Hopfield

September 15, 2025

Implementación de Red Hopfield

Integrantes: - Alyson Melissa Sánchez Serratos - Miguel Ángel Pérez Ávila

1 Importación de librerías y Variables Gloabales

[1]:
```
# Importación de librerías
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
```

[2]:
```
# Variables GLobales
test_base_path = "../data/test/"
train_base_path = "../data/train/"

# Factores de escalamiento para las imagenes
scale_factor_x = 0.7
scale_factor_y = 0.7
```

2 Funciones

[3]:
```
# Función de activación con valores donde:
# Si x >= 0: 1
# Si x <  0: 0
def escalonAsimetrico(x):
    if x >=0:
        return 1
    return 0

# Función de activación con valores -1 o 1
# Si x >= 0: 1
# Si x <  0: -1
def escalonSimetrico(x):
    if x >=0:
        return 1
    return -1
```

```
def aplicarEscalonAsimetricoAmatrix(x):
    for i in range(len(x)):
        # Se aplica la función para cada valor del vector
        x[i] = escalonAsimetrico(x[i])
    return x

def aplicarEscalonSimetricoAmatrix(x):
    for i in range(len(x)):
        # Se aplica la función para cada valor del vector
        x[i] = escalonSimetrico(x[i])
    return x
```

[4]:
```
def transform_Image2Array(image_array, actFunction):
    newImage = []
    for i in range(len(image_array)):
        auxRow = []
        for j in range(len(image_array[i])):
            if image_array[i][j] == 255:
                if actFunction == "simetrica":
                    auxRow.append(-1)
                else:
                    auxRow.append(0)
            else:
                auxRow.append(1)
        newImage.append(auxRow)

    return newImage

def flattenArray(mat):
    flat = []
    for i in range(len(mat)): # renglones
        for j in range(len(mat[0])):
            flat.append(mat[i][j])
    flat = np.array(flat, dtype=float)
    return flat

def processImage(path, activationFunction, scale_factor_x, scale_factor_y):
    image_array = cv2.imread(path)

    resized_img = cv2.resize(image_array, None, fx=scale_factor_x,␣
 ↪fy=scale_factor_y)

    image_array = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)

    newImg = np.array(transform_Image2Array(image_array, activationFunction))
    flatImg = flattenArray(newImg)
```

```
    return flatImg

def transform_SimetricBinary2AsimetricBinary(vector):
    newVector=[]
    for i in vector:
        if i == -1:
            newVector.append(0)
        else:
            newVector.append(int(i))
    return newVector

def transform_BinaryVec2str(vector):
    string = ""
    for i in vector:
        string+=str(int(i))
    return string
```

[5]:
```
def random_bool_flip(value, prob=0.2):
    if value:
        return True
    else:
        return np.random.rand() < prob

def transform_Image2Array_withNoise(image_array, actFunction, prob):
    newImage = []
    for i in range(len(image_array)):
        auxRow = []
        for j in range(len(image_array[i])):

            if random_bool_flip(False, prob):
                image_array[i][j] = np.random.choice([255,0], p=[0.5, 0.5])

            if image_array[i][j] == 255:
                if actFunction == "simetrica":
                    auxRow.append(-1)
                else:
                    auxRow.append(0)
            else:
                auxRow.append(1)

        newImage.append(auxRow)

    return newImage

def processImageWithNoise(path, activationFunction, scale_factor_x,␣
 ↪scale_factor_y, prob):
```

```
    image_array = cv2.imread(path)

    resized_img = cv2.resize(image_array, None, fx=scale_factor_x,␣
↪fy=scale_factor_y)

    image_array = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)

    newImg = np.array(transform_Image2Array_withNoise(image_array,␣
↪activationFunction, prob))
    flatImg = flattenArray(newImg)

    return flatImg
```

