- https://github.com/timestocome (https://github.com/timestocome)

This problem is just a slight variation on MNIST. It's a different number set and there is a labeled training set, a labeled holdout set and an unlabeled set for submitting to Kaggle.

What makes it interesting is that all 3 sets of integers have different std, means

The best way to solve this problem is to augment the data, shift and scale the training data to match the holdout and or submission set and run it through a convolutional network. Using all the 40,000 training data, augmenting it and using a standard convolutional network will score 98%

But what if that wasn't an option? Using only 4000 labeled samples of 40,000 in the set and 4000 unlabeled samples accuracy can hit 81%

Using a Semi-Supervised GAN gives 5% better accuracy than the conventional method if you only have a limited amount of labeled data

Kannada MNIST Semi-Supervised GAN

- Model small set of Kannada MNIST samples of 40,000 samples
- Holdout and Submission imgs vary from test data in std, mean

While the GAN slightly outperforms a stardard Supervised FF Conv on Test Data that differs from Training Data it's not by a large amount. Augmenting data and shifting the std and mean to match Test Data is probably a better option

Use labeled and unlabeled data from the training set (same std, mean)

n_train samples	GAN validation	GAN holdout	Supervised validation	Supervised holdout
1000	98%	72%	97%	67%
2000	98%	70%	97%	62%
3000	98%	72%	98%	70%
4000	98%	72%	97%	67%
5000	99%	71%	98%	65%
6000	99%	76%	98%	67%
8000	99%	72%	98%	67%
10000	99%	75%	98%	70%
12000	99%	75%	99%	74%

Use unlabeled holdout set as unlabeled data for GAN (different std, mean) check against unlabeled set

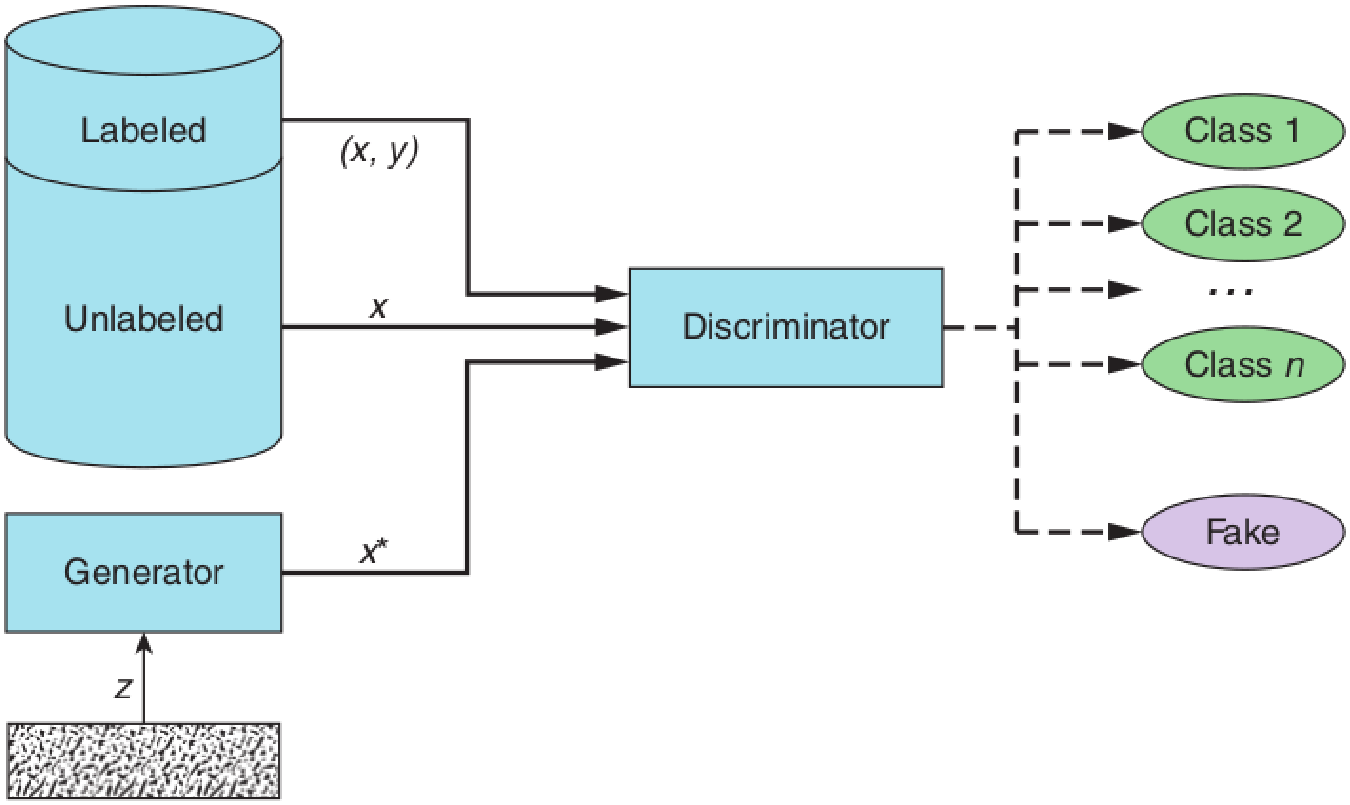
n_train samples	GAN validation	GAN holdout
1000	98%	79%
2000	98%	78%
3000	98%	%
4000	98%	80%
5000	99%	%
6000	99%	81%
8000	99%	%
10000	99%	72%
12000	99%	%

