Semi-Supervised Conditional GAN (SCGAN)

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import json

from keras import backend as K

from keras.datasets import mnist
from keras.layers import (Activation, BatchNormalization, Concatenate, Dense,
                          Dropout, Flatten, Input, Lambda, Reshape, Embedding,
                          Multiply)
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.models import Model, Sequential
from keras.optimizers import Adam
from keras.utils import to_categorical
```

Dataset

```
class Dataset:
    def __init__(self, num_labeled):

        # Number labeled examples to use for training
        self.num_labeled = num_labeled

        # Load the MNIST dataset
        (self.x_train, self.y_train), (self.x_test,
                                       self.y_test) = mnist.load_data()

        def preprocess_imgs(x):
            # Rescale [0, 255] grayscale pixel values to [-1, 1]
            x = (x.astype(np.float32) - 127.5) / 127.5
            # Expand image dimensions to width x height x channels
            x = np.expand_dims(x, axis=3)
            return x

        def preprocess_labels(y):
            return y.reshape(-1, 1)

        # Training data
        self.x_train = preprocess_imgs(self.x_train)
        self.y_train = preprocess_labels(self.y_train)

        # Testing data
        self.x_test = preprocess_imgs(self.x_test)
        self.y_test = preprocess_labels(self.y_test)

    def batch_labeled(self, batch_size):
        # Get a random batch of labeled images and their labels
        idx = np.random.randint(0, self.num_labeled, batch_size)
        imgs = self.x_train[idx]
        labels = self.y_train[idx]
        return imgs, labels

    def batch_unlabeled(self, batch_size):
        # Get a random batch of unlabeled images
        idx = np.random.randint(self.num_labeled, self.x_train.shape[0],
                                batch_size)
        imgs = self.x_train[idx]
        return imgs
```

```
    def training_set(self):
        x_train = self.x_train[range(self.num_labeled)]
        y_train = self.y_train[range(self.num_labeled)]
        return x_train, y_train

    def test_set(self):
        return self.x_test, self.y_test


# Number of labeled examples to use (rest will be used as unlabeled)
num_labeled = 100

dataset = Dataset(num_labeled)


(x_train, y_train), (x_test, y_test) = mnist.load_data()


print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-573562366363> in <module>()
----> 1 print(x_train.shape)
      2 print(y_train.shape)
      3 print(x_test.shape)
      4 print(y_test.shape)

NameError: name 'x_train' is not defined
```

SEARCH STACK OVERFLOW

Semi-Supervied GAN

```
img_rows = 28
img_cols = 28
channels = 1

# Input image dimensions
img_shape = (img_rows, img_cols, channels)

# Size of the noise vector, used as input to the Generator
z_dim = 100

# Number of classes in the dataset
num_classes = 10
```

Generator

```
def build_generator(z_dim):

    model = Sequential()

    # Reshape input into 7x7x256 tensor via a fully connected layer
    model.add(Dense(256 * 7 * 7, input_dim=z_dim))
    model.add(Reshape((7, 7, 256)))
```

```
    # Transposed convolution layer, from 7x7x256 into 14x14x128 tensor
    model.add(Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'))

    # Batch normalization
    model.add(BatchNormalization())

    # Leaky ReLU activation
    model.add(LeakyReLU(alpha=0.01))

    # Transposed convolution layer, from 14x14x128 to 14x14x64 tensor
    model.add(Conv2DTranspose(64, kernel_size=3, strides=1, padding='same'))

    # Batch normalization
    model.add(BatchNormalization())

    # Leaky ReLU activation
    model.add(LeakyReLU(alpha=0.01))

    # Transposed convolution layer, from 14x14x64 to 28x28x1 tensor
    model.add(Conv2DTranspose(1, kernel_size=3, strides=2, padding='same'))

    # Output layer with tanh activation
    model.add(Activation('tanh'))

    z = Input(shape=(z_dim, ))
    label = Input(shape=(num_classes, ), dtype='float32')
    label_embedding = Dense(z_dim, input_dim=num_classes)(label)
    print(label_embedding)
    joined_representation = Multiply()([z, label_embedding])
    print(joined_representation)
    conditioned_img = model(joined_representation)

    model = Model([z, label], conditioned_img)

    return model


build_generator(z_dim).summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0831 08:53:28.851006 140350853937024 deprecation_wrapper.py:119] From /usr/local/lib/python3.

W0831 08:53:28.867646 140350853937024 deprecation_wrapper.py:119] From /usr/local/lib/python3.

W0831 08:53:28.872331 140350853937024 deprecation_wrapper.py:119] From /usr/local/lib/python3.

