In [1]:
```
#from tensorflow.examples.tutorials.mnist import input_data
#data = input_data.read_data_sets('data/fashion')
import tensorflow_datasets
datax, datay = tensorflow_datasets.load('fashion_mnist')
```

```
Downloading and preparing dataset fashion_mnist/3.0.1 (download: 29.45 MiB, gen
erated: 36.42 MiB, total: 65.87 MiB) to /root/tensorflow_datasets/fashion_mnis
t/3.0.1...

Dl Completed...: 0 url [00:00, ? url/s]

Dl Size...: 0 MiB [00:00, ? MiB/s]

Extraction completed...: 0 file [00:00, ? file/s]




0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/fashion_mnist/3.0.
1.incompleteMYM6WY/fashion_mnist-train.tfrecord

  0%|          | 0/60000 [00:00<?, ? examples/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/fashion_mnist/3.0.
1.incompleteMYM6WY/fashion_mnist-test.tfrecord

  0%|          | 0/10000 [00:00<?, ? examples/s]

Dataset fashion_mnist downloaded and prepared to /root/tensorflow_datasets/fash
ion_mnist/3.0.1. Subsequent calls will reuse this data.
```

In [2]:
```
datax[0]
```

Out[2]:
```
't'
```

In [3]:
```
datay[0]
```

Out[3]:
```
't'
```

In [4]:
```
import tensorflow
import tensorflow as tf
import matplotlib.pyplot as plt

import numpy as np

# example of defining the discriminator model
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
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
from keras.utils.vis_utils import plot_model

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.utils.vis_utils import plot_model
```

