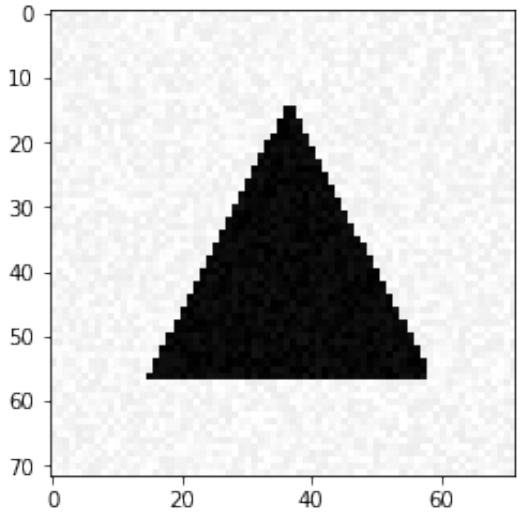
In [181]:
```
!pip install -q keras
import keras
```

In [182]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import keras.utils

def generate_a_drawing(figsize, U, V, noise=0.0):
    fig = plt.figure(figsize=(figsize,figsize))
    ax = plt.subplot(111)
    plt.axis('Off')
    ax.set_xlim(0,figsize)
    ax.set_ylim(0,figsize)
    ax.fill(U, V, "k")
    fig.canvas.draw()
    imdata = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)[
    imdata = imdata + noise * np.random.random(imdata.size)
    plt.close(fig)
    return imdata

def generate_a_rectangle(noise=0.0, free_location=False):
    figsize = 1.0
    U = np.zeros(4)
    V = np.zeros(4)
    if free_location:
        corners = np.random.random(4)
        top = max(corners[0], corners[1])
        bottom = min(corners[0], corners[1])
        left = min(corners[2], corners[3])
        right = max(corners[2], corners[3])
    else:
        side = (0.3 + 0.7 * np.random.random()) * figsize
        top = figsize/2 + side/2
        bottom = figsize/2 - side/2
        left = bottom
        right = top
    U[0] = U[1] = top
    U[2] = U[3] = bottom
    V[0] = V[3] = left
    V[1] = V[2] = right
    return generate_a_drawing(figsize, U, V, noise)


def generate_a_disk(noise=0.0, free_location=False):
    figsize = 1.0
    if free_location:
        center = np.random.random(2)
    else:
        center = (figsize/2, figsize/2)
    radius = (0.3 + 0.7 * np.random.random()) * figsize/2
    N = 50
```

```
    N = 50
    U = np.zeros(N)
    V = np.zeros(N)
    i = 0
    for t in np.linspace(0, 2*np.pi, N):
        U[i] = center[0] + np.cos(t) * radius
        V[i] = center[1] + np.sin(t) * radius
        i = i + 1
    return generate_a_drawing(figsize, U, V, noise)

def generate_a_triangle(noise=0.0, free_location=False):
    figsize = 1.0
    if free_location:
        U = np.random.random(3)
        V = np.random.random(3)
    else:
        size = (0.3 + 0.7 * np.random.random())*figsize/2
        middle = figsize/2
        U = (middle, middle+size, middle-size)
        V = (middle+size, middle-size, middle-size)
    imdata = generate_a_drawing(figsize, U, V, noise)
    return [imdata, [U[0], V[0], U[1], V[1], U[2], V[2]]]


im = generate_a_rectangle(10, True)
plt.imshow(im.reshape(72,72), cmap='gray')

im = generate_a_disk(10)
plt.imshow(im.reshape(72,72), cmap='gray')

[im, v] = generate_a_triangle(20, False)
plt.imshow(im.reshape(72,72), cmap='gray')


def generate_dataset_classification(nb_samples, noise=0.0, free_locati
    # Getting im_size:
    im_size = generate_a_rectangle().shape[0]
    X = np.zeros([nb_samples,im_size])
    Y = np.zeros(nb_samples)
    print('Creating data:')
    for i in range(nb_samples):
        if i % 10 == 0:
            print(i)
        category = np.random.randint(3)
        if category == 0:
            X[i] = generate_a_rectangle(noise, free_location)
        elif category == 1:
            X[i] = generate_a_disk(noise, free_location)
        else:
            [X[i], V] = generate_a_triangle(noise, free_location)
        Y[i] = category
    X = (X + noise) / (255 + 2 * noise)
    return [X, Y]

def generate_test_set_classification():
```

```
def generate_test_set_classification():
    np.random.seed(42)
    [X_test, Y_test] = generate_dataset_classification(300, 20, True)
    Y_test = np_utils.to_categorical(Y_test, 3)
    return [X_test, Y_test]

