Generative Adversarial Networks (GAN) were introduced by Ian Goodfellow in 2014 and attacks the problem of unsupervised learning by training two deep networks, called a Generator and Discriminator. The networks compete and cooperate with each other and in the course of training, both networks eventually learn how to perform their respective tasks.

GAN is almost always explained like the case of a counterfeiter (Generative) and the police (Discriminator). Initially, the counterfeiter will show the police a fake money. The police says it is fake. The police gives feedback to the counterfeiter why the money is fake. The counterfeiter attempts to make a new fake money based on the feedback it received. The police says the money is still fake and offers a new set of feedback. The counterfeiter attempts to make a new fake money based on the latest feedback. The cycle continues indefinitely until the police is fooled by the fake money because it looks real.

A Deep Convolutional GAN (DCGAN) is a model that is able to learn by itself how to synthesize new images. Here we create a simple, proof of concept DCGAN using the MNIST dataset to learn how to write handwritten digits.

In [1]:
```
# imports
import numpy as np
from keras.datasets import mnist
from matplotlib import pyplot as plt
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Dropout, Flatten, BatchNormalization, Activation, Reshape, LeakyReLU
from keras.layers.convolutional import Conv2D, Conv2DTranspose, UpSampling2D
from keras.optimizers import RMSprop
```

```
Using TensorFlow backend.
```

In [2]:
```
# set random seed
seed = 7
np.random.seed(seed)
```

In [3]:
```
# define model parameters
depth = 16
depth2 = 64
dim = 7
dropout = 0.4
```

Discriminator

A discriminator that tells how real an image is. The sigmoid output is a scalar value of the probability of how real the image is - 0.0 is certainly fake, 1.0 is certainly real, anything in between is a gray area. The difference between a discriminator and a typical CNN is the absence of max-pooling in between layers. Instead, a strided convolution is used for downsampling.

In [4]:

```
# define the discriminator model
img_rows = 28
img_cols = 28
img_channels = 1
def police(depth, dropout):
    # in: 28 x 28 x 1
    # out: 14 x 14 x 1
    inputs = Input(shape=(img_rows,img_cols,img_channels))
    X = Conv2D(depth*1, 5, strides=2, padding='same')(inputs)
    X = LeakyReLU(alpha=0.2)(X)
    X = Dropout(dropout)(X)

    X = Conv2D(depth*2, 5, strides=2, padding='same')(X)
    X = LeakyReLU(alpha=0.2)(X)
    X = Dropout(dropout)(X)

    X = Conv2D(depth*4, 5, strides=2, padding='same')(X)
    X = LeakyReLU(alpha=0.2)(X)
    X = Dropout(dropout)(X)

    X = Conv2D(depth*8, 5, strides=1, padding='same')(X)
    X = LeakyReLU(alpha=0.2)(X)
    X = Dropout(dropout)(X)

    X = Flatten()(X)
    output = Dense(1, activation='sigmoid')(X)
    model = Model(inputs=inputs, outputs=output)
    return model
```

In [5]:
```
# create the discriminator model
discriminator = police(depth, dropout)
optimizer = RMSprop(lr=0.0002, decay=6e-8)
discriminator.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['acc'])
print("Discriminator Model")
print("------------------------")
discriminator.summary()
```

```
Discriminator Model
------------------------
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        416       
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 16)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 32)          12832     
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 7, 7, 32)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          51264     
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 4, 4, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         204928    
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 4, 128)         0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2049      
=================================================================
Total params: 271,489
Trainable params: 271,489
Non-trainable params: 0
_________________________________________________________________
```

Generator

The generator synthesizes fake images. The fake images are generated from a 100-dimensional noise (uniform distribution between -1.0 to 1.0) using the inverse of a convolution, called a transposed convolution. Upsampling layers are used instead of fractionally-strided convolutions because they synthesize more realistic handwriting images. In between layers, batch normalization is used to stabilize learning. The output of the sigmoid at the last layer produces the fake image.