Use Submission as unlabeled, check against holdout labeled data (all sets have diff std, mean)

4000 training samples yeilds 73% accuracy

- example code and images from https://www.manning.com/books/gans-in-action (https://www.manning.com/books/gans-in-action)

- dataset https://www.kaggle.com/c/Kannada-MNIST (https://www.kaggle.com/c/Kannada-MNIST)



- Reducing the Need for Labeled Data in Generative Adversarial Networks https://ai.googleblog.com/2019/03/reducing-need-for-labeled-data-in.html (https://ai.googleblog.com/2019/03/reducing-need-for-labeled-data-in.html)
- Self-Supervised GANs via Auxiliary Rotation Loss https://arxiv.org/pdf/1811.11212.pdf (https://arxiv.org/pdf/1811.11212.pdf)
- Unsupervised Representation Learning by Predicting Image Rotations https://arxiv.org/abs/1803.07728 (https://arxiv.org/abs/1803.07728)

Kannada is a language spoken predominantly by people of Karnataka in southwestern India. The language has roughly 45 million native speakers and is written using the Kannada script.

The goal of this Kaggle competition is to use machine learning to correctly label hand-written digits written in the Kannada script. The Kannada dataset format is based on the dataset used in the original MNIST dataset where the objective was the same but the hand-written digits were Arabic numbers.

In [1]:

```
# silence is golden

import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action="ignore",category=DeprecationWarning)
warnings.filterwarnings(action="ignore",category=FutureWarning)

```

In [2]:

```
# hack to make keras work with 2*** series gpus

import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
```

In [3]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


from keras import backend as K

from keras.layers import (Activation, BatchNormalization, Concatenate, Dense
                          Dropout, Flatten, Input, Lambda, Reshape)

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, print_summary
```

```
Using TensorFlow backend.
```

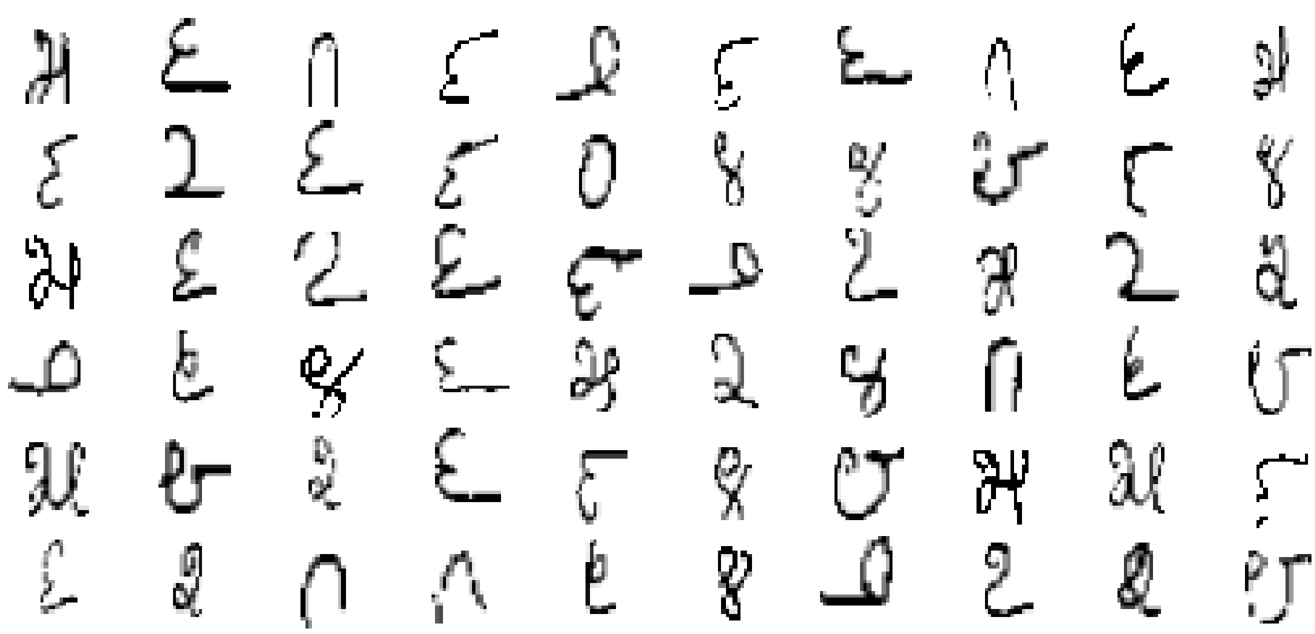
In [4]:
```
# Kannada MNIST dataset ~ 60,000 samples
# 40,000 labeled train
# 5,000 unlabeled submission
# 10,240 labeled holdout
# find labeled data, split out 100 samples for training


train = pd.read_csv('input/train.csv')
test = pd.read_csv('input/test.csv')
holdout = pd.read_csv('input/Dig-MNIST.csv')