def generate_dataset_regression(nb_samples, noise=0.0):
    # Getting im_size:
    im_size = generate_a_triangle()[0].shape[0]
    X = np.zeros([nb_samples,im_size])
    Y = np.zeros([nb_samples, 6])
    print('Creating data:')
    for i in range(nb_samples):
        if i % 10 == 0:
            print(i)
        [X[i], Y[i]] = generate_a_triangle(noise, True)
    X = (X + noise) / (255 + 2 * noise)
    return [X, Y]

import matplotlib.patches as patches

def visualize_prediction(x, y):
    fig, ax = plt.subplots(figsize=(5, 5))
    I = x.reshape((72,72))
    ax.imshow(I, extent=[-0.15,1.15,-0.15,1.15],cmap='gray')
    ax.set_xlim([0,1])
    ax.set_ylim([0,1])

    xy = y.reshape(3,2)
    tri = patches.Polygon(xy, closed=True, fill = False, edgecolor = '
    ax.add_patch(tri)

    plt.show()

def generate_test_set_regression():
    np.random.seed(42)
    [X_test, Y_test] = generate_dataset_regression(300, 20)
    Y_test = np_utils.to_categorical(Y_test, 3)
    return [X_test, Y_test]
```



# Simple Classification

In [143]:
```
# Generate the training dataset, and transfer y_train into a category
from keras import utils as np_utils
[X_train, Y_train] = generate_dataset_classification(300, 20)
Y_train = np_utils.to_categorical(Y_train)
```

```
Creating data:
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
```

In [ ]:
```

```

In [144]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation,Flatten,BatchNormalization
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers import Input,UpSampling2D,merge,Conv2D,Cropping2D,co
from keras.models import Model
from keras import layers
```

In [146]:
```
model = Sequential()
```

```
n_cols = X_train[0].shape

#one single layer, and use softmax
model.add(Dense(units=3,activation='softmax',input_shape=n_cols))

# use the Adam optimizor
model.compile(loss='categorical_crossentropy',optimizer = 'adam',metri

# fit the model
model.fit(X_train, Y_train, epochs=32, batch_size=40)
```

```
Epoch 1/32

300/300 [==============================] - 1s 2ms/step - loss: 2.197
5 - acc: 0.3433
Epoch 2/32
300/300 [==============================] - 0s 103us/step - loss: 0.8
739 - acc: 0.6000
Epoch 3/32
300/300 [==============================] - 0s 89us/step - loss: 0.56
74 - acc: 0.7933
Epoch 4/32
300/300 [==============================] - 0s 79us/step - loss: 0.44
64 - acc: 0.8333
Epoch 5/32
300/300 [==============================] - 0s 93us/step - loss: 0.35
06 - acc: 0.9433
Epoch 6/32
300/300 [==============================] - 0s 91us/step - loss: 0.31
18 - acc: 0.8800
Epoch 7/32
300/300 [==============================] - 0s 83us/step - loss: 0.26
13 - acc: 0.9633
Epoch 8/32
300/300 [==============================] - 0s 92us/step - loss: 0.21
49 - acc: 0.9733
Epoch 9/32
300/300 [==============================] - 0s 93us/step - loss: 0.20
68 - acc: 0.9733
Epoch 10/32
300/300 [==============================] - 0s 87us/step - loss: 0.17
42 - acc: 1.0000
Epoch 11/32
300/300 [==============================] - 0s 99us/step - loss: 0.15
96 - acc: 0.9833
Epoch 12/32
300/300 [==============================] - 0s 88us/step - loss: 0.15
23 - acc: 0.9800
Epoch 13/32
300/300 [==============================] - 0s 103us/step - loss: 0.1
306 - acc: 0.9967
Epoch 14/32
300/300 [==============================] - 0s 98us/step - loss: 0.12
57 - acc: 0.9833
Epoch 15/32
```

```
300/300 [==============================] - 0s 93us/step - loss: 0.12
14 - acc: 1.0000
Epoch 16/32
300/300 [==============================] - 0s 83us/step - loss: 0.11
46 - acc: 0.9933
Epoch 17/32
300/300 [==============================] - 0s 91us/step - loss: 0.09
87 - acc: 1.0000
Epoch 18/32
300/300 [==============================] - 0s 96us/step - loss: 0.09
68 - acc: 0.9900

Epoch 19/32
300/300 [==============================] - 0s 93us/step - loss: 0.11
78 - acc: 0.9800
Epoch 20/32
300/300 [==============================] - 0s 84us/step - loss: 0.09
48 - acc: 0.9833
Epoch 21/32
300/300 [==============================] - 0s 95us/step - loss: 0.08
22 - acc: 0.9967
Epoch 22/32
300/300 [==============================] - 0s 95us/step - loss: 0.07
88 - acc: 0.9933
Epoch 23/32
300/300 [==============================] - 0s 95us/step - loss: 0.06
78 - acc: 0.9967
Epoch 24/32
300/300 [==============================] - 0s 96us/step - loss: 0.06
64 - acc: 1.0000
Epoch 25/32
300/300 [==============================] - 0s 86us/step - loss: 0.06
31 - acc: 1.0000
Epoch 26/32
300/300 [==============================] - 0s 89us/step - loss: 0.06
08 - acc: 1.0000
Epoch 27/32
300/300 [==============================] - 0s 93us/step - loss: 0.06
54 - acc: 0.9967
Epoch 28/32
300/300 [==============================] - 0s 95us/step - loss: 0.05
23 - acc: 1.0000
Epoch 29/32
300/300 [==============================] - 0s 85us/step - loss: 0.05
15 - acc: 1.0000
Epoch 30/32
300/300 [==============================] - 0s 88us/step - loss: 0.05
02 - acc: 1.0000
Epoch 31/32
300/300 [==============================] - 0s 102us/step - loss: 0.0
435 - acc: 1.0000
Epoch 32/32
300/300 [==============================] - 0s 96us/step - loss: 0.04
```