Tensor("dense_1/BiasAdd:0", shape=(?, 100), dtype=float32)
Tensor("multiply_1/mul:0", shape=(?, 100), dtype=float32)
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-11-4949f1faba17> in <module>()
----> 1 build_generator(z_dim).summary()

<ipython-input-10-f95a9192110f> in build_generator(z_dim)
      7     joined_representation = Multiply()([z, label_embedding])
      8     print(joined_representation)
----> 9     conditioned_img = model(joined_representation)
     10
     11     model = Sequential()

UnboundLocalError: local variable 'model' referenced before assignment
```

SEARCH STACK OVERFLOW

```
batch_size=30
imgs, labels = dataset.batch_labeled(batch_size)
# labels = to_categorical(labels, num_classes=num_classes)
labels = np.random.randint(0, num_classes, batch_size).reshape(-1, 1)
print(labels.shape)
print(labels)
```

```
(30, 1)
[[8]
 [5]
 [2]
 [0]
 [1]
 [5]
 [7]
 [7]
 [6]
 [7]
 [4]
 [4]
 [1]
 [7]
 [0]
 [9]
 [9]
 [0]
 [7]
 [0]
 [8]
 [0]
 [1]
 [4]
 [8]
 [9]
 [8]
 [3]
 [0]
```

Discriminator

```
def build_discriminator_net(img_shape):

    model = Sequential()

    # Convolutional layer, from 28x28x1 into 14x14x32 tensor
    model.add(
        Conv2D(32,
               kernel_size=3,
               strides=2,
               input_shape=img_shape,
               padding='same'))

    # Leaky ReLU activation
    model.add(LeakyReLU(alpha=0.01))

    # Convolutional layer, from 14x14x32 into 7x7x64 tensor
    model.add(
        Conv2D(64,
               kernel_size=3,
               strides=2,
               input_shape=img_shape,
               padding='same'))

    # Batch normalization
    model.add(BatchNormalization())

    # Leaky ReLU activation
    model.add(LeakyReLU(alpha=0.01))

    # Convolutional layer, from 7x7x64 tensor into 3x3x128 tensor
    model.add(
        Conv2D(128,
               kernel_size=3,
               strides=2,
               input_shape=img_shape,
               padding='same'))
```

```
    # Batch normalization
    model.add(BatchNormalization())

    # Leaky ReLU activation
    model.add(LeakyReLU(alpha=0.01))

    # Droupout
    model.add(Dropout(0.5))

    # Flatten the tensor
    model.add(Flatten())

    # Fully connected layer with num_classes neurons
    model.add(Dense(num_classes))

    return model


def build_discriminator_supervised(discriminator_net):

    model = Sequential()

    model.add(discriminator_net)

    # Softmax activation, giving predicted probability distribution over the real classes
    model.add(Activation('softmax'))

    return model


def build_discriminator_unsupervised(discriminator_net):

    model = Sequential()

    model.add(discriminator_net)

    def predict(x):
        # Transform distribution over real classes into a binary real-vs-fake probability
        prediction = 1.0 - (1.0 /
                            (K.sum(K.exp(x), axis=-1, keepdims=True) + 1.0))
        return prediction

    # 'Real-vs-fake' output neuron defined above
    model.add(Lambda(predict))

    return model
```

Build the Model

```
def build_gan(generator, discriminator):

#     model = Sequential()

    # Combined Generator -> Discriminator model
#     model.add(generator)
#     model.add(discriminator)

    z = Input(shape=(z_dim, ))
    label = Input(shape=(num_classes, ))
    img = generator([z, label])
    output = discriminator(img)
    model = Model([z, label], output)

    return model


generator = build_generator(z_dim)
discriminator_unsupervised.trainable = False
```

```
gan = build_gan(generator, discriminator_unsupervised)
gan.summary()
gan.compile(loss='binary_crossentropy', optimizer=Adam())
```

```
Tensor("dense_7/BiasAdd:0", shape=(?, 100), dtype=float32)
Tensor("multiply_5/mul:0", shape=(?, 100), dtype=float32)
Model: "model_2"
```

Layer (type)	Output Shape	Param #	Connected to
input_11 (InputLayer)	(None, 100)	0	
input_12 (InputLayer)	(None, 10)	0	
model_1 (Model)	(None, 28, 28, 1)	1638221	input_11[0][0] input_12[0][0]
sequential_3 (Sequential)	(None, 1)	113930	model_1[1][0]

```
Total params: 1,752,151
Trainable params: 1,637,837
Non-trainable params: 114,314
```

Discriminator