# label, image
print('train', train.shape)
print(train.columns)


# id, image
print('test', test.shape)
print(test.columns)


# label, image
print('holdout', holdout.shape)
print(holdout.columns)

```

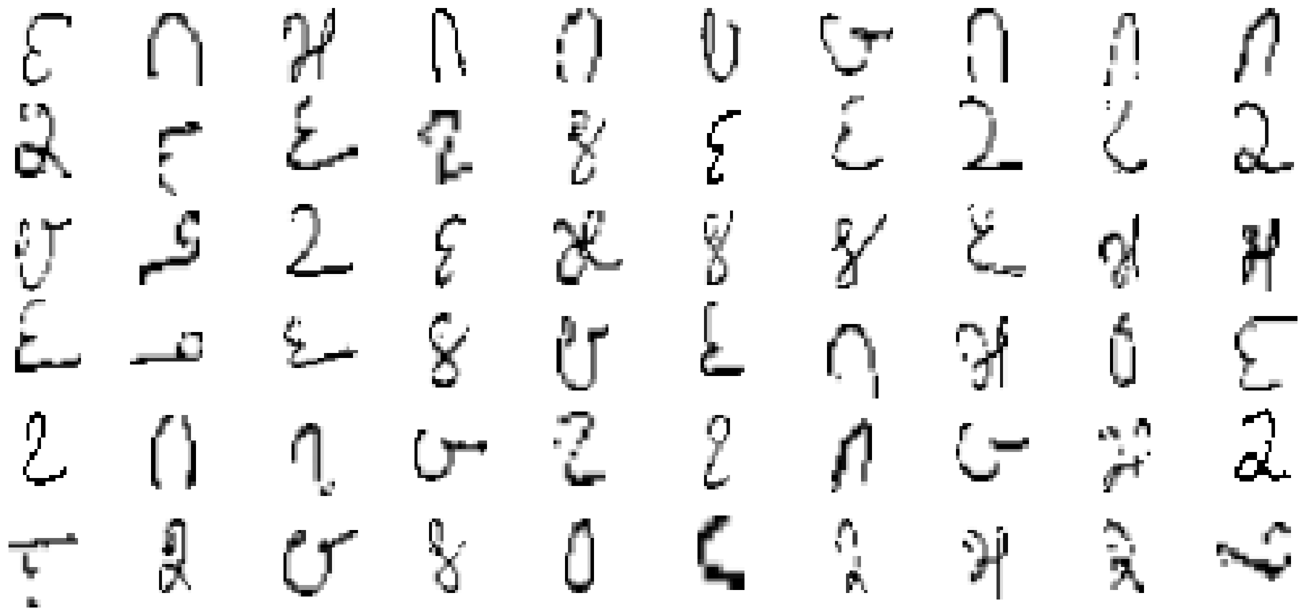
```
train (60000, 785)
Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)
test (5000, 785)
Index(['id', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)
holdout (10240, 785)
Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779',
```

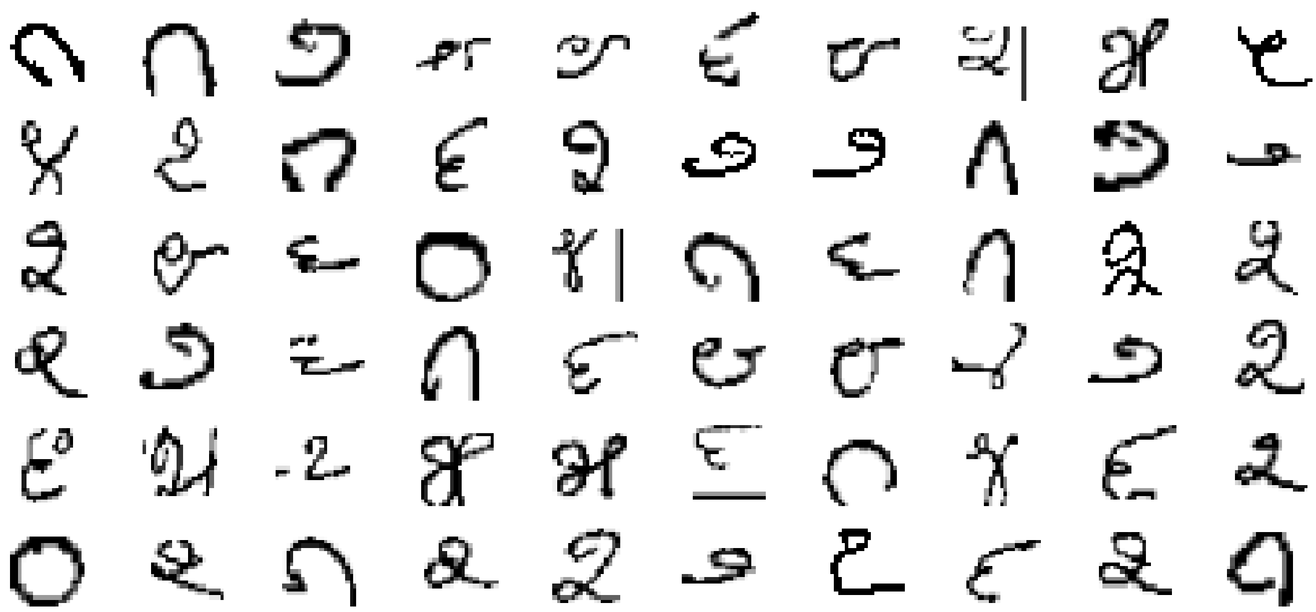
In [5]:

```python
def show_imgs(df, n):

    fig, ax = plt.subplots(n, 10)

    for i in range(n):
        for j in range(10):
            r = np.random.randint(0, len(df))
            number = df.iloc[r].values.reshape((28,28))
            ax[i][j].imshow(number, cmap=plt.cm.binary)
            ax[i][j].axis('off')

    plt.subplots_adjust(wspace=0, hspace=0)
    fig.set_figwidth(15)
    fig.set_figheight(7)
    fig.show()



print('Train')
train_imgs = train.drop(columns=['label'])
show_imgs(train_imgs, 6)

print('Test')
test_imgs = test.drop(columns=['id'])
show_imgs(test_imgs, 6)

print('Holdout')
holdout_imgs = holdout.drop(columns=['label'])
show_imgs(holdout_imgs, 6)
```

```
Train
Test
Holdout
```







In [6]:

```python
# split into x image, y label, train -> train/validate

# this is the labeled part of the training set, rest of training data is hel
num_labeled = 4000


# shuffle train set
train_s = train.sample(frac=1.)
train = train_s[0:30000]
validate = train_s[30000:40000]

print(train.shape, validate.shape)


# split into labeled and unlabeled
train_labeled = train[0:num_labeled]
train_unlabeled = train[num_labeled:len(train)]


y_train_labeled = train_labeled.label.values
x_train_labeled = train_labeled.drop(columns=['label']).values

y_train_unlabeled = train_unlabeled.label.values
x_train_unlabeled = train_unlabeled.drop(columns=['label']).values


y_validate = validate.label.values
x_validate = validate.drop(columns=['label']).values



x_submission = test.drop(columns=['id']).values

y_holdout = holdout.label.values
x_holdout = holdout.drop(columns=['label']).values

print('labeled train', x_train_labeled.shape, y_train_labeled.shape)
print('unlabeled train', x_train_unlabeled.shape, y_train_unlabeled.shape)

print('submission', x_submission.shape)
print('holdout', x_holdout.shape, y_holdout.shape)

```

```
(30000, 785) (10000, 785)
labeled train (4000, 784) (4000,)
unlabeled train (26000, 784) (26000,)
submission (5000, 784)
holdout (10240, 784) (10240,)
```

In [7]:

```
# shift and scale image data
def scale_shift(x):
    return x / 255.
    #return (x-127.5)/(2 * x.std())



x_train_unlabeled = scale_shift(x_train_unlabeled)
x_train_labeled = scale_shift(x_train_labeled)
x_validate = scale_shift(x_validate)
x_holdout = scale_shift(x_holdout)
x_submission = scale_shift(x_submission)



print(x_train_unlabeled.std(), x_train_labeled.std(), x_validate.std(), x_ho
print(x_train_unlabeled.mean(), x_train_labeled.mean(), x_validate.mean(), x_
print(x_train_unlabeled.min(), x_train_labeled.min(), x_validate.min(), x_ho
print(x_train_unlabeled.max(), x_train_labeled.max(), x_validate.max(), x_ho
```

```
0.2418371564987552 0.24257205266890997 0.24149437347731595 0.2860085202946692
0.22543569156630952
0.08234349585988249 0.08244536314525808 0.08211117997198893 0.11264889842655808
0.07261260104041618
0.0 0.0 0.0 0.0 0.0
1.0 1.0 1.0 1.0 1.0
```

In [8]:

```


# Expand image dimensions to width x height x channels
def reshape_img_data(x):
    return np.expand_dims(x, axis=3)


print(x_train_unlabeled.shape, x_train_labeled.shape, x_validate.shape, x_ho



# labels
def reshape_labels(y):
    return y.reshape(-1, 1)

y_train_unlabeled = reshape_labels(y_train_unlabeled)
y_train_labeled = reshape_labels(y_train_labeled)
y_validate = reshape_labels(y_validate)
y_holdout = reshape_labels(y_holdout)

print(y_train_unlabeled.shape, y_train_labeled.shape, y_validate.shape, y_ho
```

```
(26000, 784) (4000, 784) (10000, 784) (10240, 784) (5000, 784)
(26000, 1) (4000, 1) (10000, 1) (10240, 1)
```

In [9]:

```
# train labels 0..9, equal numbers of each
print(y_train_unlabeled.min(), y_train_unlabeled.max())
print(y_train_labeled.min(), y_train_labeled.max())

print(np.unique(y_train_labeled, return_counts=True))
print(np.unique(y_train_unlabeled, return_counts=True))


print(y_holdout.min(), y_holdout.max())
print(np.unique(y_holdout, return_counts=True))
```