Out[146]: <keras.callbacks.History at 0x1855458a90>

In [147]:
```
X_test = generate_a_disk()
X_test = X_test.reshape(1, X_test.shape[0])
model.predict(X_test)
```

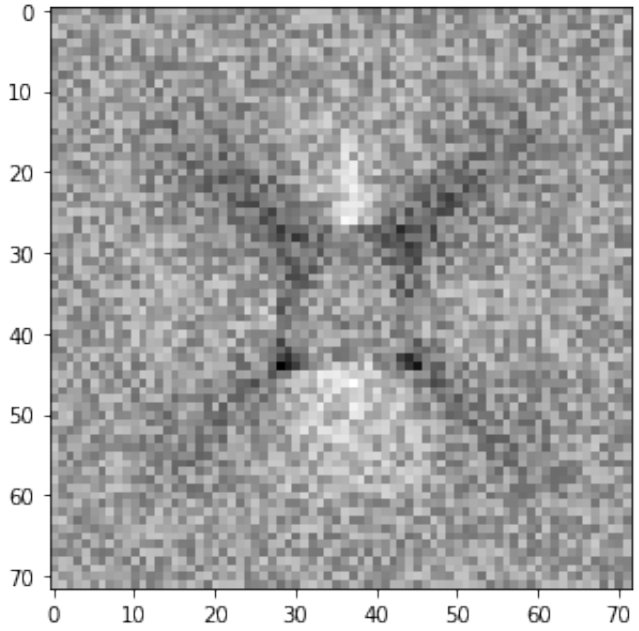
Out[147]: array([[ 0.,  1.,  0.]], dtype=float32)

# Visualization of the Solution

In [148]:
```
# get tjhe maxtrix of the classifie, and visualize the 3 columns
weights = model.get_weights()
print(type(weights))
print(weights[0])
fig, ax = plt.subplots(figsize=(5, 5))
I = weights[0][:,0].reshape((72,72))
ax.imshow(I,cmap='gray')
```

```
<class 'list'>
[[-0.0157772   0.02108267 -0.03320824]
 [-0.01422067 -0.01513903  0.01840402]
 [-0.00332865 -0.0212197   0.01201476]
 ...,
 [ 0.02798267  0.02474933 -0.02355425]
 [-0.03096665  0.00520328 -0.01810184]
 [ 0.01070937 -0.00682686 -0.00869321]]
```

Out[148]: <matplotlib.image.AxesImage at 0x1830910ba8>

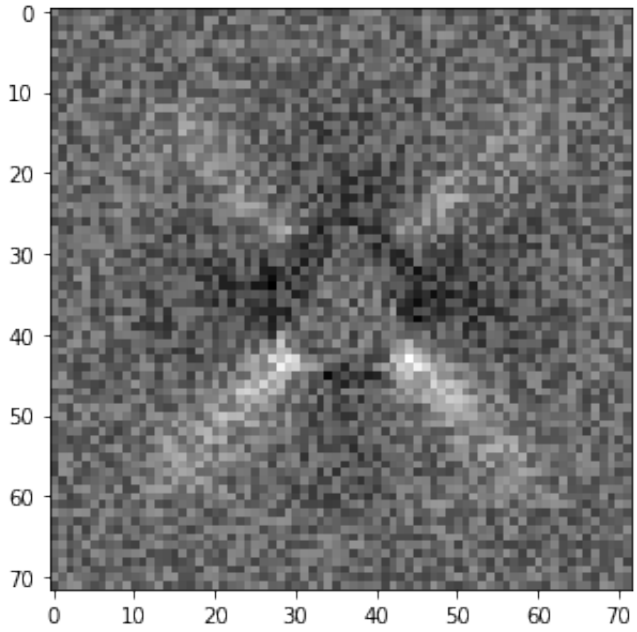


In [149]:
```
for weight in weights:
  print (weight.shape)
```

```
(5184, 3)
(3,)
```