```
# Core Discriminator network:
# These layers are shared during supervised and unsupervised training
discriminator_net = build_discriminator_net(img_shape)

# Build & compile the Discriminator for supervised training
discriminator_supervised = build_discriminator_supervised(discriminator_net)
discriminator_supervised.compile(loss='categorical_crossentropy',
                                 metrics=['accuracy'],
                                 optimizer=Adam())

# Build & compile the Discriminator for unsupervised training
discriminator_unsupervised = build_discriminator_unsupervised(discriminator_net)
discriminator_unsupervised.compile(loss='binary_crossentropy',
                                   optimizer=Adam())
```

Generator

```
# Build the Generator
generator = build_generator(z_dim)

# Keep Discriminator's parameters constant for Generator training
discriminator_unsupervised.trainable = False

# Build and compile GAN model with fixed Discriminator to train the Generator
# Note that we are using the Discriminator version with unsupervised output
gan = build_gan(generator, discriminator_unsupervised)
gan.compile(loss='binary_crossentropy', optimizer=Adam())
```

```
Tensor("dense_10/BiasAdd:0", shape=(?, 100), dtype=float32)
Tensor("multiply_6/mul:0", shape=(?, 100), dtype=float32)
```

Training

```
supervised_losses = []
unsupervised_losses = []
g_losses = []
iteration_checkpoints = []


def train(iterations, batch_size, save_interval):

    # Labels for real images: all ones
    real = np.ones((batch_size, 1))

    # Labels for fake images: all zeros
    fake = np.zeros((batch_size, 1))

    for iteration in range(iterations):

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

        # Get labeled examples
        imgs, labels = dataset.batch_labeled(batch_size)

        # One-hot encode labels
        labels = to_categorical(labels, num_classes=num_classes)

        # Get unlabeled examples
        imgs_unlabeled = dataset.batch_unlabeled(batch_size)

        # Generate a batch of fake images
        z = np.random.normal(0, 1, (batch_size, z_dim))
        fake_labels = np.random.randint(0, num_classes, batch_size).reshape(-1, 1)
        fake_labels = to_categorical(fake_labels, num_classes=num_classes)
        gen_imgs = generator.predict([z, fake_labels])

        # Train on real labeled examples
        d_loss_supervised, accuracy = discriminator_supervised.train_on_batch(imgs, labels)

        # Train on real unlabeled examples
        d_loss_real = discriminator_unsupervised.train_on_batch(imgs_unlabeled, real)

        # Train on fake examples
        d_loss_fake = discriminator_unsupervised.train_on_batch(gen_imgs, fake)

        d_loss_unsupervised = 0.5 * np.add(d_loss_real, d_loss_fake)

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

        # Generate a batch of fake images
        z = np.random.normal(0, 1, (batch_size, z_dim))
        gen_imgs = generator.predict([z, labels])

        # Train Generator
        g_loss = gan.train_on_batch([z,labels], np.ones((batch_size, 1)))

        if (iteration + 1) % save_interval == 0:

            # Save Discriminator supervised classification loss to be plotted after training
            supervised_losses.append(d_loss_supervised)
            unsupervised_losses.append(d_loss_unsupervised)
            g_losses.append(g_loss)

            iteration_checkpoints.append(iteration + 1)

            # Output training progress
            print(
                "%d [D loss supervised: %.4f, acc.: %.2f%%] [D loss unsupervised: %.4f] [G loss: %f
                % (iteration + 1, d_loss_supervised, 100 * accuracy,
                   d_loss_unsupervised, g_loss))
```

```
discriminator_supervised.save("./models/discriminator_supervised-" + str(iteration+1) +

with open('./losses/supervised_losses.json', 'w') as json_file:
      json.dump(str(supervised_losses), json_file)
with open('./losses/unsupervised_losses.json', 'w') as json_file:
      json.dump(str(unsupervised_losses), json_file)
with open('./losses/g_losses.json', 'w') as json_file:
      json.dump(str(g_losses), json_file)
```

Train the Model and Inspect Output

Note that the 'Discrepancy between trainable weights and collected trainable' warning from Keras is exp
design: The Generator's trainable parameters are intentionally held constant during Discriminator training, and vi