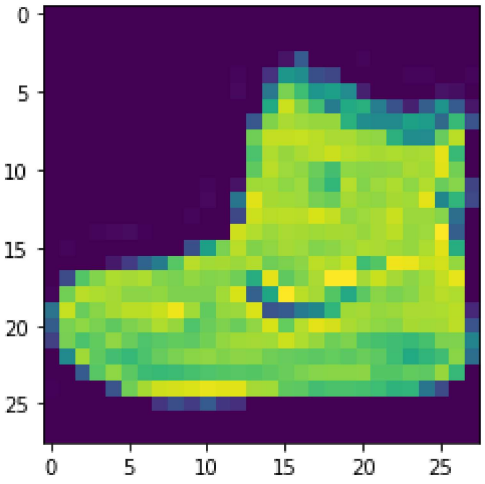
In [5]:
```
data = tensorflow.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz)
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz)
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz)
16384/5148 [===============================================================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz)
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

In [6]:
```
(trainx,trainy),(testx,testy) = data
```

In [7]:
```
plt.imshow(trainx[0])
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x7fc39f42f110>
```



In [8]:
```
trainy[0]
```

Out[8]:
```
9
```

In [9]:
```
category={0:'T-shirt/top',
1:'Trouser',
2:'Pullover',
3:'Dress',
4:'Coat',
5:'Sandal',
6:'Shirt',
7:'Sneaker',
8:'Bag',
9:'Ankle boot'}
```

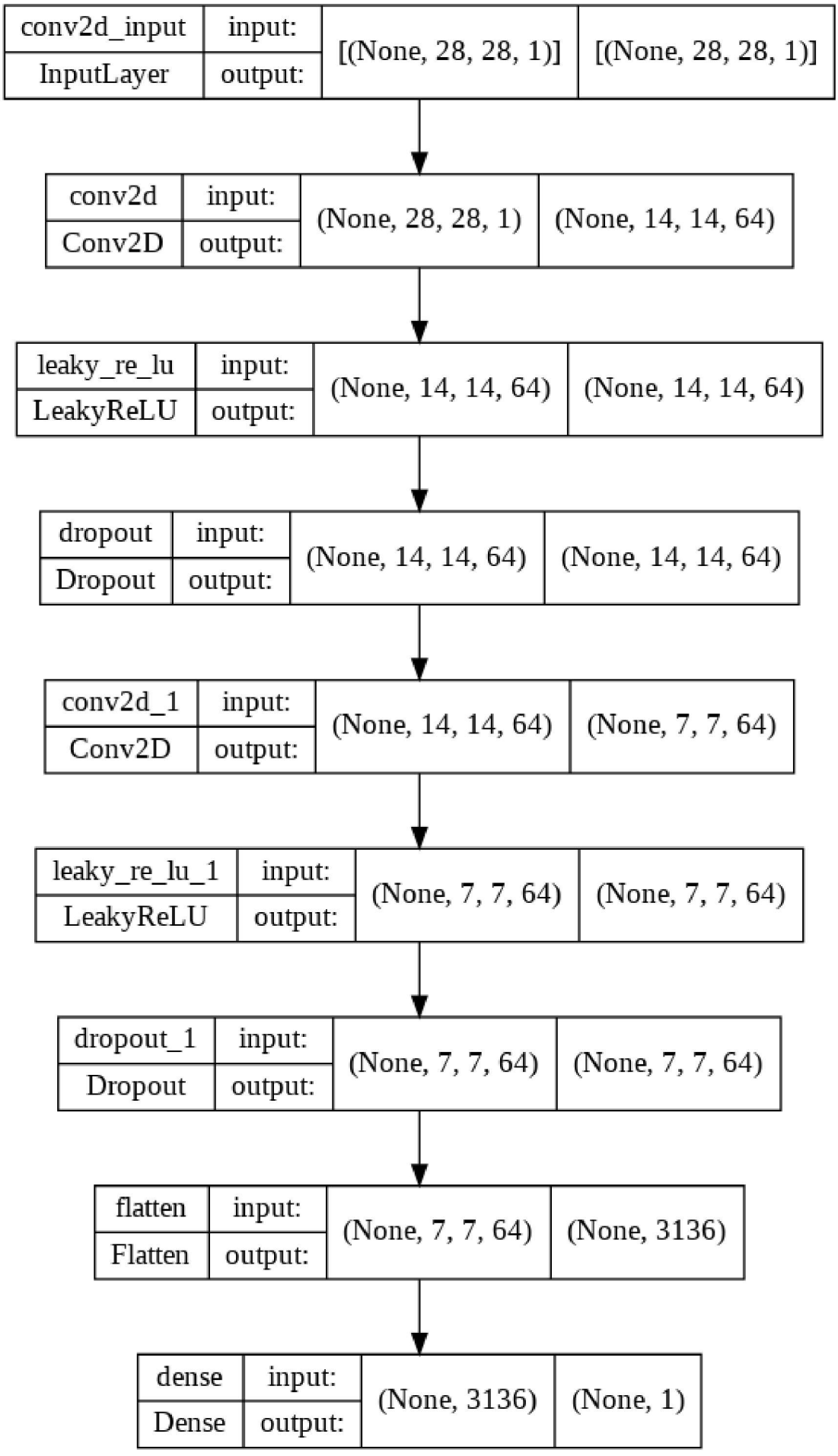
In [10]:
```
# plot images from the training dataset
for i in range(25):
    # define subplot
    plt.subplot(5, 5, 1 + i)
    # turn off axis
    plt.axis('off')
    # plot raw pixel data
    plt.imshow(trainx[i], cmap='gray_r')
plt.show()
```

In [11]:
```
# 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_sh
    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
```

In [12]:
```
# define model
model = define_discriminator()
# summarize the model
model.summary()
# plot the model
plot_model(model, to_file='discriminator_plot.png', show_shapes=True, show_layer_
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        640

 leaky_re_lu (LeakyReLU)     (None, 14, 14, 64)        0

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 64)          36928

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 64)          0

 dropout_1 (Dropout)         (None, 7, 7, 64)          0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 1)                 3137

=================================================================
Total params: 40,705
Trainable params: 40,705
Non-trainable params: 0
_________________________________________________________________

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarn
ing: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Out[12]:


conv2d_input
InputLayer
input:
output:
[(None, 28, 28, 1)]
[(None, 28, 28, 1)]
conv2d
Conv2D
input:
output:
(None, 28, 28, 1)
(None, 14, 14, 64)
leaky_re_lu
LeakyReLU
input:
output:
(None, 14, 14, 64)
(None, 14, 14, 64)
dropout
Dropout
input:
output:
(None, 14, 14, 64)
(None, 14, 14, 64)
conv2d_1
Conv2D
input:
output:
(None, 14, 14, 64)
(None, 7, 7, 64)
leaky_re_lu_1
LeakyReLU
input:
output:
(None, 7, 7, 64)
(None, 7, 7, 64)
dropout_1
Dropout
input:
output:
(None, 7, 7, 64)
(None, 7, 7, 64)
flatten
Flatten
input:
output:
(None, 7, 7, 64)
(None, 3136)
dense
Dense
input:
output:
(None, 3136)
(None, 1)