3 Implementación Hopefield (matriz de pesos)

[6]:
```
def execHopefield(X):
    # len(c) o número de neuronas. EN este caso = 64
    n_neurons = X.shape[1]

    # Dimensiones de W = (len(c), len(c)) = 0(3,3)
    W = np.zeros(shape=(n_neurons, n_neurons))

    """
    Construir matriz W como:
        Wij = | Si i =j : 0
              | Si i!=j : (1/N)*sum(cMi * cMj)

        Wij = matriz de pesos
        N = numero de neuronas o len(C)
        sum = sumatoria iterando para todas las muestras {C0, C1, ..., Cn}
        cMi = muestra número M (de acuerdo con la iteración de la sumatoria) en␣
↪la posición i del vector
        cMj = muestra número M (de acuerdo con la iteración de la sumatoria) en␣
↪la posición j del vector
    """
    for i in range(n_neurons):
        for j in range(n_neurons):
            if i == j:
                W[i, j] = 0
            else:
                acum = 0
                for m in range(X.shape[0]):
                    acum += X[m][i]*X[m][j]
                W[i, j] = (1/n_neurons) * acum

    print("W shape: ", W.shape)
```

```
    print(W)

    return W
```

[7]:
```
def predict(W, input, activationfunction, epochs_control=10000):
    # Definir vector de entrada para la predicción
    y_input = input.T

    epochs_counter = 0
    while True:
        # Calcular U y activarla
        if activationfunction == "simetrica":
            u = aplicarEscalonSimetricoAmatrix(np.dot(W, y_input))
        else:
            u = aplicarEscalonAsimetricoAmatrix(np.dot(W, y_input))

        if np.array_equal(u, y_input) or epochs_counter == epochs_control:
            # Convergencia alcanzada o se alcanzo el numero de
            # epocas de control para vitar ciclos infinitos
            break

        # Convergencia no alcanzada
        y_input = u
        epochs_counter+=1

    return u.T
```

4 Funcion de Pruebas Automatizadas

[8]:
```
def systematicTest(W, sample, activationFunction, pixels):
    aInput = np.array(sample)
    aInput = np.reshape(aInput, (aInput.shape[0],1) )
    input = np.reshape(aInput, shape=(pixels,pixels))

    y_pred = predict(W, aInput.T, activationFunction, 10000)
    print(y_pred)
    output = np.reshape(y_pred, shape=(pixels,pixels))

    if activationFunction == "simetrica":
        input2image = np.where(input == -1, 255, 0).astype(np.uint8)
        output2image = np.where(output == -1, 255, 0).astype(np.uint8)
    else:
        input2image = np.where(input == 0, 255, 0).astype(np.uint8)
        output2image = np.where(output == 0, 255, 0).astype(np.uint8)
```

```
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))

    axes[0].imshow(input2image, cmap='gray', vmin=0, vmax=255)
    axes[0].set_title("Patrón Real")
    axes[0].axis("off")

    axes[1].imshow(output2image, cmap='gray', vmin=0, vmax=255)
    axes[1].set_title("Patrón Predicho")
    axes[1].axis("off")

    display(fig)
    plt.close(fig)
```

5 Entrenamiento Hopfield | Función de Activación Escalón Simétrico

[9]:
```
# Funcion de activación
activationFunction = "simetrica"
```

5.1 Lectura de Archivos

[10]:
```
# Data sets
Xtrain = []
Ytrain = []
Ytest= []
Xtest = []

# Construcción de conjunto de prueba
tagIdxDict_test = {}
index = 0
for file in sorted(os.listdir(test_base_path)):
        # Procesamiento de imagenes
        image = processImage(test_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y)
        Xtest.append(image)

        tag = file[0]

        # Obtener etiqueta en notación binaria
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
```

```
            else:
                vector.append(int(i))

        tagIdxDict_test[tag_binario] = tag

        # Añadir la etiqueta al conjunto Y
        Ytest.append(vector)
        index+=1

# Construcción de conjunto de entrenamiento
tagIdxDict_train = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        # Procesamiento de imagenes
        image = processImage(train_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y)
        Xtrain.append(image)

        tag = file[0]

        # Obtener etiqueta en notación binaria
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_train[tag_binario] = tag

        # Añadir la etiqueta al conjunto Y
        Ytrain.append(vector)
        index+=1

# Cast de list() a np.array()
Xtrain = np.array(Xtrain)
Xtest = np.array(Xtest)
Ytrain = np.array(Ytrain)
Ytest = np.array(Ytest)

print(Xtrain.shape)
print(Xtest.shape)
print(Ytrain.shape)
print(Ytest.shape)
```