```
# Set hyperparameters
iterations = 8000
batch_size = 32
save_interval = 100

# Train the SGAN for the specified number of iterations
train(iterations, batch_size, save_interval)
```

```
1200 [D loss supervised: 0.0015, acc.: 100.00%] [D loss unsupervised: 0.2486] [G loss: 3.6053
1300 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.2637] [G loss: 4.2859
1400 [D loss supervised: 0.0033, acc.: 100.00%] [D loss unsupervised: 0.6150] [G loss: 2.1298
1500 [D loss supervised: 0.0021, acc.: 100.00%] [D loss unsupervised: 0.0913] [G loss: 4.2499
1600 [D loss supervised: 0.0009, acc.: 100.00%] [D loss unsupervised: 0.4089] [G loss: 3.8748
1700 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.2292] [G loss: 4.2289
1800 [D loss supervised: 0.0011, acc.: 100.00%] [D loss unsupervised: 0.1458] [G loss: 4.2190
1900 [D loss supervised: 0.0045, acc.: 100.00%] [D loss unsupervised: 0.3515] [G loss: 3.2169
2000 [D loss supervised: 0.0021, acc.: 100.00%] [D loss unsupervised: 0.0664] [G loss: 5.0107
2100 [D loss supervised: 0.0018, acc.: 100.00%] [D loss unsupervised: 0.1858] [G loss: 4.2539
2200 [D loss supervised: 0.0018, acc.: 100.00%] [D loss unsupervised: 0.2054] [G loss: 3.6670
2300 [D loss supervised: 0.0007, acc.: 100.00%] [D loss unsupervised: 0.3538] [G loss: 3.0598
2400 [D loss supervised: 0.0053, acc.: 100.00%] [D loss unsupervised: 0.6273] [G loss: 1.5188
2500 [D loss supervised: 0.0008, acc.: 100.00%] [D loss unsupervised: 0.1564] [G loss: 3.0242
2600 [D loss supervised: 0.0015, acc.: 100.00%] [D loss unsupervised: 0.1453] [G loss: 6.1552
2700 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.0859] [G loss: 5.1901
2800 [D loss supervised: 0.0018, acc.: 100.00%] [D loss unsupervised: 0.0429] [G loss: 6.2016
2900 [D loss supervised: 0.0015, acc.: 100.00%] [D loss unsupervised: 0.2928] [G loss: 1.9902
3000 [D loss supervised: 0.0007, acc.: 100.00%] [D loss unsupervised: 0.0690] [G loss: 3.4524
3100 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.5125] [G loss: 2.4579
3200 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.1085] [G loss: 3.5221
3300 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.2102] [G loss: 3.2897
3400 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.0680] [G loss: 4.0260
3500 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.1160] [G loss: 6.3541
3600 [D loss supervised: 0.0009, acc.: 100.00%] [D loss unsupervised: 0.0822] [G loss: 4.8990
3700 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.2399] [G loss: 3.9845
3800 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.2292] [G loss: 2.9913
3900 [D loss supervised: 0.0012, acc.: 100.00%] [D loss unsupervised: 0.2023] [G loss: 4.9069
4000 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.0705] [G loss: 3.3226
4100 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.2168] [G loss: 4.0503
4200 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.0688] [G loss: 5.4686
4300 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.1774] [G loss: 3.0140
4400 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.1466] [G loss: 7.4096
4500 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.3313] [G loss: 6.0915
4600 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.5807] [G loss: 3.4651
4700 [D loss supervised: 0.0008, acc.: 100.00%] [D loss unsupervised: 0.1103] [G loss: 3.9679
4800 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.3319] [G loss: 3.1179
4900 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.1431] [G loss: 5.5641
5000 [D loss supervised: 0.0025, acc.: 100.00%] [D loss unsupervised: 0.2131] [G loss: 5.5211
5100 [D loss supervised: 0.0007, acc.: 100.00%] [D loss unsupervised: 0.2927] [G loss: 5.5090
5200 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.4138] [G loss: 4.0514
5300 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.1114] [G loss: 5.2649
5400 [D loss supervised: 0.0005, acc.: 100.00%] [D loss unsupervised: 0.1025] [G loss: 5.3153
5500 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.1309] [G loss: 5.5445
5600 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.3039] [G loss: 4.2793
5700 [D loss supervised: 0.0006, acc.: 100.00%] [D loss unsupervised: 0.1257] [G loss: 2.6876
5800 [D loss supervised: 0.0010, acc.: 100.00%] [D loss unsupervised: 0.2295] [G loss: 3.6816
5900 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.0804] [G loss: 5.0494
6000 [D loss supervised: 0.0004, acc.: 100.00%] [D loss unsupervised: 0.1100] [G loss: 5.9021
6100 [D loss supervised: 0.0018, acc.: 100.00%] [D loss unsupervised: 0.2499] [G loss: 2.4962
6200 [D loss supervised: 0.0011, acc.: 100.00%] [D loss unsupervised: 0.4002] [G loss: 3.2409
6300 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.5039] [G loss: 3.1114
6400 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.3112] [G loss: 4.9353
6500 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.6173] [G loss: 1.9282
6600 [D loss supervised: 0.0023, acc.: 100.00%] [D loss unsupervised: 0.1615] [G loss: 2.9433
6700 [D loss supervised: 0.0000, acc.: 100.00%] [D loss unsupervised: 0.1107] [G loss: 4.1170
6800 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.1711] [G loss: 3.2262
6900 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.3522] [G loss: 3.6554
```