In [6]:
```
# define the generator model
def thief(dim, depth2, dropout):
    inputs = Input(shape=(100,))
    # in: 100
    # out: dim x dim x depth*4
    X = Dense(dim*dim*depth2)(inputs)
    X = BatchNormalization(momentum=0.9)(X)
    X = Activation('relu')(X)
    X = Reshape((dim, dim, depth2))(X)
    X = Dropout(dropout)(X)

    # in: dim x dim x depth*4
    # out: dim*2 x dim*2 x depth*4 / 2
    X = UpSampling2D()(X)
    X = Conv2DTranspose(int(depth2/2), 5, padding='same')(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Activation('relu')(X)

    X = UpSampling2D()(X)
    X = Conv2DTranspose(int(depth2/4), 5, padding='same')(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Activation('relu')(X)

    X = Conv2DTranspose(int(depth2/8), 5, padding='same')(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Activation('relu')(X)

    # output is a 28 x 28 x 1 grayscale image at [0.0,1.0] per pixel
    X = Conv2DTranspose(1, 5, padding='same')(X)
    output = Activation('sigmoid')(X)
    model = Model(inputs=inputs, outputs=output)
    return model
```

In [7]:
```
# define the generator model
generator = thief(dim, depth2*4, dropout)
generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12544)             1266944
_________________________________________________________________
batch_normalization_1 (Batch (None, 12544)             50176
_________________________________________________________________
activation_1 (Activation)    (None, 12544)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 128)       819328
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
activation_2 (Activation)    (None, 14, 14, 128)       0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 64)        204864
_________________________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 28, 28, 32)        51232
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
activation_4 (Activation)    (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 1)         801
_________________________________________________________________
activation_5 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 2,394,241
Trainable params: 2,368,705
Non-trainable params: 25,536
_________________________________________________________________
```

Adversarial Model

The adversarial model is just the generator-discriminator stacked together. The Generator part is trying to fool the Discriminator and learning from its feedback at the same time.

In [8]:
```
# create the adversarial model
adv_model = Sequential()
adv_model.add(generator)
adv_model.add(discriminator)
AMoptimizer = RMSprop(lr=0.0001, decay=3e-8)
adv_model.compile(optimizer=AMoptimizer, loss='binary_crossentropy', metrics=['acc'])
print("Adversarial Model")
print("------------------------")
adv_model.summary()
```

```
Adversarial Model
------------------------
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_2 (Model)              (None, 28, 28, 1)         2394241
_________________________________________________________________
model_1 (Model)              (None, 1)                 271489
=================================================================
Total params: 2,665,730
Trainable params: 2,640,194
Non-trainable params: 25,536
_________________________________________________________________
```

In [9]:

```
# create the ground truth training set
from tensorflow.examples.tutorials.mnist import input_data
x_train = input_data.read_data_sets("mnist", one_hot=True).train.images
x_train = x_train.reshape(-1, img_rows, img_cols, img_channels).astype(np.float32)
```