```
0 9
0 9
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([409, 444, 396, 409, 365, 384, 38
9, 398, 401, 405]))
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([2555, 2591, 2521, 2612, 2675, 25
56, 2563, 2657, 2603, 2667]))
0 9
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([1024, 1024, 1024, 1024, 1024, 10
24, 1024, 1024, 1024, 1024]))
```

In [10]:

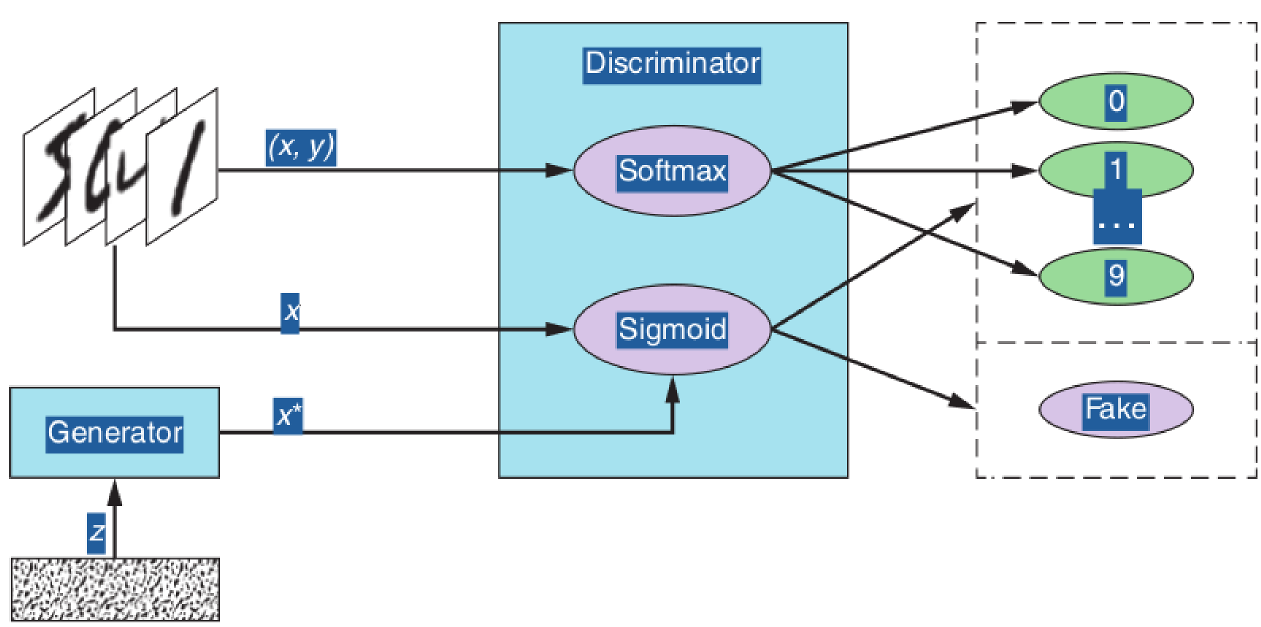
```
img_rows = 28
img_cols = 28
channels = 1

# 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
```

In [11]:

```python




# random batch of labeled train data
def batch_labeled(batch_size):

    x = x_train_labeled
    y = y_train_labeled

    # Get a random batch of labeled images and their labels
    idx = np.random.randint(0, num_labeled, batch_size)
    imgs = x[idx]
    labels = y[idx]

    imgs = imgs.reshape(batch_size, img_rows, img_cols, 1)
    labels = to_categorical(labels, num_classes=num_classes)

    return imgs, labels


# random batch unlabeled train data
def batch_unlabeled(batch_size):

    x = x_train_unlabeled
    y = y_train_unlabeled


    # Get a random batch of unlabeled images
    idx = np.random.randint(num_labeled, x.shape[0], batch_size)
    imgs = x[idx]

    imgs = imgs.reshape(batch_size, img_rows, img_cols, 1)


    return imgs


# full training set
def train_set():

    imgs = np.concatenate((x_train_labeled, x_train_unlabeled), axis=0)
    y = np.concatenate((y_train_labeled, y_train_unlabeled), axis=0)

    imgs = imgs.reshape(len(imgs), img_rows, img_cols, 1)
    labels = to_categorical(y, num_classes=num_classes)


    return imgs, labels




# validation set is a random selection of samples from train set
def test_set():

    imgs = x_validate
    y = y_validate

    imgs = imgs.reshape(len(imgs), img_rows, img_cols, 1)
    labels = to_categorical(y, num_classes=num_classes)

    return imgs, labels



# unseen set, somewhat different std and mean in images
def holdout_set():

    imgs = x_holdout
    y = y_holdout

    imgs = imgs.reshape(len(imgs), img_rows, img_cols, 1)
    labels = to_categorical(y, num_classes=num_classes)

    return imgs, labels


```

```
# unseen set, somewhat different std and mean in images
def submission_set():

    imgs = x_submission
    imgs = imgs.reshape(len(imgs), img_rows, img_cols, 1)

    return imgs




#######################################################################
# experimental
# use unlabeled holdout and validation data as unlabeled instead of part
#    of the training set