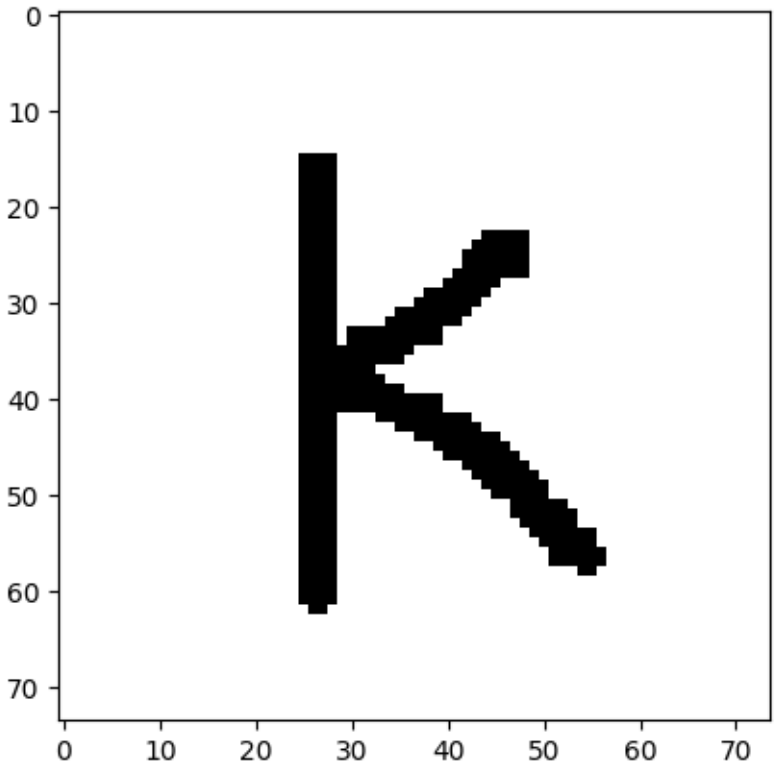
```
(26, 5476)
(26, 5476)
(26, 5)
(26, 5)
```

[11]:
```
# Mostrar ejemplo de muestra
pixels = 74
input = np.reshape(Xtrain[10], shape=(pixels,pixels))

input2image = np.where(input == -1, 255, 0).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

[11]:
```
<matplotlib.image.AxesImage at 0x7fa02c171850>
```



[12]:
```
# Mostrar tags y valores del conjunto de datos
for key, value in tagIdxDict_train.items():
    print("Tag: ", key, "    |  Value: ", value)
```

```
Tag:  00000    |  Value:  a
Tag:  00001    |  Value:  b
Tag:  00010    |  Value:  c
Tag:  00011    |  Value:  d
```

```
Tag:  00100    |  Value:  e
Tag:  00101    |  Value:  f
Tag:  00110    |  Value:  g
Tag:  00111    |  Value:  h
Tag:  01000    |  Value:  i
Tag:  01001    |  Value:  j
Tag:  01010    |  Value:  k
Tag:  01011    |  Value:  l
Tag:  01100    |  Value:  m
Tag:  01101    |  Value:  n
Tag:  01110    |  Value:  o
Tag:  01111    |  Value:  p
Tag:  10000    |  Value:  q
Tag:  10001    |  Value:  r
Tag:  10010    |  Value:  s
Tag:  10011    |  Value:  t
Tag:  10100    |  Value:  u
Tag:  10101    |  Value:  v
Tag:  10110    |  Value:  w
Tag:  10111    |  Value:  x
Tag:  11000    |  Value:  y
Tag:  11001    |  Value:  z
```

5.2 Separación de Conjuntos

[13]:
```python
# Indices de separación para el conjunto de prueba y entrenamiento
idx = 10
idx_sup = 14
m = idx_sup-idx

Xtrain_simetric = Xtrain[idx:idx_sup, :]
Xtest_simetric = Xtest[idx:idx_sup, :]
Ytrain_simetric = Ytrain[idx:idx_sup]
Ytest_simetric = Ytest[idx:idx_sup]
print(Xtrain_simetric.shape)
print(Xtest_simetric.shape)
print(Ytrain_simetric.shape)
print(Ytest_simetric.shape)
```

