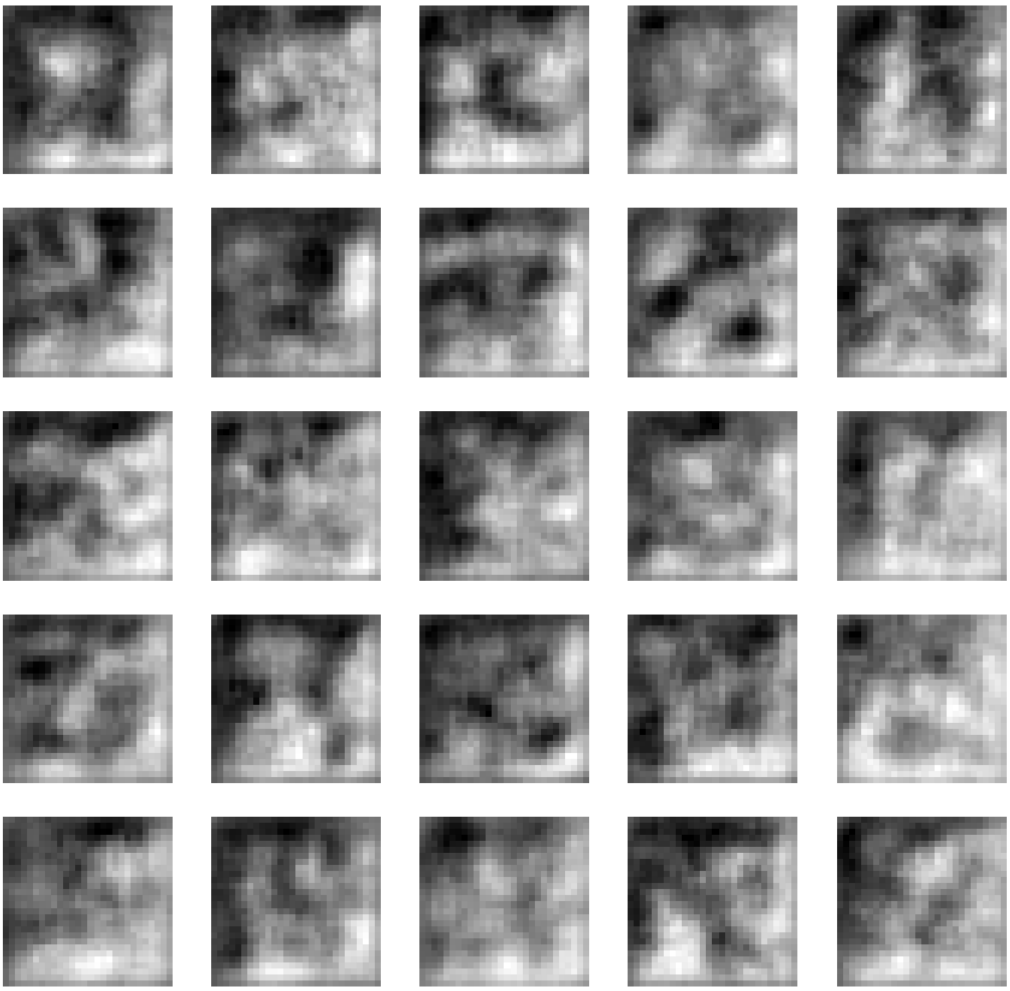



```
fig = plt.figure()
plt.plot([x[3] for x in gan.d_losses], color='black', linewidth=0.25)
plt.plot([x[4] for x in gan.d_losses], color='green', linewidth=0.25)
plt.plot([x[5] for x in gan.d_losses], color='red', linewidth=0.25)
plt.plot([x[1] for x in gan.g_losses], color='orange', linewidth=0.25)

plt.xlabel('batch', fontsize=18)
plt.ylabel('accuracy', fontsize=16)

#plt.xlim(0, 2000)

plt.show()
```

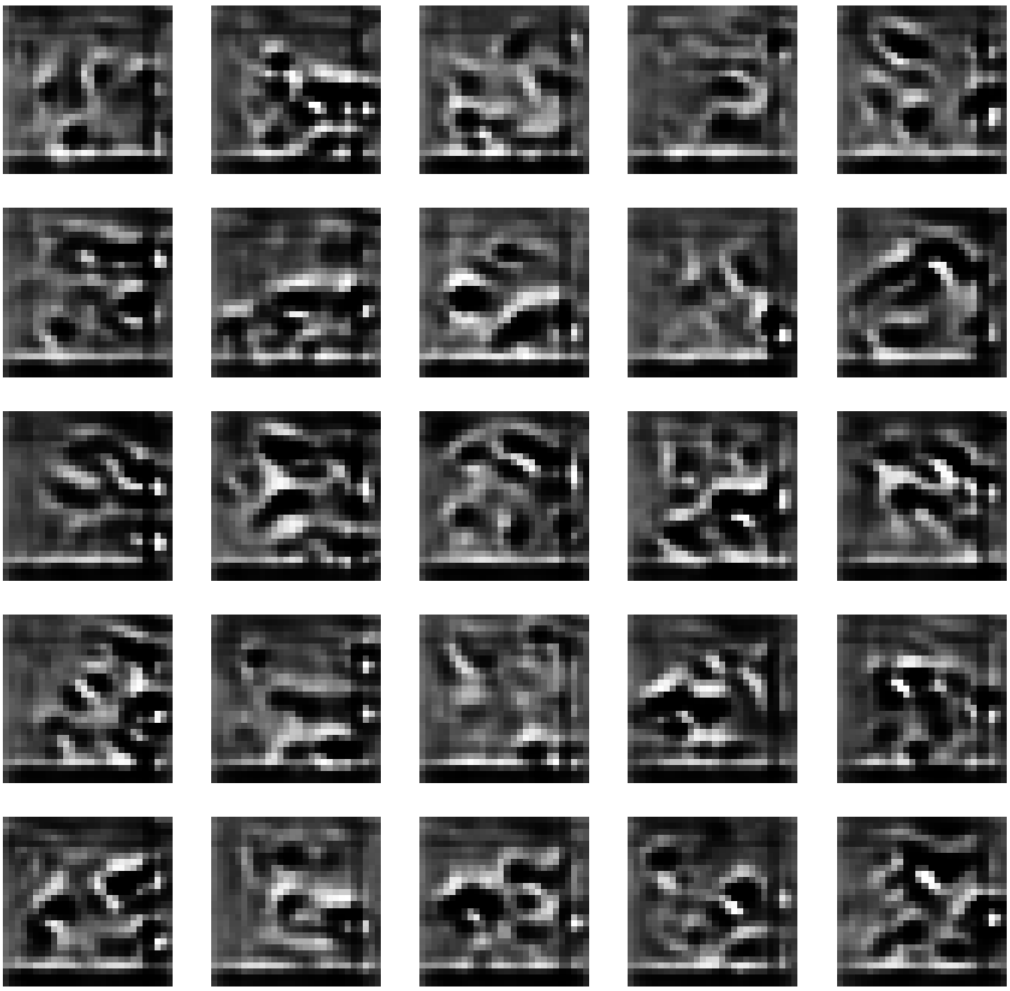

```
!ls run/gan/0001_camel/images
```

```