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
```

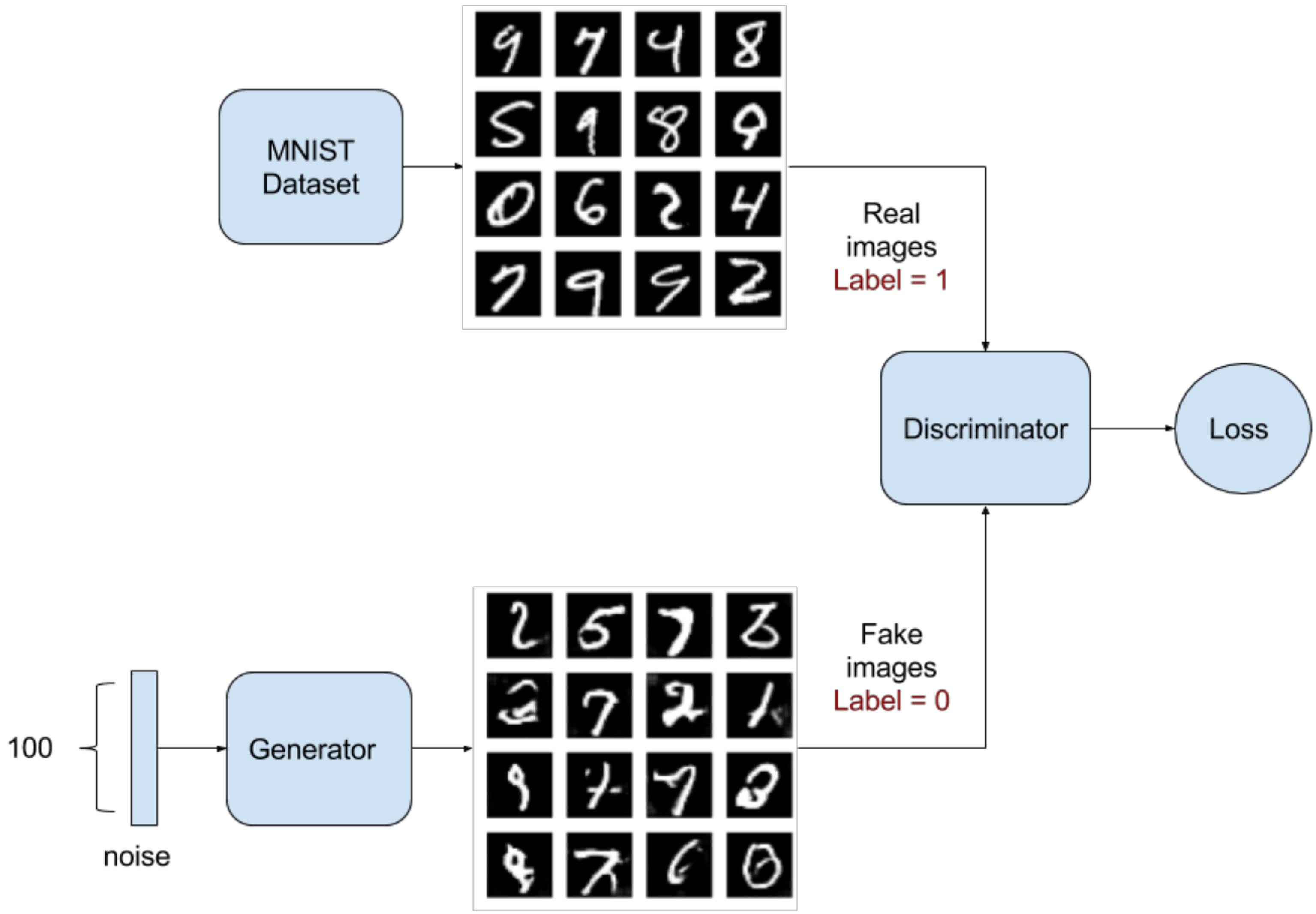
In [10]:
```
# validate mnist training input
print(x_train.shape)
print(x_train[0])
```

```
(55000, 28, 28, 1)
[[[0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]]

 [[0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
```

Training the Models

We first determine if Discriminator model is correct by training it alone with real and fake images. Afterwards, the Discriminator and Adversarial models are trained one after the other.



In [15]:
```
# create function to save and plot image outputs
def plot_images(save2file=False, fake=True, samples=16, noise=None, step=0):
    filename = 'mnist.png'
    if fake:
        if noise is None:
            noise = np.random.uniform(-1.0, 1.0, size=[samples, 100])
        else:
            filename = "mnist_%d.png" % step
        images = generator.predict(noise)
    else:
        i = np.random.randint(0, x_train.shape[0], samples)
        images = x_train[i, :, :, :]