# random batch unlabeled train data
def batch_holdout(batch_size):

    x = x_holdout

    # Get a random batch of unlabeled images
    idx = np.random.randint(num_labeled, x.shape[0], batch_size)
    imgs = x[idx]

    imgs = imgs.reshape(batch_size, img_rows, img_cols, 1)


    return imgs



# random batch unlabeled train data
def batch_submission(batch_size):

    x = x_submission


    # Get a random batch of unlabeled images
    idx = np.random.randint(num_labeled, x.shape[0], batch_size)
    imgs = x[idx]

    imgs = imgs.reshape(batch_size, img_rows, img_cols, 1)


    return imgs

```

Semi-Supervied GAN



Generator

In [12]:

```
# model takes in a random noise vector of z_dim and learns to output a digit
# that is good enough that discriminator can't tell it from an MNIST databas



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')
    model.add(BatchNormalization())
    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'))
    model.add(BatchNormalization())
    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'))
    model.add(Activation('tanh'))


    return model
```

Discriminator

In [13]:

```
# takes input image and classifies it from 0...9


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'))
    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'))
    model.add(BatchNormalization())
    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'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.5))


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

    return model
```

In [14]:

```
# discriminator to label class
def build_discriminator_supervised(discriminator_net):

    model = Sequential()
    model.add(discriminator_net)
    model.add(Activation('softmax'))

    return model
```

In [15]:

```
# discriminator to id real/fake image
def build_discriminator_unsupervised(discriminator_net):

    model = Sequential()
    model.add(discriminator_net)

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

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

    return model
```

Build the Model

In [16]:

```python
def build_gan(generator, discriminator):

    model = Sequential()

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

    return model
```

Discriminator

In [17]:

```
# 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=['a
print('Supervised discriminator')
print_summary(discriminator_supervised)

# Build & compile the Discriminator for unsupervised training
discriminator_unsupervised = build_discriminator_unsupervised(discriminator_n
discriminator_unsupervised.compile(loss='binary_crossentropy', optimizer=Adan
print('Unsupervised discriminator')
print_summary(discriminator_unsupervised)
```

```
WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprecate
d. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated. Pl
ease use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:4138: The name tf.random_uniform is deprecate
d. Please use tf.random.uniform instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:174: The name tf.get_default_session is deprec
ated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:181: The name tf.ConfigProto is deprecated. Pl
ease use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is depre
cated. Please use tf.compat.v1.nn.fused_batch_norm instead.

WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.
ops.nn_ops) with keep_prob is deprecated and will be removed in a future versio
n.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - kee
p_prob`.
WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ker
as/optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.
compat.v1.train.Optimizer instead.

Supervised discriminator
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
sequential_1 (Sequential)    (None, 10)                113930    
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0         
=================================================================
Total params: 113,930
Trainable params: 113,546
Non-trainable params: 384
_________________________________________________________________
WARNING:tensorflow:From /home/herself/anaconda3/lib/python3.7/site-packages/ten
sorflow/python/ops/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Unsupervised discriminator
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
sequential_1 (Sequential)    (None, 10)                113930    
```

```
lambda_1 (Lambda)            (None, 1)                 0
=================================================================
Total params: 113,930
Trainable params: 113,546
Non-trainable params: 384
_________________________________________________________________
```

Generator

In [18]:

```
# Build the Generator
generator = build_generator(z_dim)
print('Generator')
print_summary(generator)

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

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

```
Generator
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 12544)             1266944
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 128)       295040
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_4 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 28, 28, 1)         577
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 1,637,121
Trainable params: 1,636,737
Non-trainable params: 384
_________________________________________________________________
GAN
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_4 (Sequential)    (None, 28, 28, 1)         1637121
_________________________________________________________________
sequential_3 (Sequential)    (None, 1)                 113930
=================================================================
Total params: 1,751,051
Trainable params: 1,636,737
Non-trainable params: 114,314
_________________________________________________________________
```