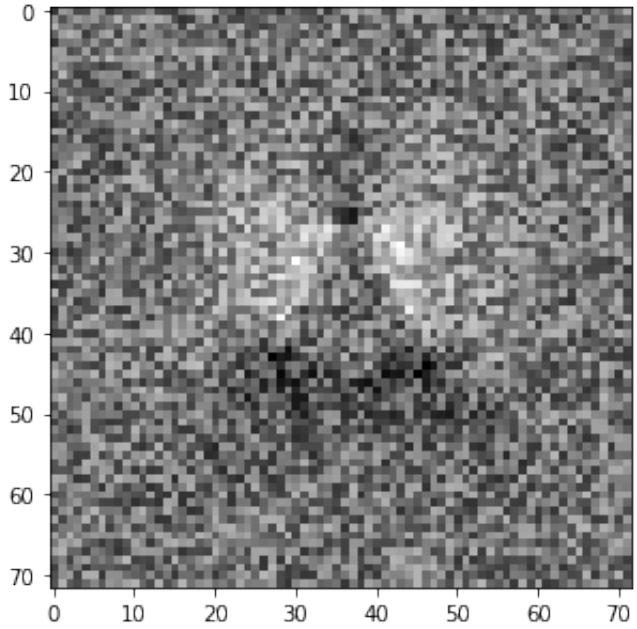
In [150]:
```
weights = model.get_weights()
fig, ax = plt.subplots(figsize=(5, 5))
I = weights[0][:,1].reshape((72,72))
ax.imshow(I,cmap='gray')
```

Out[150]: <matplotlib.image.AxesImage at 0x1830a324e0>



In [151]:
```
weights = model.get_weights()
fig, ax = plt.subplots(figsize=(5, 5))
I = weights[0][:,2].reshape((72,72))
ax.imshow(I,cmap='gray')
```

Out[151]: <matplotlib.image.AxesImage at 0x182f53a9b0>




# A More Difficult Classification Problem

In [152]:
```
#generate the new training set
[X_train, Y_train] = generate_dataset_classification(2000, 20, True)

# transform x and y
Y_train = np_utils.to_categorical (Y_train,3)
X_train = X_train.reshape(X_train.shape[0],1,72,72)
```

```
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
```

In [153]:
```
model_2 = Sequential()
# input convolution layer
model_2.add(Convolution2D(16,(5,5),activation = 'relu',input_shape=(1,
#maxpooling layer
model_2.add(MaxPooling2D(pool_size=(2, 2)))
#Flatten
model_2.add(Flatten())
#full_connected layer
model_2.add(Dense(units=30, activation = 'relu' ))
#ouput layer
model_2.add(Dense(units=3,activation='softmax'))
```

In [154]:
```
#compile the model_2
model_2.compile(loss='categorical_crossentropy',optimizer = 'Adam',met

# fit the model_2
model_2.fit(X_train, Y_train, epochs=32, batch_size=32)
```

```
Epoch 1/32
2000/2000 [==============================] - 15s 8ms/step - loss: 1.
0920 - acc: 0.4850
Epoch 2/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
8092 - acc: 0.6235
Epoch 3/32
```

```
Epoch 3/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
7007 - acc: 0.7045
Epoch 4/32
2000/2000 [==============================] - 15s 8ms/step - loss: 0.
6320 - acc: 0.7480

Epoch 5/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
5781 - acc: 0.7690
Epoch 6/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
5091 - acc: 0.8155: 4s -
Epoch 7/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
4601 - acc: 0.8230
Epoch 8/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
4248 - acc: 0.8520
Epoch 9/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
3573 - acc: 0.8840
Epoch 10/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
2947 - acc: 0.9105
Epoch 11/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
2737 - acc: 0.9170
Epoch 12/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
2427 - acc: 0.9175
Epoch 13/32
2000/2000 [==============================] - 14s 7ms/step - loss: 0.
2019 - acc: 0.9495
Epoch 14/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
1647 - acc: 0.9600
Epoch 15/32
2000/2000 [==============================] - 14s 7ms/step - loss: 0.
1578 - acc: 0.9605
Epoch 16/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
1318 - acc: 0.9685
Epoch 17/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
1084 - acc: 0.9740
Epoch 18/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0993 - acc: 0.9775
Epoch 19/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0796 - acc: 0.9850
Epoch 20/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0892 - acc: 0.9795
```

```
Epoch 21/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0627 - acc: 0.9910
Epoch 22/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.

0527 - acc: 0.9940
Epoch 23/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
0495 - acc: 0.9935
Epoch 24/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
0413 - acc: 0.9965
Epoch 25/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0415 - acc: 0.9950
Epoch 26/32
2000/2000 [==============================] - 12s 6ms/step - loss: 0.
0332 - acc: 0.9965
Epoch 27/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
0293 - acc: 0.9965
Epoch 28/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
0296 - acc: 0.9970
Epoch 29/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
0298 - acc: 0.9975
Epoch 30/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
0228 - acc: 0.9985
Epoch 31/32
2000/2000 [==============================] - 13s 6ms/step - loss: 0.
0191 - acc: 0.9985
Epoch 32/32
2000/2000 [==============================] - 13s 7ms/step - loss: 0.
0194 - acc: 0.9990
```