sample_0.png    sample_205.png  sample_310.png  sample_420.png  sample_530.png
sample_100.png  sample_20.png   sample_315.png  sample_425.png  sample_535.png
sample_105.png  sample_210.png  sample_320.png  sample_430.png  sample_540.png
sample_10.png   sample_215.png  sample_325.png  sample_435.png  sample_545.png
sample_110.png  sample_220.png  sample_330.png  sample_440.png  sample_550.png
sample_115.png  sample_225.png  sample_335.png  sample_445.png  sample_555.png
sample_120.png  sample_230.png  sample_340.png  sample_450.png  sample_55.png
sample_125.png  sample_235.png  sample_345.png  sample_455.png  sample_560.png
sample_130.png  sample_240.png  sample_350.png  sample_45.png   sample_565.png
sample_135.png  sample_245.png  sample_355.png  sample_460.png  sample_570.png
sample_140.png  sample_250.png  sample_35.png   sample_465.png  sample_575.png
sample_145.png  sample_255.png  sample_360.png  sample_470.png  sample_580.png
sample_150.png  sample_25.png   sample_365.png  sample_475.png  sample_585.png
sample_155.png  sample_260.png  sample_370.png  sample_480.png  sample_590.png
sample_15.png   sample_265.png  sample_375.png  sample_485.png  sample_595.png