```
(4, 5476)
(4, 5476)
(4, 5)
(4, 5)
```

5.3 Entrenamiento

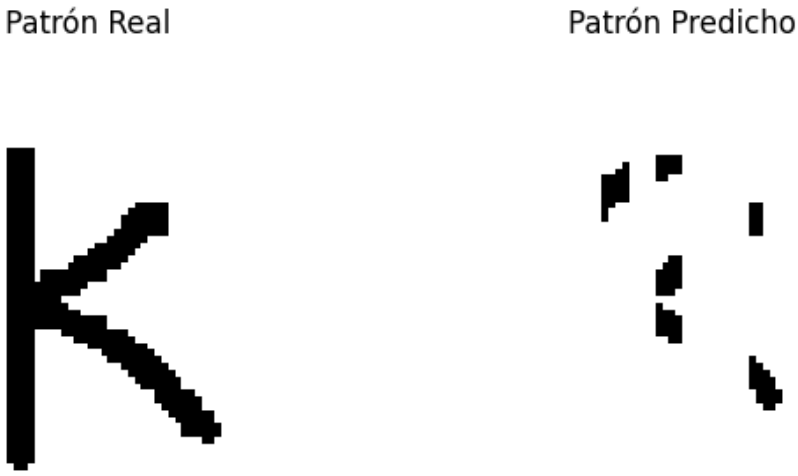
[ ]:
```
optimizedW = execHopefield(Xtrain_simetric)
```

5.4 Predicciones

5.4.1 Utilizando las Muestras de Entrenamiento

[15]:
```
sample_M = 0
systematicTest(optimizedW, Xtrain_simetric[sample_M], activationFunction,␣
↪pixels)
```

```
[[-1. -1. -1. … -1. -1. -1.]]
```



[16]:
```
M=m
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtrain_simetric[m], activationFunction, pixels)
```

```
================ Test:  0
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

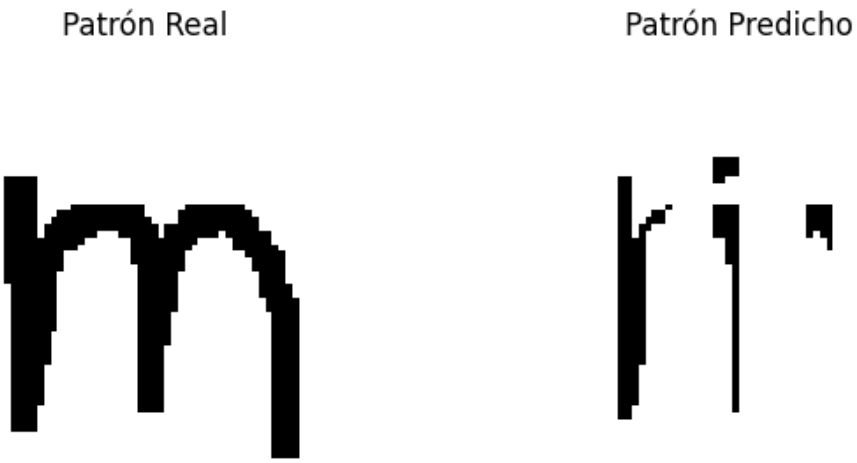
Patrón Predicho

```
================ Test:  1
[[-1. -1. -1. … -1. -1. -1.]]
```

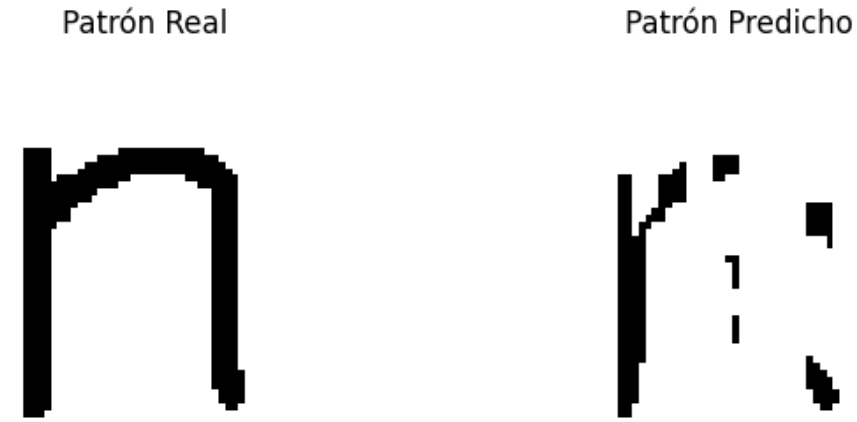
Patrón Real

Patrón Predicho

```
================ Test:  2
[[-1. -1. -1. … -1. -1. -1.]]
```




```
================ Test:  3
[[-1. -1. -1. … -1. -1. -1.]]
```