Out[154]: <keras.callbacks.History at 0x1850cb9ba8>

***model.evaluate(X_test, Y_test)***

In [160]:
```
[X_test, Y_test] = generate_test_set_classification()
X_test = X_test.reshape(X_test.shape[0],1,72,72)
model_2.evaluate(X_test, Y_test)
```

```
Creating data:
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300/300 [==============================] - 1s 4ms/step
```

Out[160]:
```
[0.26112047990163167, 0.92333333412806196]
```

In [ ]:
```

```

# 5 A Regression Problem

In [161]:
```
# generate training set, and reshape the data
[X_train, Y_train] = generate_dataset_regression(300, 20)
X_train = X_train.reshape(-1,1,72,72)
```

```
Creating data:
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
```

In [165]:
```
Y_train[0]
```

Out[165]:
```
array([ 0.71986584,  0.92000602,  0.31926686,  0.97619732,  0.565220
72,
        0.02423682])
```

In [169]:
```
x = Y_train[0]

Y_train_norm=[]
for index in range(len(Y_train)):
    x = Y_train[index]
    xx = sorted([[x[2*i], x[2*i],x[2*i+1]] for i in range(3)])
    Y_train_norm.append([y for x in xx for y in x[1:]] )

Y_train_norm = np.array(Y_train_norm)
```

In [170]:
```
model_3 = Sequential()


# input convolution layer
BatchNormalization(axis=1)
model_3.add(Convolution2D(16,(3,3),activation = 'relu',input_shape=(1,
model_3.add(MaxPooling2D(pool_size=(2, 2)))

BatchNormalization(axis=1)
model_3.add(Convolution2D(32,(3,3),activation = 'relu',padding = 'same
model_3.add(MaxPooling2D(pool_size = (2,2)))

BatchNormalization(axis=1)
model_3.add(Convolution2D(64,(3,3),activation = 'relu',padding = 'same
model_3.add(MaxPooling2D(pool_size = (2,2)))


BatchNormalization(axis=1)
model_3.add(Convolution2D(128,(3,3),activation = 'relu',padding = 'sam
model_3.add(MaxPooling2D(pool_size = (2,2)))

#Flatten
model_3.add(Flatten())
BatchNormalization(axis=1)
model_3.add(Dense(units=200,activation = 'tanh'))
model_3.add(Dense(units=200,activation = 'tanh'))
model_3.add(Dense(units=6,activation='tanh'))
```

In [171]:
```
#compile the model_3
model_3.compile(loss='mean_squared_error',optimizer = 'SGD',metrics =

# fit the model_3
model_3.fit(X_train, Y_train_norm, epochs=20, batch_size=50)
```

```
Epoch 1/20
300/300 [==============================] - 3s 11ms/step - loss: 0.27
46 - acc: 0.4033
Epoch 2/20
300/300 [==============================] - 2s 7ms/step - loss: 0.070
2 - acc: 0.4833
```

```
Epoch 3/20
300/300 [==============================] - 2s 8ms/step - loss: 0.060
8 - acc: 0.4833
Epoch 4/20
300/300 [==============================] - 2s 8ms/step - loss: 0.060
0 - acc: 0.4833
Epoch 5/20
300/300 [==============================] - 2s 7ms/step - loss: 0.059
5 - acc: 0.4833
Epoch 6/20
300/300 [==============================] - 2s 7ms/step - loss: 0.059
5 - acc: 0.4833
Epoch 7/20
300/300 [==============================] - 3s 9ms/step - loss: 0.059
1 - acc: 0.4833
Epoch 8/20
300/300 [==============================] - 3s 8ms/step - loss: 0.059
0 - acc: 0.4833
Epoch 9/20
300/300 [==============================] - 3s 11ms/step - loss: 0.05
88 - acc: 0.4833
Epoch 10/20
300/300 [==============================] - 2s 8ms/step - loss: 0.058
7 - acc: 0.4833
Epoch 11/20
300/300 [==============================] - 2s 8ms/step - loss: 0.058
3 - acc: 0.4833
Epoch 12/20
300/300 [==============================] - 2s 8ms/step - loss: 0.058
2 - acc: 0.4833
Epoch 13/20
300/300 [==============================] - 3s 9ms/step - loss: 0.058
1 - acc: 0.4833
Epoch 14/20
300/300 [==============================] - 3s 9ms/step - loss: 0.058
0 - acc: 0.4833
Epoch 15/20
300/300 [==============================] - 2s 8ms/step - loss: 0.057
9 - acc: 0.4833
Epoch 16/20
300/300 [==============================] - 2s 8ms/step - loss: 0.057
7 - acc: 0.4833
Epoch 17/20
300/300 [==============================] - 3s 9ms/step - loss: 0.057
6 - acc: 0.4833
Epoch 18/20
300/300 [==============================] - 2s 8ms/step - loss: 0.057
4 - acc: 0.4833
Epoch 19/20
300/300 [==============================] - 3s 9ms/step - loss: 0.057
4 - acc: 0.4833
Epoch 20/20
300/300 [==============================] - 3s 8ms/step - loss: 0.057
```

```
3 - acc: 0.4833
```