In [13]:
```
# load and prepare mnist training images
def load_real_samples():
    # load mnist dataset
    (trainX, _), (_, _) = tensorflow.keras.datasets.fashion_mnist.load_data()
    # expand to 3d, e.g. add channels dimension
    X = np.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
```

In [14]:
```
# select real samples
def generate_real_samples(dataset, n_samples):
    # choose random instances
    ix = np.random.randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    X = dataset[ix]
    # generate 'real' class labels (1)
    y = np.ones((n_samples, 1))
    return X, y
```

In [15]:
```
# generate n fake samples with class labels
def generate_fake_samples(n_samples):
    # generate uniform random numbers in [0,1]
    X = np.random.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 = np.zeros((n_samples, 1))
    return X, y
```

In [16]:
```
# 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))
```

In [17]:
```
# define the discriminator model
model = define_discriminator()
# load image data
dataset = load_real_samples()
# fit the model
train_discriminator(model, dataset)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarn
ing: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)

>1 real=55% fake=18%
>2 real=66% fake=27%
>3 real=67% fake=41%
>4 real=65% fake=51%
>5 real=54% fake=59%
>6 real=51% fake=72%
>7 real=61% fake=77%
>8 real=61% fake=84%
>9 real=62% fake=91%
>10 real=59% fake=93%
>11 real=48% fake=95%
>12 real=54% fake=95%
>13 real=56% fake=97%
>14 real=56% fake=97%
>15 real=55% fake=99%
>16 real=52% fake=99%
>17 real=55% fake=100%
>18 real=52% fake=100%
>19 real=63% fake=100%
>20 real=65% fake=100%
>21 real=61% fake=100%
>22 real=68% fake=100%
>23 real=61% fake=100%
>24 real=66% fake=100%
>25 real=63% fake=100%
>26 real=76% fake=100%
>27 real=79% fake=100%
>28 real=80% fake=100%
>29 real=66% fake=100%
>30 real=78% fake=100%
>31 real=76% fake=100%
>32 real=84% fake=100%
>33 real=74% fake=100%
>34 real=86% fake=100%
>35 real=87% fake=100%
>36 real=92% fake=100%
>37 real=98% fake=100%
>38 real=90% fake=100%
>39 real=88% fake=100%
>40 real=93% fake=100%
>41 real=95% fake=100%
>42 real=92% fake=100%
>43 real=94% fake=100%
>44 real=93% fake=100%
>45 real=98% fake=100%
```