sample_160.png  sample_270.png  sample_380.png  sample_490.png  sample_5.png
sample_165.png  sample_275.png  sample_385.png  sample_495.png  sample_60.png
sample_170.png  sample_280.png  sample_390.png  sample_500.png  sample_65.png
sample_175.png  sample_285.png  sample_395.png  sample_505.png  sample_70.png
sample_180.png  sample_290.png  sample_400.png  sample_50.png   sample_75.png
sample_185.png  sample_295.png  sample_405.png  sample_510.png  sample_80.png
sample_190.png  sample_300.png  sample_40.png   sample_515.png  sample_85.png
sample_195.png  sample_305.png  sample_410.png  sample_520.png  sample_90.png
sample_200.png  sample_30.png   sample_415.png  sample_525.png  sample_95.png
```

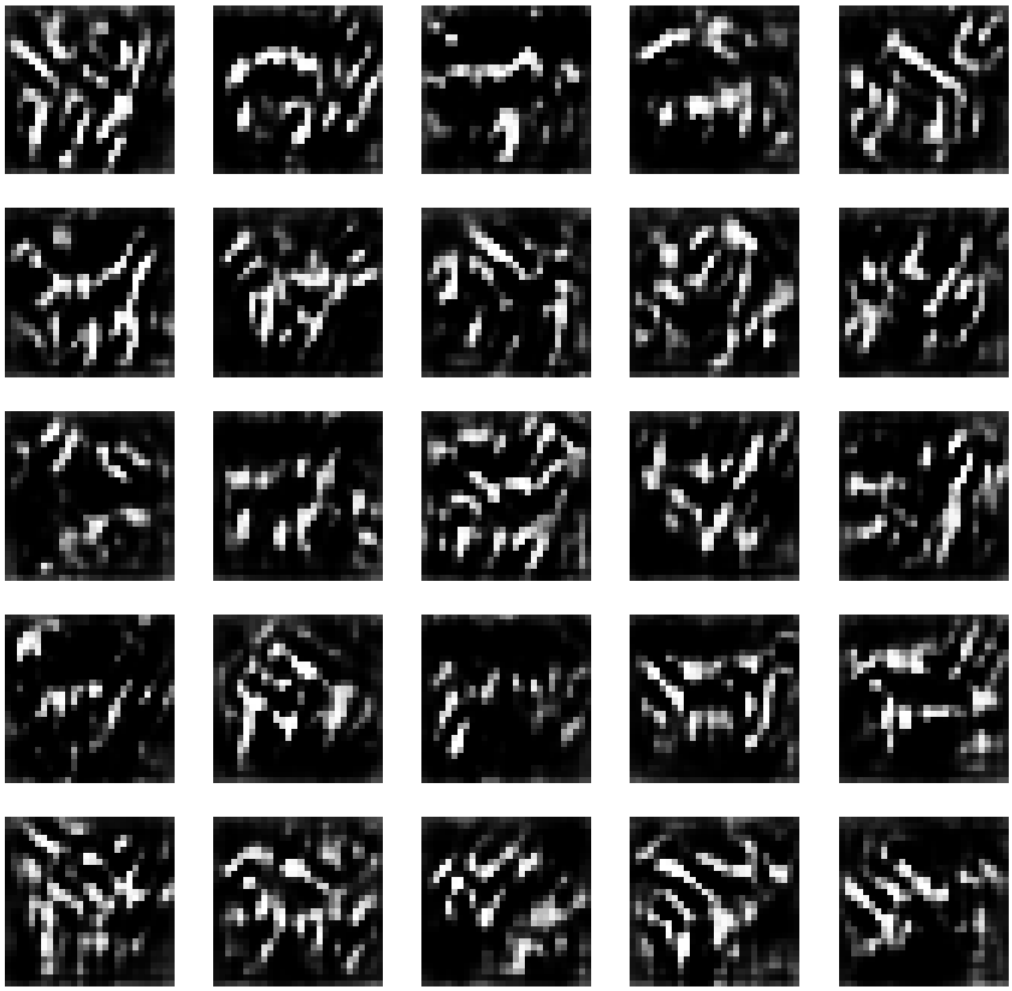
```
from IPython.display import Image as IPImage
from IPython.display import display_png

display_png(IPImage('run/gan/0001_camel/images/sample_0.png'))
```



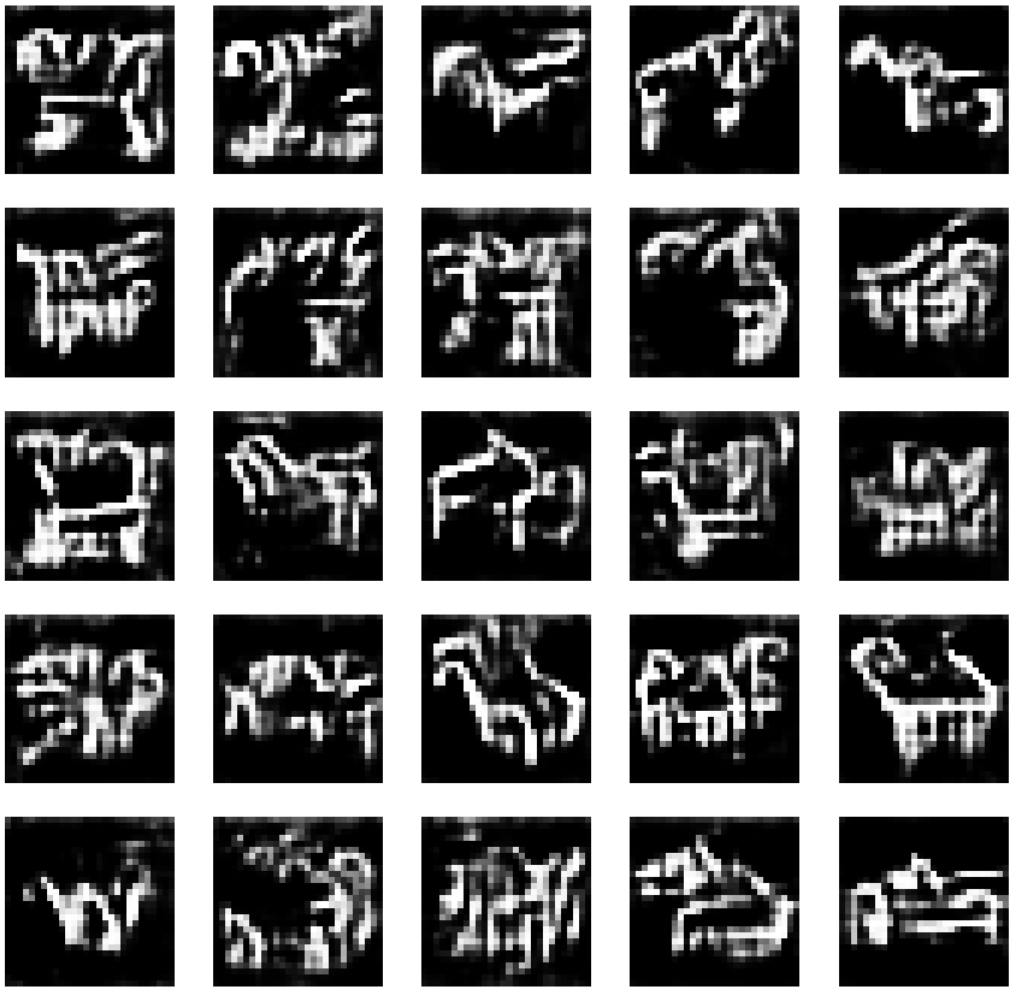
```
display_png(IPImage('run/gan/0001_camel/images/sample_100.png'))
```



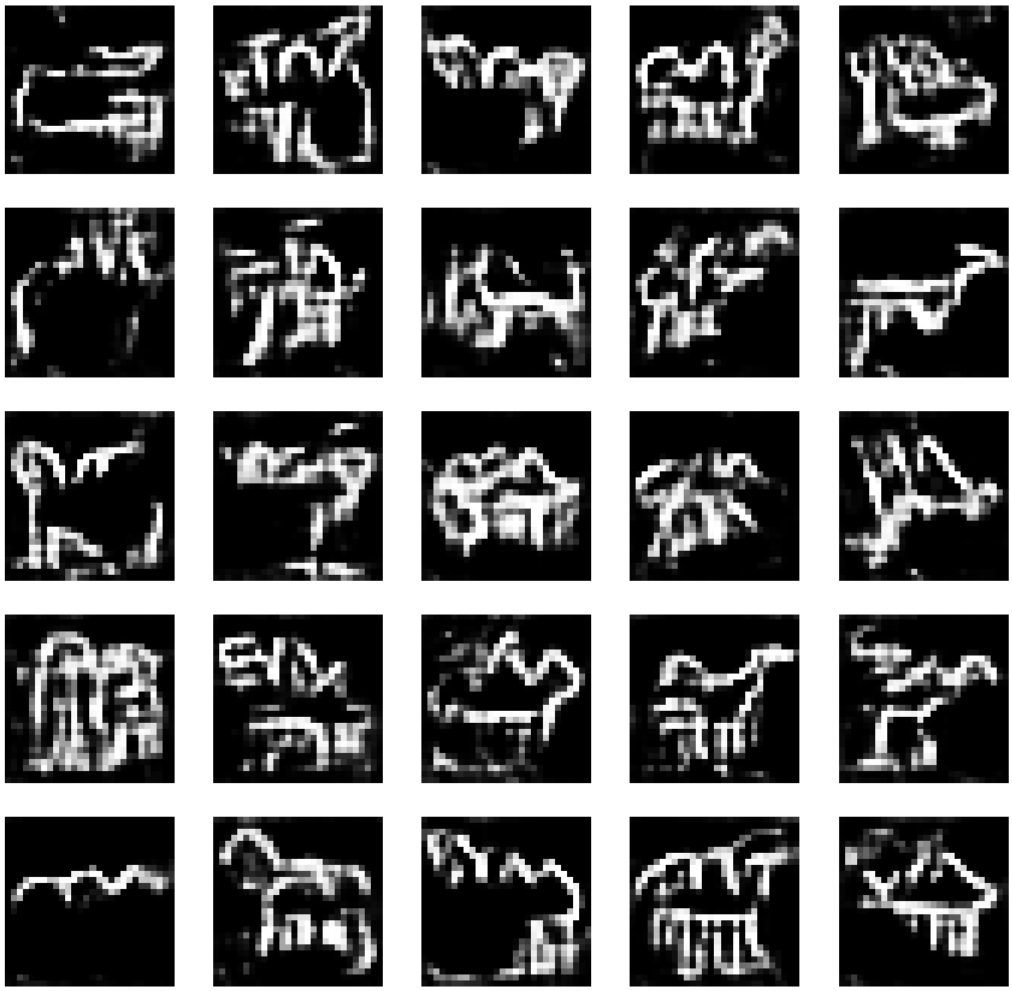
```