Out[171]: <keras.callbacks.History at 0x182ada3d30>

In [172]:
```
# this is a problem with the generate_dataset_regression function, whi
#Therefore ,I use the generate_dataset_regression to generate the test


[X_test, Y_test] = generate_dataset_regression(100,20)
print(X_test[0])
print(Y_test[0])
X_test = X_test.reshape(-1,1,72,72)

Y_test_norm=[]
for index in range(len(Y_test)):
    x = Y_test[index]
    xx = sorted([[x[2*i],x[2*i+1]] for i in range(3)])
    Y_test_norm.append([y for x in xx for y in x] )

Y_test_norm = np.array(Y_test_norm)
model_3.evaluate(X_test, Y_test_norm)
```

```
Creating data:
0
10
20
30
40
50
60
70
80
90
[ 0.9968872   0.95936457  0.97097937 ...,  0.96349031  0.93627219
  0.99993519]
[ 0.90433804  0.0996677   0.58621749  0.57567444  0.0282895   0.9292
0882]
100/100 [==============================] - 1s 8ms/step
```

Out[172]: [0.061105605661869046, 0.46000000000000002]

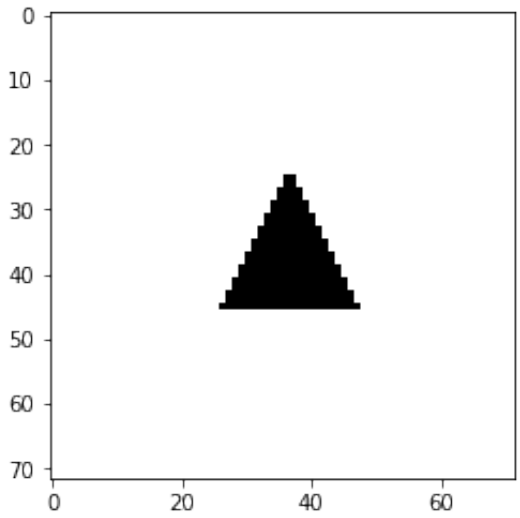
# Bonus Question

In [173]:
```
# change the generate_a_* functions, each function will return two gra


def generate_pari_rectangle(noise=0, free_location=False):
    figsize = 1.0
    U = np.zeros(4)
    V = np.zeros(4)
    if free_location:
        corners = np.random.random(4)
```

```
        top = max(corners[0], corners[1])
        bottom = min(corners[0], corners[1])
        left = min(corners[2], corners[3])
        right = max(corners[2], corners[3])
    else:
        side = (0.3 + 0.7 * np.random.random()) * figsize
        top = figsize/2 + side/2
        bottom = figsize/2 - side/2
        left = bottom
        right = top
    U[0] = U[1] = top
    U[2] = U[3] = bottom
    V[0] = V[3] = left
    V[1] = V[2] = right
    return generate_a_drawing(figsize, U, V,noise=50),generate_a_drawi

def generate_pair_disk(noise=0.0, free_location=False):
    figsize = 1.0
    if free_location:
        center = np.random.random(2)
    else:
        center = (figsize/2, figsize/2)
    radius = (0.3 + 0.7 * np.random.random()) * figsize/2
    N = 50
    U = np.zeros(N)
    V = np.zeros(N)
    i = 0
    for t in np.linspace(0, 2*np.pi, N):
        U[i] = center[0] + np.cos(t) * radius
        V[i] = center[1] + np.sin(t) * radius
        i = i + 1
    return generate_a_drawing(figsize, U, V, noise=50),generate_a_draw

def generate_pair_triangle(noise=0.0, free_location=False):
    figsize = 1.0
    if free_location:
        U = np.random.random(3)
        V = np.random.random(3)
    else:
        size = (0.3 + 0.7 * np.random.random())*figsize/2
        middle = figsize/2
        U = (middle, middle+size, middle-size)
        V = (middle+size, middle-size, middle-size)
    imdata = generate_a_drawing(figsize, U, V, noise=50),generate_a_dra
    return imdata
```