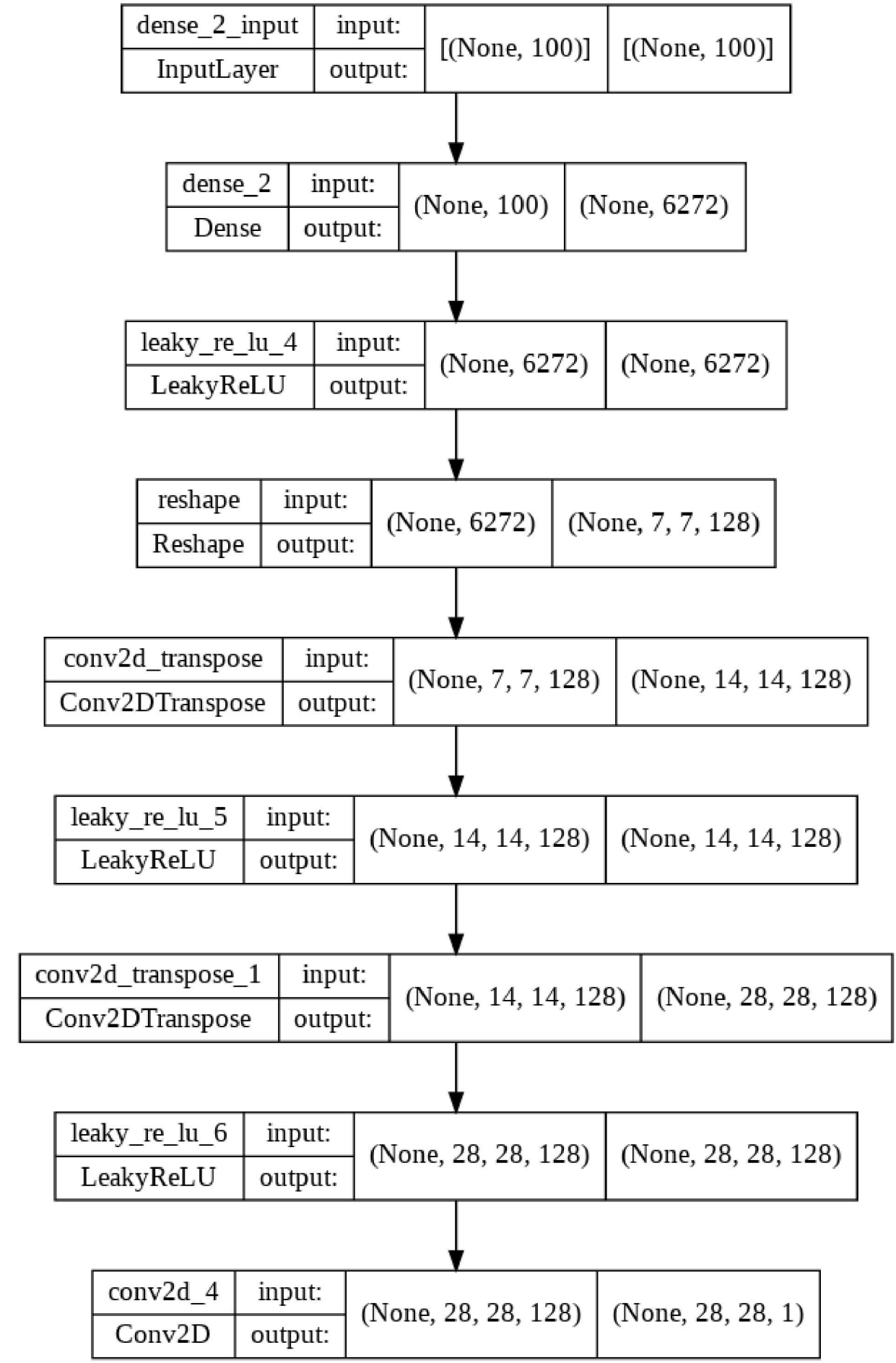
```
>46 real=93% fake=100%
>47 real=95% fake=100%
>48 real=93% fake=100%
>49 real=95% fake=100%
>50 real=95% fake=100%
>51 real=95% fake=100%
>52 real=95% fake=100%
>53 real=95% fake=100%
>54 real=95% fake=100%
>55 real=97% fake=100%
>56 real=97% fake=100%
>57 real=95% fake=100%
>58 real=98% fake=100%
>59 real=97% fake=100%
>60 real=98% fake=100%
>61 real=98% fake=100%
>62 real=98% fake=100%
>63 real=99% fake=100%
>64 real=99% fake=100%
>65 real=98% fake=100%
>66 real=96% fake=100%
>67 real=99% fake=100%
>68 real=98% fake=100%
>69 real=96% fake=100%
>70 real=98% fake=100%
>71 real=99% fake=100%
>72 real=99% fake=100%
>73 real=98% fake=100%
>74 real=98% fake=100%
>75 real=96% fake=100%
>76 real=99% fake=100%
>77 real=99% fake=100%
>78 real=97% fake=100%
>79 real=99% fake=100%
>80 real=99% fake=100%
>81 real=98% fake=100%
>82 real=98% fake=100%
>83 real=98% fake=100%
>84 real=98% fake=100%
>85 real=98% fake=100%
>86 real=98% fake=100%
>87 real=97% fake=100%
>88 real=99% fake=100%
>89 real=100% fake=100%
>90 real=100% fake=100%
>91 real=99% fake=100%
>92 real=100% fake=100%
>93 real=98% fake=100%
>94 real=99% fake=100%
>95 real=100% fake=100%
>96 real=100% fake=100%
>97 real=100% fake=100%
>98 real=98% fake=100%
>99 real=99% fake=100%
>100 real=98% fake=100%
```

In [18]:
```
# define the standalone generator model
def define_generator(latent_dim):
    model = Sequential()
    # foundation for 7x7 image
    n_nodes = 128 * 7 * 7
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    # upsample to 14x14
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 28x28
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(1, (7,7), activation='sigmoid', padding='same'))
    return model
```