```
7000 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.3426] [G loss: 2.2260
7100 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.1917] [G loss: 4.3175
7200 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.3234] [G loss: 2.9108
7300 [D loss supervised: 0.0003, acc.: 100.00%] [D loss unsupervised: 0.5501] [G loss: 3.2072
7400 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.4792] [G loss: 4.1104
7500 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.1507] [G loss: 4.5675
7600 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.2958] [G loss: 3.4749
7700 [D loss supervised: 0.0007, acc.: 100.00%] [D loss unsupervised: 0.4151] [G loss: 2.3766
7800 [D loss supervised: 0.0001, acc.: 100.00%] [D loss unsupervised: 0.3412] [G loss: 1.8800
7900 [D loss supervised: 0.0000, acc.: 100.00%] [D loss unsupervised: 0.3039] [G loss: 2.4744
8000 [D loss supervised: 0.0002, acc.: 100.00%] [D loss unsupervised: 0.8170] [G loss: 4.2713
```

```
%ls models
```

```
discriminator_supervised-2000.h5  dis_super-2000.h5
discriminator_supervised-200.h5   dis_super-200.h5
discriminator_supervised-2100.h5  dis_super-2100.h5
discriminator_supervised-2200.h5  dis_super-2200.h5
discriminator_supervised-2300.h5  dis_super-2300.h5
discriminator_supervised-2400.h5  dis_super-2400.h5
discriminator_supervised-2500.h5  dis_super-2500.h5
discriminator_supervised-2600.h5  dis_super-2600.h5
discriminator_supervised-2700.h5  dis_super-2700.h5
discriminator_supervised-2800.h5  dis_super-2800.h5
discriminator_supervised-2900.h5  dis_super-2900.h5
discriminator_supervised-3000.h5  dis_super-3000.h5
discriminator_supervised-300.h5   dis_super-300.h5
discriminator_supervised-3100.h5  dis_super-3100.h5
discriminator_supervised-3200.h5  dis_super-3200.h5
discriminator_supervised-3300.h5  dis_super-3300.h5
discriminator_supervised-3400.h5  dis_super-3400.h5
discriminator_supervised-3500.h5  dis_super-3500.h5
discriminator_supervised-3600.h5  dis_super-3600.h5
discriminator_supervised-3700.h5  dis_super-3700.h5
discriminator_supervised-3800.h5  dis_super-3800.h5
discriminator_supervised-3900.h5  dis_super-3900.h5
discriminator_supervised-4000.h5  dis_super-4000.h5
discriminator_supervised-400.h5   dis_super-400.h5
discriminator_supervised-4100.h5  dis_super-4100.h5
discriminator_supervised-4200.h5  dis_super-4200.h5
discriminator_supervised-4300.h5  dis_super-4300.h5
discriminator_supervised-4400.h5  dis_super-4400.h5
discriminator_supervised-4500.h5  dis_super-4500.h5
discriminator_supervised-4600.h5  dis_super-4600.h5
discriminator_supervised-4700.h5  dis_super-4700.h5
discriminator_supervised-4800.h5  dis_super-4800.h5
discriminator_supervised-4900.h5  dis_super-4900.h5
discriminator_supervised-5000.h5  dis_super-5000.h5
discriminator_supervised-500.h5   dis_super-500.h5
discriminator_supervised-5100.h5  dis_super-5100.h5
discriminator_supervised-5200.h5  dis_super-5200.h5
discriminator_supervised-5300.h5  dis_super-5300.h5
discriminator_supervised-5400.h5  dis_super-5400.h5
discriminator_supervised-5500.h5  dis_super-5500.h5
discriminator_supervised-5600.h5  dis_super-5600.h5
discriminator_supervised-5700.h5  dis_super-5700.h5
discriminator_supervised-5800.h5  dis_super-5800.h5
discriminator_supervised-5900.h5  dis_super-5900.h5
discriminator_supervised-6000.h5  dis_super-6000.h5
discriminator_supervised-600.h5   dis_super-600.h5
discriminator_supervised-6100.h5  dis_super-6100.h5
discriminator_supervised-6200.h5  dis_super-6200.h5
discriminator_supervised-6300.h5  dis_super-6300.h5
discriminator_supervised-6400.h5  dis_super-6400.h5
discriminator_supervised-6500.h5  dis_super-6500.h5
discriminator_supervised-6600.h5  dis_super-6600.h5
discriminator_supervised-6700.h5  dis_super-6700.h5
discriminator_supervised-6800.h5  dis_super-6800.h5
discriminator_supervised-6900.h5  dis_super-6900.h5
discriminator_supervised-7000.h5  dis_super-7000.h5
discriminator_supervised-700.h5   dis_super-700.h5
discriminator_supervised-7100.h5  dis_super-7100.h5
```