Training

In [19]:

```
supervised_losses = []
iteration_checkpoints = []



def train(iterations, batch_size, sample_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 = batch_labeled(batch_size)

############################################################################
        # Get unlabeled examples
        #imgs_unlabeled = batch_unlabeled(batch_size)
        #imgs_unlabeled = batch_holdout(batch_size)
        imgs_unlabeled = batch_submission(batch_size)

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

        # Train on real labeled examples
        d_loss_supervised, accuracy = discriminator_supervised.train_on_batch

        # Train on real unlabeled examples
        d_loss_real = discriminator_unsupervised.train_on_batch( imgs_unlabe

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

        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)

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

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

            # Save Discriminator supervised classification loss to be plotte
            supervised_losses.append(d_loss_supervised)
            iteration_checkpoints.append(iteration + 1)

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

Train the Model and Inspect Output

Note that the `'Discrepancy between trainable weights and collected trainable'` warning from Keras is expected. It is by design: The Generator's trainable parameters are intentionally held constant during Discriminator training, and vice versa.

In [20]:

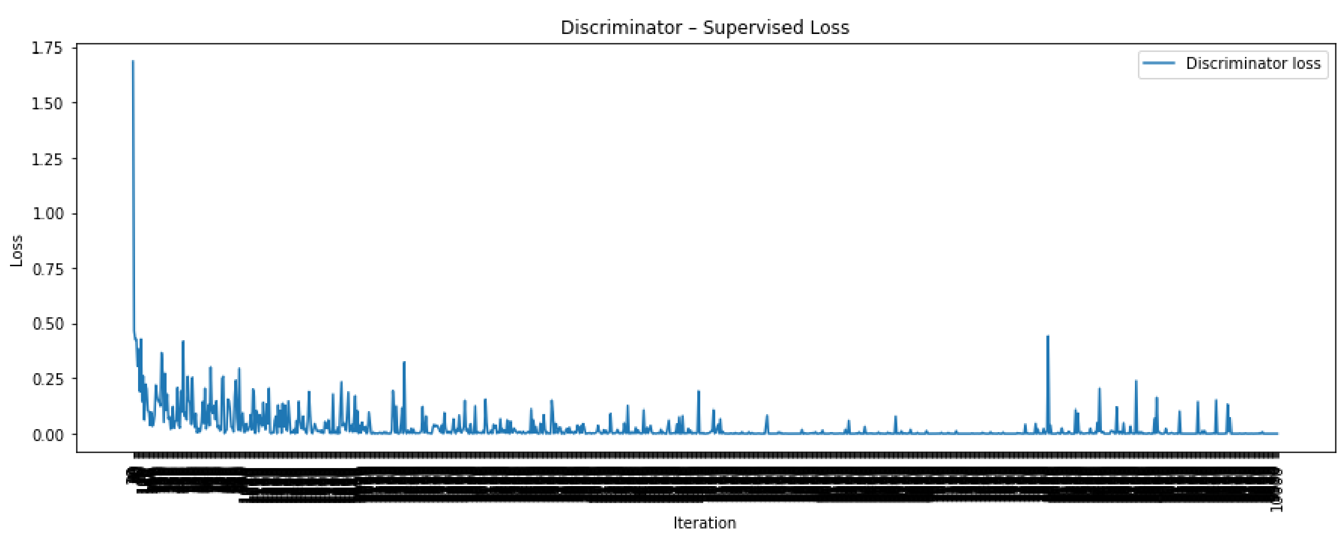
```python
# Set hyperparameters
n_epochs = 10000
batch_size = 32
sample_interval = 10

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

```
10 [D loss supervised: 1.6872, acc.: 46.88%] [D loss unsupervised: 0.2600] [G
loss: 1.899957]
20 [D loss supervised: 0.4659, acc.: 90.62%] [D loss unsupervised: 0.0328] [G
loss: 0.474590]
30 [D loss supervised: 0.4263, acc.: 90.62%] [D loss unsupervised: 0.0345] [G
loss: 0.861463]
40 [D loss supervised: 0.4284, acc.: 81.25%] [D loss unsupervised: 0.0162] [G
loss: 0.874039]
50 [D loss supervised: 0.3062, acc.: 93.75%] [D loss unsupervised: 0.0058] [G
loss: 0.199279]
60 [D loss supervised: 0.3830, acc.: 87.50%] [D loss unsupervised: 0.0086] [G
loss: 0.147347]
70 [D loss supervised: 0.1902, acc.: 96.88%] [D loss unsupervised: 0.0053] [G
loss: 0.253196]
80 [D loss supervised: 0.4286, acc.: 84.38%] [D loss unsupervised: 0.0074] [G
loss: 0.331915]
90 [D loss supervised: 0.1436, acc.: 93.75%] [D loss unsupervised: 0.0044] [G
loss: 0.023412]
100 [D loss supervised: 0.2604, acc.: 90.62%] [D loss unsupervised: 0.0028] [G
```

In [21]:

```python
losses = np.array(supervised_losses)

# Plot Discriminator supervised loss
plt.figure(figsize=(15, 5))
plt.plot(iteration_checkpoints, losses, label="Discriminator loss")

plt.xticks(iteration_checkpoints, rotation=90)

plt.title("Discriminator – Supervised Loss")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x7f1c58490c50>



SGAN Classifier – Training, Test, Holdout Accuracy

In [22]:

```python
# compute training data accuaracy
x, y = train_set()

_, accuracy = discriminator_supervised.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
30000/30000 [==============================] - 1s 37us/step
Test Accuracy: 98.47%
```

In [23]:
```
# compute validation data accuaracy
x, y = test_set()

_, accuracy = discriminator_supervised.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
10000/10000 [==============================] - 0s 36us/step
Test Accuracy: 98.41%
```

In [24]:
```
# compute holdout data accuracy
x, y = holdout_set()