In [19]:
```
# define the size of the latent space
latent_dim = 100
# define the generator model
model = define_generator(latent_dim)
# summarize the model
model.summary()
# plot the model
plot_model(model, to_file='generator_plot.png', show_shapes=True, show_layer_name
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 6272)              633472

 leaky_re_lu_4 (LeakyReLU)   (None, 6272)              0

 reshape (Reshape)           (None, 7, 7, 128)         0

 conv2d_transpose (Conv2DTra  (None, 14, 14, 128)      262272
 nspose)

 leaky_re_lu_5 (LeakyReLU)   (None, 14, 14, 128)       0

 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 128)      262272
 ranspose)

 leaky_re_lu_6 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_4 (Conv2D)           (None, 28, 28, 1)         6273

=================================================================
Total params: 1,164,289
Trainable params: 1,164,289
Non-trainable params: 0
_________________________________________________________________
```

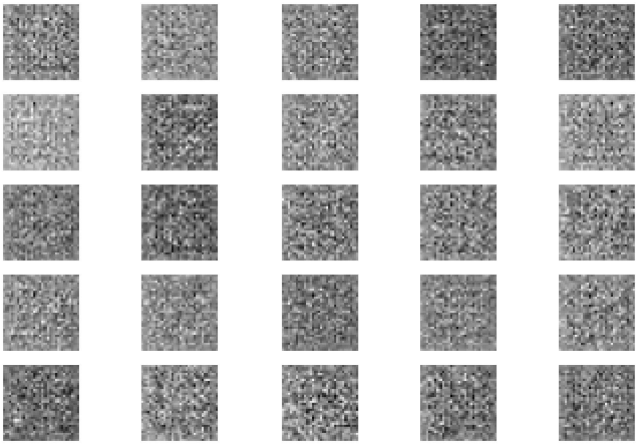
Out[19]:


dense_2_input
InputLayer
input:
output:
[(None, 100)]
[(None, 100)]
dense_2
Dense
input:
output:
(None, 100)
(None, 6272)
leaky_re_lu_4
LeakyReLU
input:
output:
(None, 6272)
(None, 6272)
reshape
Reshape
input:
output:
(None, 6272)
(None, 7, 7, 128)
conv2d_transpose
Conv2DTranspose
input:
output:
(None, 7, 7, 128)
(None, 14, 14, 128)
leaky_re_lu_5
LeakyReLU
input:
output:
(None, 14, 14, 128)
(None, 14, 14, 128)
conv2d_transpose_1
Conv2DTranspose
input:
output:
(None, 14, 14, 128)
(None, 28, 28, 128)
leaky_re_lu_6
LeakyReLU
input:
output:
(None, 28, 28, 128)
(None, 28, 28, 128)
conv2d_4
Conv2D
input:
output:
(None, 28, 28, 128)
(None, 28, 28, 1)

In [20]:
```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = np.random.randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input
```

In [21]:
```
# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = np.zeros((n_samples, 1))
    return X, y
```

In [22]:
```
# size of the latent space
latent_dim = 100
# define the discriminator model
model = define_generator(latent_dim)
# generate samples
n_samples = 25
X, _ = generate_fake_samples(model, latent_dim, n_samples)
# plot the generated samples
for i in range(n_samples):
    # define subplot
    plt.subplot(5, 5, 1 + i)
    # turn off axis labels
    plt.axis('off')
    # plot single image
    plt.imshow(X[i, :, :, 0], cmap='gray_r')
# show the figure
plt.show()
```