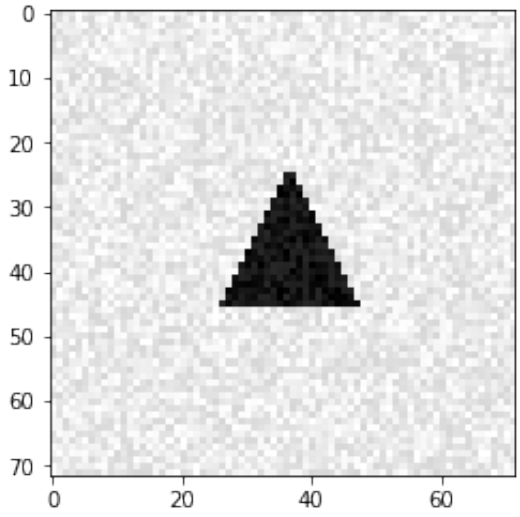
In [174]:
```
# use the generate_pair_triangle to show a pair smaple
[im,im_2] = generate_pair_triangle()

plt.imshow(im_2.reshape(72,72), cmap='gray')
```

Out[174]: <matplotlib.image.AxesImage at 0x186acb2c18>




In [175]:
```
plt.imshow(im.reshape(72,72), cmap='gray')
```

Out[175]: <matplotlib.image.AxesImage at 0x1856b35eb8>



In [183]:
```
# redefine this function, to generate a X_train set ( graph with noise

def generate_dataset_denosing(nb_samples, free_location=False):
    X_train=[]
    Y_train=[]
    print('Creating data:')
    for i in range(nb_samples):
        category = np.random.randint(1,3)
        if category == 0:
          [ima_1,ima_2]=generate_pair_rectangle()
          a = ima_1
          b= ima_2
        elif category == 1:
          [ima_1,ima_2] = generate_pair_disk()
          a = ima_1
          b = ima_2

        else:
          [ima_1,ima_2] = generate_pair_triangle()
          a = ima_1
          b = ima_2
        X_train.append(a)
        Y_train.append(b)
    print ('Data Created')
    return X_train,Y_train
```

In [184]:
```
# generate the training data, and reshape the data
[X_train,Y_train] = generate_dataset_denosing(300)
X_train = np.reshape(X_train,(300,72,72,1))
Y_train = np.reshape(Y_train,(300,72,72,1))
```

```
Creating data:
Data Created
```

In [185]:
```
# build a U-Net Model
def get_crop_shape(target, refer):

  cw = (target.get_shape()[2] - refer.get_shape()[2]).value
  assert (cw >= 0)
  if cw % 2 != 0:
    cw1, cw2 = int(cw/2), int(cw/2) + 1
  else:
    cw1, cw2 = int(cw/2), int(cw/2)
  ch = (target.get_shape()[1] - refer.get_shape()[1]).value
  assert (ch >= 0)
  if ch % 2 != 0:
      ch1, ch2 = int(ch/2), int(ch/2) + 1
  else:
      ch1, ch2 = int(ch/2), int(ch/2)
  return (ch1, ch2), (cw1, cw2)

concat_axis = 3
inputs = Input((72, 72, 1))
```

```
inputs = Input((72,72,1))

conv1 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(i
pool1 = layers.MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(64, (3, 3), padding="same", activation="relu")(pool1)
pool2 = layers.MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(p
pool3 = layers.MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = Conv2D(128, (3, 3), padding="same", activation="relu")(pool3)
pool4 = layers.MaxPooling2D(pool_size=(2, 2))(conv4)
conv5 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(p

up_conv5 = layers.UpSampling2D(size=(2, 2))(conv5)
ch, cw = get_crop_shape(conv4, up_conv5)
crop_conv4 = layers.Cropping2D(cropping=(ch,cw))(conv4)

up6 = layers.concatenate([up_conv5, crop_conv4], axis=concat_axis)
conv6 = Conv2D(256, (3, 3), padding="same", activation="sigmoid", data_
up_conv6 = UpSampling2D(size=(2, 2), data_format="channels_last")(conv
ch, cw = get_crop_shape(conv3, up_conv6)
crop_conv3 = Cropping2D(cropping=(ch,cw), data_format="channels_last")

up7  = concatenate([up_conv6, crop_conv3], axis=concat_axis)
conv7 = Conv2D(128, (3, 3), padding="same", activation="sigmoid", data_
up_conv7 = UpSampling2D(size=(2, 2), data_format="channels_last")(conv
ch, cw = get_crop_shape(conv2, up_conv7)
crop_conv2 = Cropping2D(cropping=(ch,cw), data_format="channels_last")

up8   = concatenate([up_conv7, crop_conv2], axis=concat_axis)
conv8 = Conv2D(64, (3, 3), padding="same", activation="sigmoid", data_
up_conv8 = layers.UpSampling2D(size=(2, 2))(conv8)
ch, cw = get_crop_shape(conv1, up_conv8)
crop_conv1 = layers.Cropping2D(cropping=(ch,cw))(conv1)

up9 = layers.concatenate([up_conv8, crop_conv1], axis=concat_axis)
conv9 = layers.Conv2D(32, (3, 3), activation='sigmoid', padding='same'
ch, cw = get_crop_shape(inputs, conv9)
conv9 = layers.ZeroPadding2D(padding=((ch[0], ch[1]), (cw[0], cw[1])))
conv10 = layers.Conv2D(1, (1, 1))(conv9)

#flatten =  Flatten()(conv10)
model_4 = Model(input=inputs, output=conv10)
model_4.compile(loss='mean_squared_error',optimizer='rmsprop', metrics=
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:65: Use
rWarning: Update your `Model` call to the Keras 2 API: `Model(inputs
```

```
=Tensor("in..., outputs=Tensor("co...)`
```

