Practical 6

Aim: Write a program for the Implementation of a Generative Adversarial Network for generating synthetic shapes (like digits)

Code:

```
from numpy import expand_dims
from numpy import ones
from numpy import zeros
from numpy.random import rand
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LeakyReLU
```

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
  model = Sequential()
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',
input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.4))
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.4))
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
  return model
```

```
# load and prepare mnist training images
def load_real_samples():
 # load mnist dataset
 (trainX, _), (_, _) = load_data()
 # expand to 3d, e.g. add channels dimension
 X = expand_dims(trainX, axis=-1)
```

```
 # convert from unsigned ints to floats
 X = X.astype('float32')
 # scale from [0,255] to [0,1]
 X = X / 255.0
 return X
```

```
# select real samples
def generate_real_samples(dataset, n_samples):
 # choose random instances
 ix = randint(0, dataset.shape[0], n_samples)
 # retrieve selected images
 X = dataset[ix]
 # generate 'real' class labels (1)
 y = ones((n_samples, 1))
 return X, y
```

```
# generate n fake samples with class labels
def generate_fake_samples(n_samples):
 # generate uniform random numbers in [0,1]
 X = rand(28 * 28 * n_samples)
 # reshape into a batch of grayscale images
 X = X.reshape((n_samples, 28, 28, 1))
 # generate 'fake' class labels (0)
 y = zeros((n_samples, 1))
 return X, y
```

```
# train the discriminator model
def train_discriminator(model, dataset, n_iter=100, n_batch=256):
 half_batch = int(n_batch / 2)
 # manually enumerate epochs
 for i in range(n_iter):
 # get randomly selected 'real' samples
  X_real, y_real = generate_real_samples(dataset, half_batch)
 # update discriminator on real samples
  _, real_acc = model.train_on_batch(X_real, y_real)
 # generate 'fake' examples
  X_fake, y_fake = generate_fake_samples(half_batch)
 # update discriminator on fake samples
  _, fake_acc = model.train_on_batch(X_fake, y_fake)
 # summarize performance
  print('>%d real=%.0f%% fake=%.0f%%' % (i+1, real_acc*100,
fake_acc*100))
```

```
# define the discriminator model
```

```
model = define_discriminator()
# load image data
dataset = load_real_samples()
# fit the model
train_discriminator(model, dataset)
```

Output:

```
>65 real=100% fake=100%
>66 real=100% fake=100%
>67 real=100% fake=100%
>68 real=100% fake=100%
>69 real=100% fake=100%
>70 real=100% fake=100%
>71 real=100% fake=100%
>72 real=100% fake=100%
>73 real=100% fake=100%
>74 real=100% fake=100%
>75 real=100% fake=100%
>76 real=100% fake=100%
>77 real=100% fake=100%
>78 real=100% fake=100%
>79 real=100% fake=100%
>80 real=100% fake=100%
>81 real=100% fake=100%
>82 real=100% fake=100%
>83 real=100% fake=100%
>84 real=100% fake=100%
>85 real=100% fake=100%
>86 real=100% fake=100%
>87 real=100% fake=100%
>88 real=100% fake=100%
>89 real=100% fake=100%
>90 real=100% fake=100%
>91 real=100% fake=100%
>92 real=100% fake=100%
```