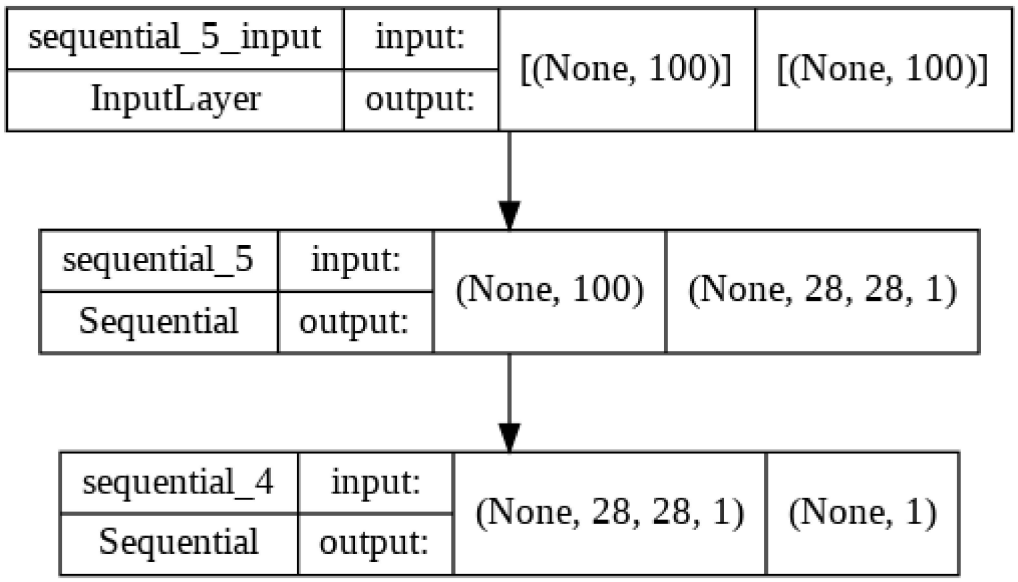


In [23]:
```
# define the combined generator and discriminator model, for updating the generat
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model
```

In [24]:
```
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# summarize gan model
gan_model.summary()
# plot gan model
plot_model(gan_model, to_file='gan_plot.png', show_shapes=True, show_layer_names=
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_5 (Sequential)   (None, 28, 28, 1)         1164289

 sequential_4 (Sequential)   (None, 1)                 40705

=================================================================
Total params: 1,204,994
Trainable params: 1,164,289
Non-trainable params: 40,705
_________________________________________________________________

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarn
ing: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Out[24]:



In [25]:
```
# train the composite model
def train_gan(gan_model, latent_dim, n_epochs=100, n_batch=256):
    # manually enumerate epochs
    for i in range(n_epochs):
        # prepare points in latent space as input for the generator
        x_gan = generate_latent_points(latent_dim, n_batch)
        # create inverted labels for the fake samples
        y_gan = np.ones((n_batch, 1))
        # update the generator via the discriminator's error
        gan_model.train_on_batch(x_gan, y_gan)
```

In [26]:
```
# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = generate_real_samples(dataset, half_batch)
            # generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batc
            # create training set for the discriminator
            X, y = np.vstack((X_real, X_fake)), np.vstack((y_real, y_fake))
            # update discriminator model weights
            d_loss, _ = d_model.train_on_batch(X, y)
            # prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # create inverted labels for the fake samples
            y_gan = np.ones((n_batch, 1))
            # update the generator via the discriminator's error
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            # summarize loss on this batch
            print('>%d, %d/%d, d=%.3f, g=%.3f' % (i+1, j+1, bat_per_epo, d_loss,
```

In [27]:
```
# evaluate the discriminator, plot generated images, save generator model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples
    # prepare real samples
    X_real, y_real = generate_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100, acc_fake*100))
```

In [28]:
```
# create and save a plot of generated images (reversed grayscale)
def save_plot(examples, epoch, n=10):
    # plot images
    for i in range(n * n):
        # define subplot
        plt.subplot(n, n, 1 + i)
        # turn off axis
        plt.axis('off')
        # plot raw pixel data
        plt.imshow(examples[i, :, :, 0], cmap='gray_r')
    # save plot to file
    filename = 'generated_plot_e%03d.png' % (epoch+1)
    plt.savefig(filename)
    plt.close()
```

In [29]:
```
# evaluate the discriminator, plot generated images, save generator model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples
    # prepare real samples
    X_real, y_real = generate_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100, acc_fake*100))
    # save plot
    save_plot(x_fake, epoch)
    # save the generator model tile file
    filename = 'generator_model_%03d.h5' % (epoch + 1)
    g_model.save(filename)
```

In [30]:
```
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [30]:
```
# generate images
latent_points = generate_latent_points(100, 25)
# generate images
X = model.predict(latent_points)
# plot the result
save_plot(X, 5)
```