In [186]:
```
model_4.fit(X_train,Y_train,epochs=20, batch_size=32)
```

```
Epoch 1/20
300/300 [==============================] - 30s 102ms/step - loss: 52
151.3580 - acc: 0.0684
Epoch 2/20
300/300 [==============================] - 28s 93ms/step - loss: 519
90.0401 - acc: 0.0588
Epoch 3/20
300/300 [==============================] - 29s 95ms/step - loss: 519
65.5042 - acc: 0.1685
Epoch 4/20
300/300 [==============================] - 27s 89ms/step - loss: 519
42.7035 - acc: 0.1878
Epoch 5/20
300/300 [==============================] - 25s 83ms/step - loss: 519
20.2876 - acc: 0.1905
Epoch 6/20
300/300 [==============================] - 24s 79ms/step - loss: 518
97.9852 - acc: 0.1945
Epoch 7/20
300/300 [==============================] - 24s 79ms/step - loss: 518
75.8240 - acc: 0.1962
Epoch 8/20
300/300 [==============================] - 24s 78ms/step - loss: 518
53.8347 - acc: 0.1966
Epoch 9/20
300/300 [==============================] - 24s 80ms/step - loss: 518
23.8266 - acc: 0.1970
Epoch 10/20
300/300 [==============================] - 24s 80ms/step - loss: 517
97.4738 - acc: 0.1971
Epoch 11/20
300/300 [==============================] - 23s 77ms/step - loss: 517
72.5152 - acc: 0.1971
Epoch 12/20
300/300 [==============================] - 24s 79ms/step - loss: 517
47.3055 - acc: 0.1971
Epoch 13/20
300/300 [==============================] - 25s 83ms/step - loss: 517
22.0333 - acc: 0.1971
Epoch 14/20
300/300 [==============================] - 24s 80ms/step - loss: 516
96.4715 - acc: 0.1968
Epoch 15/20
300/300 [==============================] - 23s 78ms/step - loss: 516
71.3779 - acc: 0.1971
Epoch 16/20
300/300 [==============================] - 24s 78ms/step - loss: 516
46.4246 - acc: 0.1971
Epoch 17/20
```

```
300/300 [==============================] - 24s 80ms/step - loss: 516
21.4936 - acc: 0.1969
Epoch 18/20
300/300 [==============================] - 23s 78ms/step - loss: 515
96.5389 - acc: 0.1971
Epoch 19/20
300/300 [==============================] - 23s 78ms/step - loss: 515
71.7388 - acc: 0.1971
Epoch 20/20
300/300 [==============================] - 24s 79ms/step - loss: 515
46.5883 - acc: 0.1971
```

Out[186]:
```
<keras.callbacks.History at 0x1838b29f98>
```

In [187]:
```
[X_test,Y_test] = generate_dataset_denosing(100)
X_test = np.reshape(X_test,(100,72,72,1))
Y_test = np.reshape(Y_test,(100,72,72,1))
model_4.evaluate(X_test, Y_test)
```

```
Creating data:
Data Created
100/100 [==============================] - 4s 39ms/step
```

Out[187]:
```
[52317.851718749997, 0.18424768805503844]
```

In [188]:
```
model_4.predit(X_test[0])
```

```
---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-188-9a4b4aa4b3cf> in <module>()
----> 1 model_4.predit(X_test[0])

AttributeError: 'Model' object has no attribute 'predit'
```

In [ ]:
```

```