    plt.figure(figsize=(10,10))
    for i in range(images.shape[0]):
        plt.subplot(4, 4, i+1)
        image = images[i, :, :, :]
        image = np.reshape(image, [img_rows, img_cols])
        plt.imshow(image, cmap='gray')
        plt.axis('off')
    plt.tight_layout()
    if save2file:
        plt.savefig(filename)
        plt.close('all')
    else:
        plt.show()
```

In [16]:
```
# define the training function
def train(train_steps, batch_size, save_interval=0):
    noise_input = None
    if save_interval > 0:
        noise_input = np.random.uniform(-1.0, 1.0, size=[16,100])
    for i in range(train_steps):
        # define the truth training image set
        images_train = x_train[np.random.randint(0, x_train.shape[0], size=batch_size), :, :, :]
        noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
        images_fake = generator.predict(noise)
        # create combined image group for training the discriminator
        x = np.concatenate((images_train, images_fake))
        y = np.ones([batch_size*2, 1])
        y[batch_size:, :] = 0
        # train the discriminator model on the image group
        d_loss = discriminator.train_on_batch(x, y)

        # train the adversarial model
        noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
        y = np.ones([batch_size, 1])
        a_loss = adv_model.train_on_batch(noise, y)

        # print outputs
        mesg = "%d: [Discriminator loss: %f, acc: %f]" % (i, d_loss[0], d_loss[1])
        mesg = ("%s: [Adversarial loss: %f, acc: %f]" % (mesg, a_loss[0], a_loss[1]))
        print(mesg)

        # save output samples to file
        if save_interval > 0:
            if (i + 1) % save_interval == 0:
                plot_images(save2file=True, samples=noise_input.shape[0], noise=noise_input, step=(i+1))
```

In [14]:
```
# execute training function and plot images
train(train_steps=5000, batch_size=256, save_interval=250)
```