```
discriminator_supervised-7200.h5  dis_super-7200.h5
discriminator_supervised-7300.h5  dis_super-7300.h5
discriminator_supervised-7400.h5  dis_super-7400.h5
discriminator_supervised-7500.h5  dis_super-7500.h5
discriminator_supervised-7600.h5  dis_super-7600.h5
discriminator_supervised-7700.h5  dis_super-7700.h5
discriminator_supervised-7800.h5  dis_super-7800.h5
discriminator_supervised-7900.h5  dis_super-7900.h5
discriminator_supervised-8000.h5  dis_super-8000.h5
discriminator_supervised-800.h5   dis_super-800.h5
discriminator_supervised-900.h5   dis_super-900.h5
```

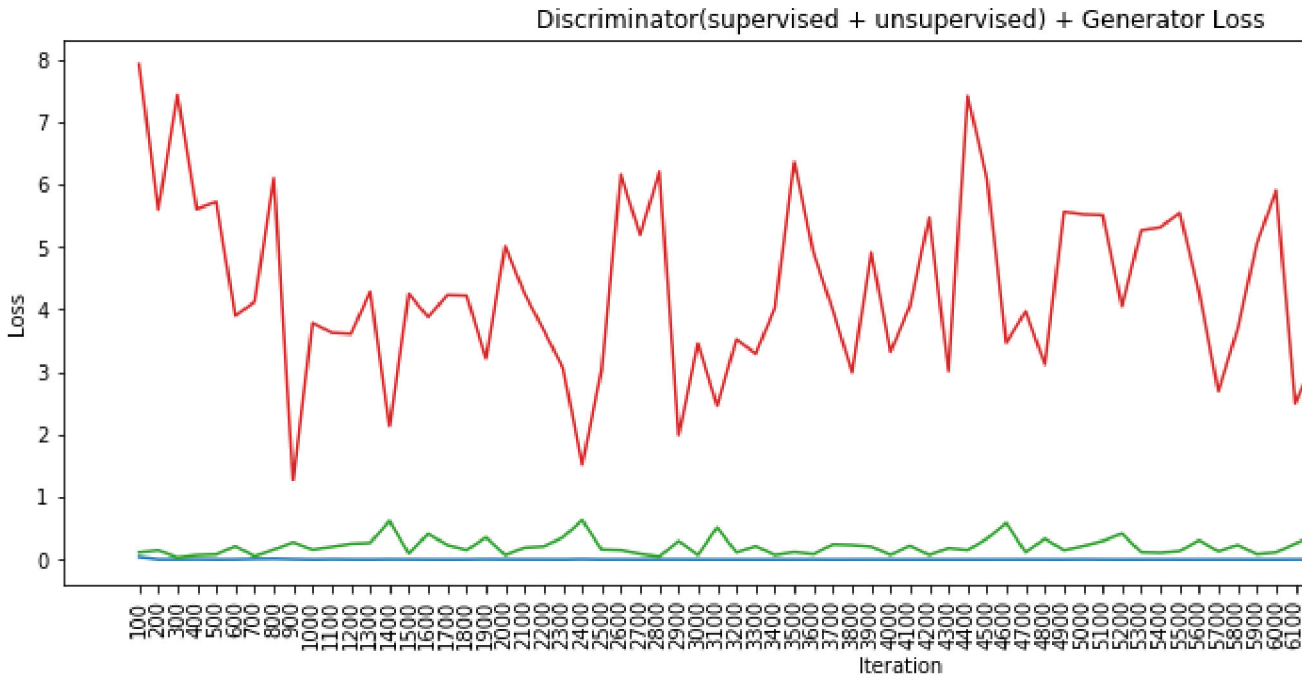
```
supervised_losses = np.array(supervised_losses)
unsupervised_losses = np.array(unsupervised_losses)
g_losses = np.array(g_losses)