In [33]:
```
!ls -l
```

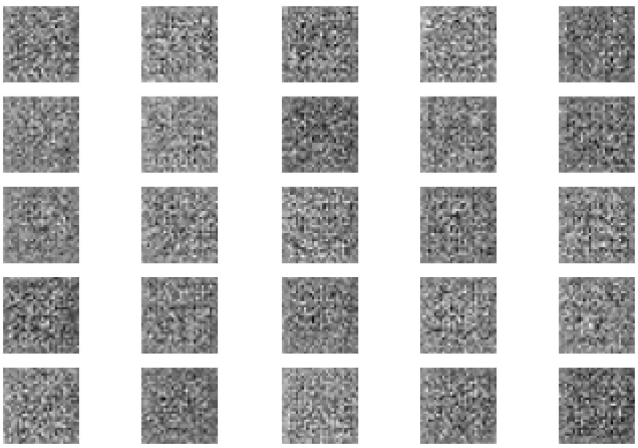
```
total 160
-rw-r--r-- 1 root root 62053 Feb 27 02:20 discriminator_plot.png
-rw-r--r-- 1 root root 20339 Feb 27 02:20 gan_plot.png
-rw-r--r-- 1 root root 72272 Feb 27 02:20 generator_plot.png
drwxr-xr-x 1 root root  4096 Feb 18 14:33 sample_data
```

In [32]:
```
!ls -l sample_data/
```

```
total 55504
-rwxr-xr-x 1 root root     1697 Jan  1  2000 anscombe.json
-rw-r--r-- 1 root root   301141 Feb 18 14:33 california_housing_test.csv
-rw-r--r-- 1 root root  1706430 Feb 18 14:33 california_housing_train.csv
-rw-r--r-- 1 root root 18289443 Feb 18 14:33 mnist_test.csv
-rw-r--r-- 1 root root 36523880 Feb 18 14:33 mnist_train_small.csv
-rwxr-xr-x 1 root root      930 Jan  1  2000 README.md
```

In [38]:
```
def save_plot(examples, n):
    # plot images
    for i in range(n * n):
        # define subplot
        plt.subplot(n, n, 1 + i)
        # turn off axis
        plt.axis('off')
        # plot raw pixel data
        plt.imshow(examples[i, :, :, 0], cmap='gray_r')
        plt.savefig('./plot.png')
    plt.show()
```

In [39]:
```
# generate images
latent_points = generate_latent_points(100, 25)
# generate images
X = model.predict(latent_points)
# plot the result
save_plot(X, 5)
```



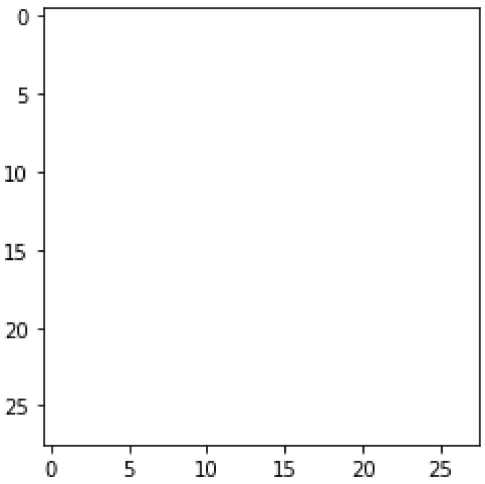
In [40]:
```
ls -l
```

```
total 244
-rw-r--r-- 1 root root 62053 Feb 27 02:20 discriminator_plot.png
-rw-r--r-- 1 root root 20339 Feb 27 02:20 gan_plot.png
-rw-r--r-- 1 root root 72272 Feb 27 02:20 generator_plot.png
-rw-r--r-- 1 root root 83506 Feb 27 04:11 plot.png
drwxr-xr-x 1 root root  4096 Feb 18 14:33 sample_data/
```

In [43]:
```
!plot.png
```

```
/bin/bash: plot.png: command not found
```

In [45]:
```
# all 0s
vector = np.asarray([[0.0 for _ in range(100)]])
# generate image
X = model.predict(vector)
# plot the result
plt.imshow(X[0, :, :, 0], cmap='gray_r')
plt.show()
```



In [46]:
```
model.save('model.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

In [48]:
```
!ls -lh
```

```
total 4.8M
-rw-r--r-- 1 root root  61K Feb 27 02:20 discriminator_plot.png
-rw-r--r-- 1 root root  20K Feb 27 02:20 gan_plot.png
-rw-r--r-- 1 root root  71K Feb 27 02:20 generator_plot.png
-rw-r--r-- 1 root root 4.5M Feb 27 04:13 model.h5
-rw-r--r-- 1 root root  82K Feb 27 04:11 plot.png
drwxr-xr-x 1 root root 4.0K Feb 18 14:33 sample_data
```

In [ ]:
```
```