display_png(IPImage('run/gan/0001_camel/images/sample_200.png'))
```



display_png(IPImage('run/gan/0001_camel/images/sample_300.png'))



display_png(IPImage('run/gan/0001_camel/images/sample_400.png'))

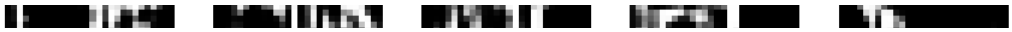


```
display_png(IPImage('run/gan/0001_camel/images/sample_500.png'))
```

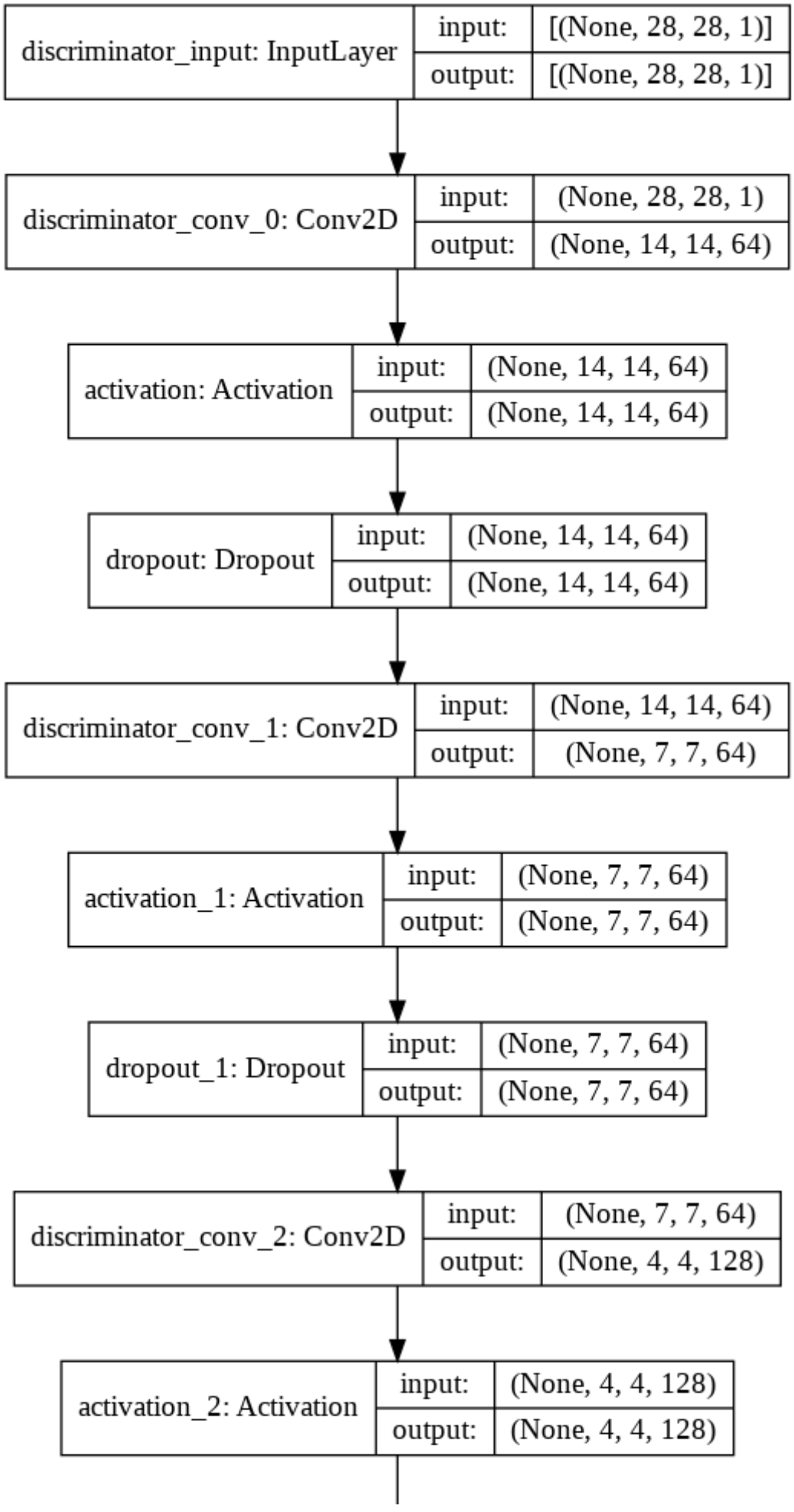
display_png(IPImage('run/gan/0001_camel/images/sample_595.png'))

```
!ls run/gan/0001_camel/viz
```

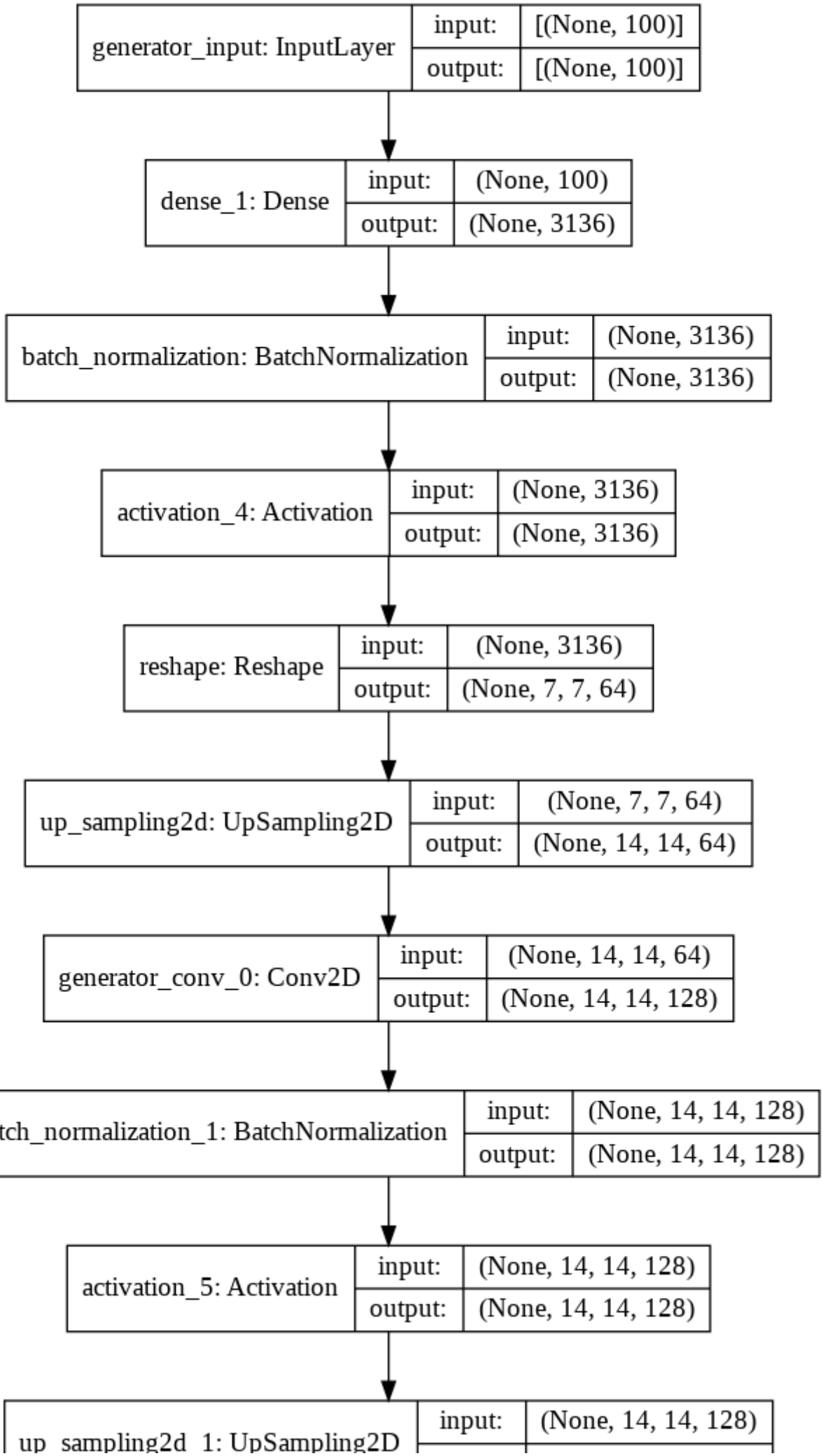
```
discriminator.png  generator.png  model.png
```

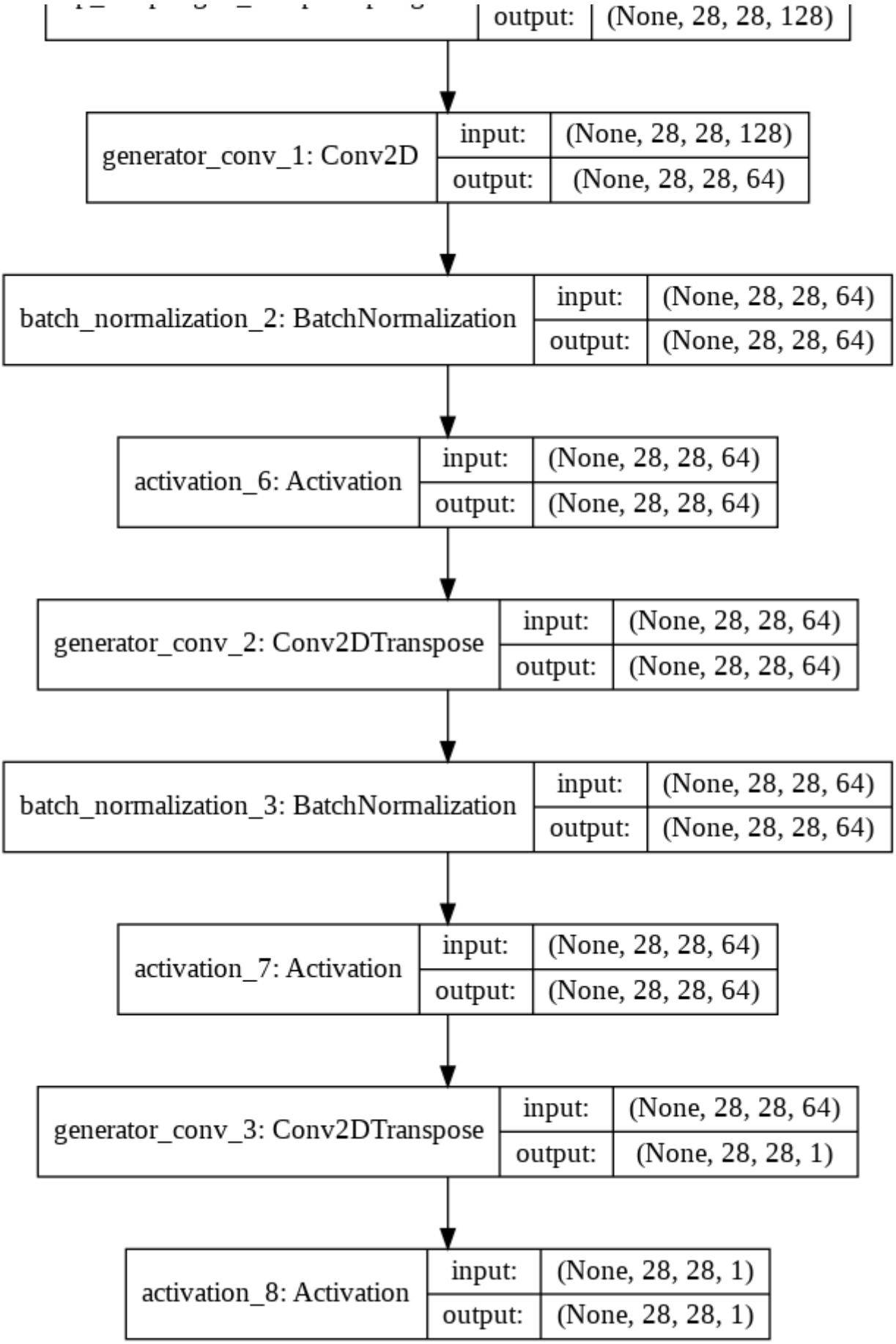