# Plot Discriminator supervised loss
plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, supervised_losses, label="Discriminator supervised loss", color='ta
plt.plot(iteration_checkpoints, unsupervised_losses, label="Discriminator unsupervised loss", color
plt.plot(iteration_checkpoints, g_losses, label="Generator supervised loss", color='tab:red')

plt.xticks(iteration_checkpoints, rotation=90)

plt.title("Discriminator(supervised + unsupervised) + Generator Loss")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa5789c2f60>
```




```
print(supervised_losses[-10:])
```

```
[0.00017288796, 8.089666e-05, 0.00030267364, 0.00014982608, 0.0002316281, 9.00905e-05, 0.0007
```

SGAN Classifier – Training and Test Accuracy

```
x, y = dataset.training_set()
y = to_categorical(y, num_classes=num_classes)

# Compute classification accuracy on the training set
_, accuracy = discriminator_supervised.evaluate(x, y)
print("Training Accuracy: %.2f%%" % (100 * accuracy))
```

```
100/100 [==============================] - 0s 139us/step
Training Accuracy: 100.00%
```

```
x, y = dataset.test_set()
y = to_categorical(y, num_classes=num_classes)

# Compute classification accuracy on the test set
_, accuracy = discriminator_supervised.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
10000/10000 [==============================] - 1s 94us/step
Test Accuracy: 84.15%
```

Fully-Supervised Classifier

```
# Fully supervised classifier with the same network architecture as the SGAN Discriminator
mnist_classifier = build_discriminator_supervised(build_discriminator_net(img_shape))
mnist_classifier.compile(loss='categorical_crossentropy',
                         metrics=['accuracy'],
                         optimizer=Adam())
```

```
imgs, labels = dataset.training_set()

# One-hot encode labels
labels = to_categorical(labels, num_classes=num_classes)

# Train the classifier
training = mnist_classifier.fit(x=imgs,
                                y=labels,
                                batch_size=32,
                                epochs=350,
                                verbose=1)
losses = training.history['loss']
accuracies = training.history['acc']
```

```
Epoch 24/350
100/100 [==============================] - 0s 254us/step - loss: 0.0076 - acc: 1.0000
Epoch 25/350
100/100 [==============================] - 0s 252us/step - loss: 0.0078 - acc: 1.0000
Epoch 26/350
100/100 [==============================] - 0s 254us/step - loss: 0.0083 - acc: 1.0000
Epoch 27/350
100/100 [==============================] - 0s 251us/step - loss: 0.0076 - acc: 1.0000
Epoch 28/350
100/100 [==============================] - 0s 248us/step - loss: 0.0067 - acc: 1.0000
Epoch 29/350
100/100 [==============================] - 0s 260us/step - loss: 0.0067 - acc: 1.0000
Epoch 30/350
100/100 [==============================] - 0s 258us/step - loss: 0.0069 - acc: 1.0000
Epoch 31/350
100/100 [==============================] - 0s 247us/step - loss: 0.0055 - acc: 1.0000
Epoch 32/350
100/100 [==============================] - 0s 236us/step - loss: 0.0049 - acc: 1.0000
Epoch 33/350
100/100 [==============================] - 0s 249us/step - loss: 0.0046 - acc: 1.0000
Epoch 34/350
100/100 [==============================] - 0s 238us/step - loss: 0.0040 - acc: 1.0000
Epoch 35/350
100/100 [==============================] - 0s 239us/step - loss: 0.0044 - acc: 1.0000
Epoch 36/350
100/100 [==============================] - 0s 232us/step - loss: 0.0042 - acc: 1.0000
Epoch 37/350
100/100 [==============================] - 0s 274us/step - loss: 0.0041 - acc: 1.0000
Epoch 38/350
100/100 [==============================] - 0s 257us/step - loss: 0.0038 - acc: 1.0000
Epoch 39/350
100/100 [==============================] - 0s 238us/step - loss: 0.0031 - acc: 1.0000
Epoch 40/350
100/100 [==============================] - 0s 253us/step - loss: 0.0022 - acc: 1.0000
Epoch 41/350
100/100 [==============================] - 0s 237us/step - loss: 0.0036 - acc: 1.0000
Epoch 42/350
100/100 [==============================] - 0s 232us/step - loss: 0.0029 - acc: 1.0000
Epoch 43/350
100/100 [==============================] - 0s 245us/step - loss: 0.0027 - acc: 1.0000
Epoch 44/350
100/100 [==============================] - 0s 255us/step - loss: 0.0022 - acc: 1.0000
Epoch 45/350
100/100 [==============================] - 0s 274us/step - loss: 0.0031 - acc: 1.0000
Epoch 46/350
100/100 [==============================] - 0s 252us/step - loss: 0.0023 - acc: 1.0000
Epoch 47/350
100/100 [==============================] - 0s 240us/step - loss: 0.0027 - acc: 1.0000
Epoch 48/350
100/100 [==============================] - 0s 254us/step - loss: 0.0021 - acc: 1.0000
Epoch 49/350
100/100 [==============================] - 0s 226us/step - loss: 0.0040 - acc: 1.0000
Epoch 50/350
100/100 [==============================] - 0s 248us/step - loss: 0.0024 - acc: 1.0000
Epoch 51/350
100/100 [==============================] - 0s 235us/step - loss: 0.0022 - acc: 1.0000
Epoch 52/350
100/100 [==============================] - 0s 240us/step - loss: 0.0022 - acc: 1.0000
```