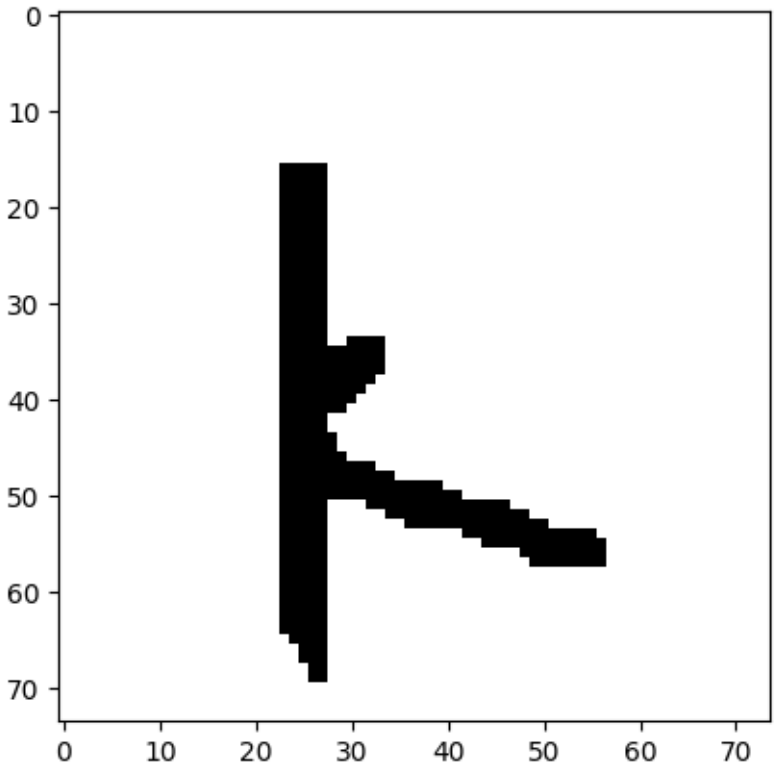
5.4.2 Utilizando las Muestras de Prueba - Ruido de Forma

[17]:
```
input = np.reshape(Xtest_simetric[0], shape=(pixels,pixels))

input2image = np.where(input == -1, 255, 0).astype(np.uint8)
```

```
plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

[17]: <matplotlib.image.AxesImage at 0x7f9fa026fbd0>



[18]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_simetric[m], activationFunction, pixels)
```

```
================ Test:  0
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

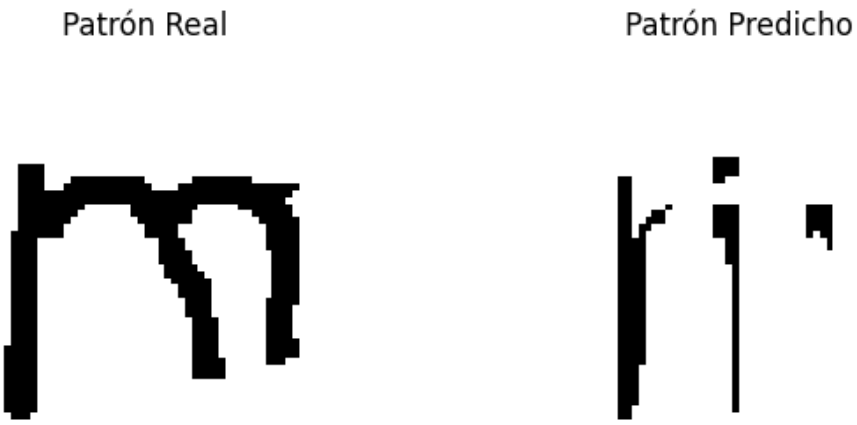
Patrón Predicho