```
display_png(IPImage('run/gan/0001_camel/viz/discriminator.png'))
```

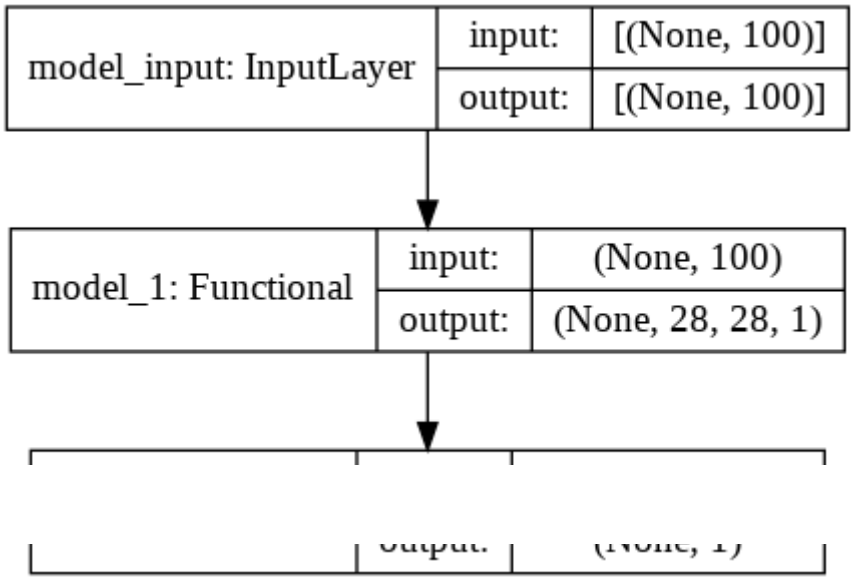



```
display_png(IPImage('run/gan/0001_camel/viz/generator.png'))
```


generator_input: InputLayer
input: [(None, 100)]
output: [(None, 100)]
dense_1: Dense
input: (None, 100)
output: (None, 3136)
batch_normalization: BatchNormalization
input: (None, 3136)
output: (None, 3136)
activation_4: Activation
input: (None, 3136)
output: (None, 3136)
reshape: Reshape
input: (None, 3136)
output: (None, 7, 7, 64)
up_sampling2d: UpSampling2D
input: (None, 7, 7, 64)
output: (None, 14, 14, 64)
generator_conv_0: Conv2D
input: (None, 14, 14, 64)
output: (None, 14, 14, 128)
batch_normalization_1: BatchNormalization
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
activation_5: Activation
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
up_sampling2d_1: UpSampling2D
input: (None, 14, 14, 128)




```
display_png(IPImage('run/gan/0001_camel/viz/model.png'))
```


model_input: InputLayer
input:
[(None, 100)]
output:
[(None, 100)]
model_1: Functional
input:
(None, 100)
output:
(None, 28, 28, 1)