```
Epoch 53/350
100/100 [==============================] - 0s 312us/step - loss: 0.0020 - acc: 1.0000
Epoch 54/350
100/100 [==============================] - 0s 290us/step - loss: 0.0022 - acc: 1.0000
Epoch 55/350
100/100 [==============================] - 0s 226us/step - loss: 0.0022 - acc: 1.0000
Epoch 56/350
100/100 [==============================] - 0s 255us/step - loss: 0.0017 - acc: 1.0000
Epoch 57/350
100/100 [==============================] - 0s 242us/step - loss: 0.0031 - acc: 1.0000
Epoch 58/350
100/100 [==============================] - 0s 232us/step - loss: 0.0037 - acc: 1.0000
Epoch 59/350
100/100 [==============================] - 0s 231us/step - loss: 0.0015 - acc: 1.0000
Epoch 60/350
100/100 [==============================] - 0s 233us/step - loss: 0.0019 - acc: 1.0000
Epoch 61/350
100/100 [==============================] - 0s 236us/step - loss: 0.0015 - acc: 1.0000
Epoch 62/350
100/100 [==============================] - 0s 239us/step - loss: 0.0019 - acc: 1.0000
Epoch 63/350
100/100 [==============================] - 0s 261us/step - loss: 0.0049 - acc: 1.0000
Epoch 64/350
100/100 [==============================] - 0s 291us/step - loss: 0.0020 - acc: 1.0000
Epoch 65/350
100/100 [==============================] - 0s 230us/step - loss: 0.0122 - acc: 1.0000
Epoch 66/350
100/100 [==============================] - 0s 235us/step - loss: 0.0130 - acc: 0.9900
Epoch 67/350
100/100 [==============================] - 0s 232us/step - loss: 0.0027 - acc: 1.0000
Epoch 68/350
100/100 [==============================] - 0s 229us/step - loss: 0.0024 - acc: 1.0000
Epoch 69/350
100/100 [==============================] - 0s 233us/step - loss: 0.0024 - acc: 1.0000
Epoch 70/350
100/100 [==============================] - 0s 252us/step - loss: 0.0024 - acc: 1.0000
Epoch 71/350
100/100 [==============================] - 0s 248us/step - loss: 0.0035 - acc: 1.0000
Epoch 72/350
100/100 [==============================] - 0s 233us/step - loss: 0.0022 - acc: 1.0000
Epoch 73/350
100/100 [==============================] - 0s 262us/step - loss: 0.0040 - acc: 1.0000
Epoch 74/350
100/100 [==============================] - 0s 274us/step - loss: 0.0021 - acc: 1.0000
Epoch 75/350
100/100 [==============================] - 0s 278us/step - loss: 0.0022 - acc: 1.0000
Epoch 76/350
100/100 [==============================] - 0s 245us/step - loss: 0.0021 - acc: 1.0000
Epoch 77/350
100/100 [==============================] - 0s 242us/step - loss: 0.0017 - acc: 1.0000
Epoch 78/350
100/100 [==============================] - 0s 238us/step - loss: 0.0017 - acc: 1.0000
Epoch 79/350
100/100 [==============================] - 0s 240us/step - loss: 0.0014 - acc: 1.0000
Epoch 80/350
100/100 [==============================] - 0s 239us/step - loss: 0.0010 - acc: 1.0000
Epoch 81/350
100/100 [==============================] - 0s 279us/step - loss: 0.0011 - acc: 1.0000
Epoch 82/350
```