```
0: [Discriminator loss: 0.694369, acc: 0.466797]: [Adversarial loss: 0.707928, acc: 0.097656]
1: [Discriminator loss: 0.678317, acc: 0.607422]: [Adversarial loss: 0.685899, acc: 0.726562]
2: [Discriminator loss: 0.661741, acc: 0.562500]: [Adversarial loss: 0.656999, acc: 0.980469]
3: [Discriminator loss: 0.639216, acc: 0.519531]: [Adversarial loss: 0.611857, acc: 1.000000]
4: [Discriminator loss: 0.608894, acc: 0.539062]: [Adversarial loss: 0.529676, acc: 1.000000]
5: [Discriminator loss: 0.569575, acc: 0.513672]: [Adversarial loss: 0.416652, acc: 1.000000]
6: [Discriminator loss: 0.531489, acc: 0.500000]: [Adversarial loss: 0.282076, acc: 1.000000]
7: [Discriminator loss: 0.493789, acc: 0.500000]: [Adversarial loss: 0.164512, acc: 1.000000]
8: [Discriminator loss: 0.459084, acc: 0.500000]: [Adversarial loss: 0.084239, acc: 1.000000]
9: [Discriminator loss: 0.434782, acc: 0.500000]: [Adversarial loss: 0.039957, acc: 1.000000]
10: [Discriminator loss: 0.410518, acc: 0.500000]: [Adversarial loss: 0.018235, acc: 1.000000]
11: [Discriminator loss: 0.391741, acc: 0.500000]: [Adversarial loss: 0.008866, acc: 1.000000]
12: [Discriminator loss: 0.371077, acc: 0.564453]: [Adversarial loss: 0.004310, acc: 1.000000]
13: [Discriminator loss: 0.359576, acc: 0.931641]: [Adversarial loss: 0.002239, acc: 1.000000]
14: [Discriminator loss: 0.349246, acc: 1.000000]: [Adversarial loss: 0.001031, acc: 1.000000]
15: [Discriminator loss: 0.342388, acc: 1.000000]: [Adversarial loss: 0.000599, acc: 1.000000]
16: [Discriminator loss: 0.336047, acc: 1.000000]: [Adversarial loss: 0.000286, acc: 1.000000]
17: [Discriminator loss: 0.331474, acc: 1.000000]: [Adversarial loss: 0.000142, acc: 1.000000]
18: [Discriminator loss: 0.325991, acc: 1.000000]: [Adversarial loss: 0.000093, acc: 1.000000]
19: [Discriminator loss: 0.321889, acc: 1.000000]: [Adversarial loss: 0.000063, acc: 1.000000]
20: [Discriminator loss: 0.317731, acc: 1.000000]: [Adversarial loss: 0.000039, acc: 1.000000]
21: [Discriminator loss: 0.313671, acc: 1.000000]: [Adversarial loss: 0.000033, acc: 1.000000]
22: [Discriminator loss: 0.308798, acc: 1.000000]: [Adversarial loss: 0.000021, acc: 1.000000]
23: [Discriminator loss: 0.303517, acc: 1.000000]: [Adversarial loss: 0.000017, acc: 1.000000]
24: [Discriminator loss: 0.297951, acc: 1.000000]: [Adversarial loss: 0.000011, acc: 1.000000]
25: [Discriminator loss: 0.290137, acc: 1.000000]: [Adversarial loss: 0.000011, acc: 1.000000]
26: [Discriminator loss: 0.283030, acc: 1.000000]: [Adversarial loss: 0.000009, acc: 1.000000]
27: [Discriminator loss: 0.274333, acc: 1.000000]: [Adversarial loss: 0.000011, acc: 1.000000]
28: [Discriminator loss: 0.263819, acc: 1.000000]: [Adversarial loss: 0.000010, acc: 1.000000]
29: [Discriminator loss: 0.252953, acc: 1.000000]: [Adversarial loss: 0.000015, acc: 1.000000]
30: [Discriminator loss: 0.240684, acc: 1.000000]: [Adversarial loss: 0.000010, acc: 1.000000]
31: [Discriminator loss: 0.226282, acc: 1.000000]: [Adversarial loss: 0.000013, acc: 1.000000]
32: [Discriminator loss: 0.211824, acc: 1.000000]: [Adversarial loss: 0.000028, acc: 1.000000]
33: [Discriminator loss: 0.193136, acc: 1.000000]: [Adversarial loss: 0.000036, acc: 1.000000]
34: [Discriminator loss: 0.175390, acc: 1.000000]: [Adversarial loss: 0.000034, acc: 1.000000]
35: [Discriminator loss: 0.157860, acc: 1.000000]: [Adversarial loss: 0.000029, acc: 1.000000]
36: [Discriminator loss: 0.137462, acc: 1.000000]: [Adversarial loss: 0.000256, acc: 1.000000]
37: [Discriminator loss: 0.121004, acc: 1.000000]: [Adversarial loss: 0.000305, acc: 1.000000]
38: [Discriminator loss: 0.107141, acc: 1.000000]: [Adversarial loss: 0.000056, acc: 1.000000]
39: [Discriminator loss: 0.089320, acc: 1.000000]: [Adversarial loss: 0.000128, acc: 1.000000]
40: [Discriminator loss: 0.076693, acc: 1.000000]: [Adversarial loss: 0.000631, acc: 1.000000]
41: [Discriminator loss: 0.067316, acc: 1.000000]: [Adversarial loss: 0.009857, acc: 0.996094]
42: [Discriminator loss: 0.068658, acc: 1.000000]: [Adversarial loss: 0.000683, acc: 1.000000]
43: [Discriminator loss: 0.055372, acc: 1.000000]: [Adversarial loss: 0.000166, acc: 1.000000]
44: [Discriminator loss: 0.047479, acc: 1.000000]: [Adversarial loss: 0.000023, acc: 1.000000]
45: [Discriminator loss: 0.047416, acc: 0.996094]: [Adversarial loss: 1.916961, acc: 0.523438]
```