_, accuracy = discriminator_supervised.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
10240/10240 [==============================] - 0s 40us/step
Test Accuracy: 73.44%
```

In [25]:
```
# create Kaggle submission data
# score was not good enough to bother submitting, code here only to show how
x = submission_set()

gan_predictions = discriminator_supervised.predict(x)
print('gan preds', gan_predictions.shape)
gan_preds = np.argmax(gan_predictions, axis = 1)

print(gan_preds)
```

```
gan preds (5000, 10)
[3 0 2 ... 1 6 3]
```

Fully-Supervised Classifier

- use same number of labeled samples
- see diff in number of training epoches needed

In [26]:
```
# Fully supervised classifier with the same network architecture as the SGAN
mnist_classifier = build_discriminator_supervised(build_discriminator_net(im
mnist_classifier.compile(loss='categorical_crossentropy', metrics=['accuracy
```

In [27]:

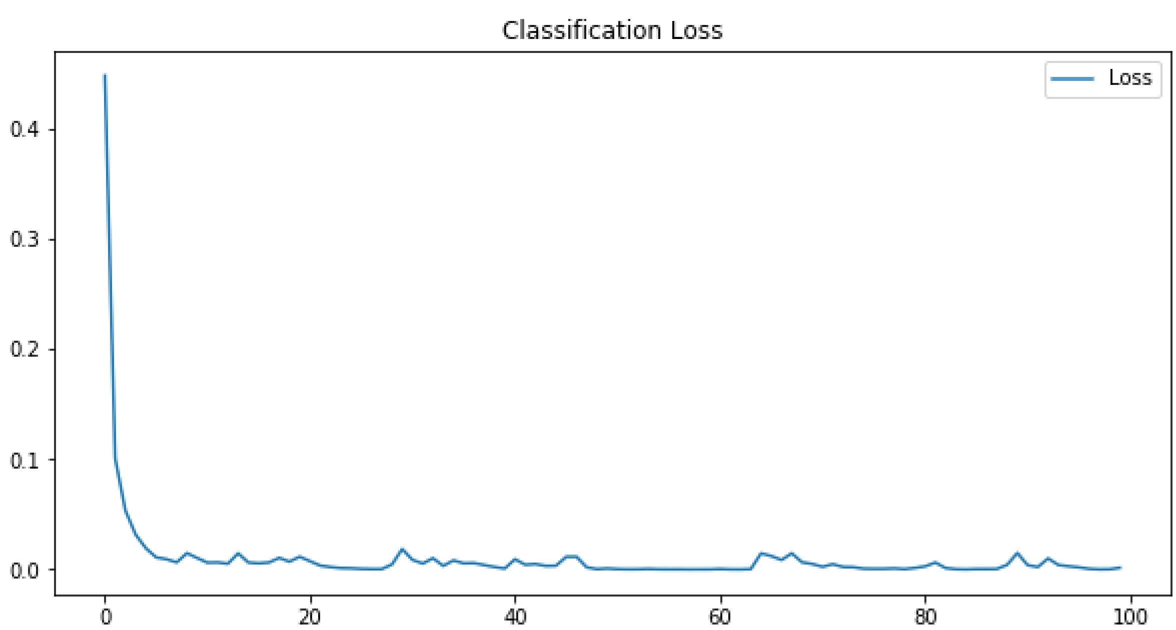
```python
imgs, labels = batch_labeled(num_labeled)


# Train the classifier
training = mnist_classifier.fit(x=imgs,
                                y=labels,
                                batch_size=batch_size,
                                epochs=n_epochs//100,
                                verbose=1)

losses = training.history['loss']
accuracies = training.history['acc']
```

```
Epoch 1/100
4000/4000 [==============================] - 1s 249us/step - loss: 0.4484 - ac
c: 0.8560
Epoch 2/100
4000/4000 [==============================] - 0s 107us/step - loss: 0.1011 - ac
c: 0.9667
Epoch 3/100
4000/4000 [==============================] - 0s 105us/step - loss: 0.0536 - ac
c: 0.9848
Epoch 4/100
4000/4000 [==============================] - 0s 109us/step - loss: 0.0317 - ac
c: 0.9933
Epoch 5/100
4000/4000 [==============================] - 0s 114us/step - loss: 0.0192 - ac
c: 0.9955
Epoch 6/100
4000/4000 [==============================] - 0s 105us/step - loss: 0.0110 - ac
c: 0.9978
Epoch 7/100
```

In [28]:

```python
# Plot classification loss
plt.figure(figsize=(10, 5))
plt.plot(np.array(losses), label="Loss")
plt.title("Classification Loss")
plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x7f1bf0d07128>



In [29]:

```python
# check training data accuracy

x, y = train_set()
_, accuracy = mnist_classifier.evaluate(x, y)
print("Training Accuracy: %.2f%%" % (100 * accuracy))
```

```
30000/30000 [==============================] - 1s 40us/step
Training Accuracy: 97.95%
```

In [30]:

```
# check validation data accuracy

x, y = test_set()
_, accuracy = mnist_classifier.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
10000/10000 [==============================] - 0s 38us/step
Test Accuracy: 97.96%
```

In [31]:

```
# check holdout data accuracy

x, y = holdout_set()
_, accuracy = mnist_classifier.evaluate(x, y)
print("Test Accuracy: %.2f%%" % (100 * accuracy))
```

```
10240/10240 [==============================] - 0s 40us/step
Test Accuracy: 71.66%
```

In [32]:

```
# create Kaggle submission data

x = submission_set()

clf_predictions = mnist_classifier.predict(x)
print('clf preds', clf_predictions.shape)
clf_preds = np.argmax(clf_predictions, axis = 1)
print(clf_preds)
```

```
clf preds (5000, 10)
[3 0 2 ... 1 6 3]
```