```
================ Test:  1
[[-1. -1. -1. … -1. -1. -1.]]
```

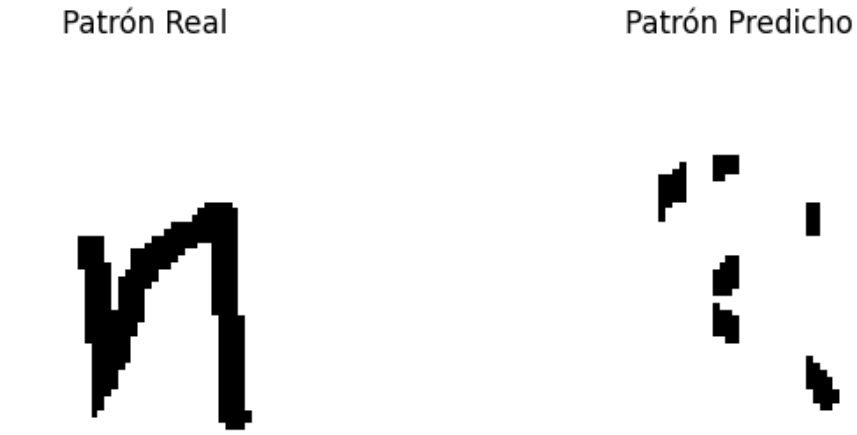
Patrón Real

Patrón Predicho

```
================ Test:  2
[[-1. -1. -1. … -1. -1. -1.]]
```




```
================ Test:  3
[[-1. -1. -1. … -1. -1. -1.]]
```



5.4.3 Utilizando las Muestras de Prueba - Ruido de Bits aleatorios - Factor 0.2

Transformar Muestras de entrenamiento a añadiendo ruido a las imagenes de acuerdo con una probabilidad de activación para cada pixel

[19]:
```
Xtest_noise_simetric = []
Ytest_noise_simetric = []
```

```
noiseProb = 0.2

# Construcción de conjunto de entrenamiento
tagIdxDict_test_noise = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        image = processImageWithNoise(train_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y, noiseProb)
        Xtest_noise_simetric.append(image)

        tag = file[0]

        # Obtener etiqueta en formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_test_noise[tag_binario] = tag

        # Añadir etiqueta al vector Y
        Ytest_noise_simetric.append(vector)
        index+=1

Xtest_noise_simetric = np.array(Xtest_noise_simetric)
Ytest_noise_simetric = np.array(Ytest_noise_simetric)

print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

```
(26, 5476)
(26, 5)
```

[20]:
```
Xtest_noise_simetric = Xtest_noise_simetric[idx:idx_sup, :]
Ytest_noise_simetric = Ytest_noise_simetric[idx:idx_sup]
print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

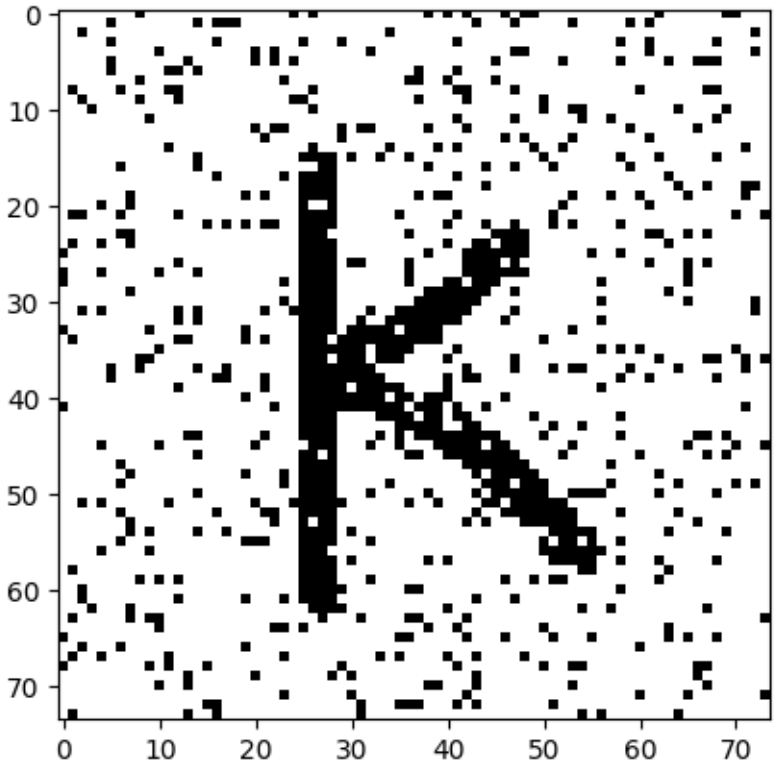
```
(4, 5476)
(4, 5)
```

[21]:
```
input = np.reshape(Xtest_noise_simetric[0], shape=(pixels,pixels))

input2image = np.where(input == -1, 255, 0).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

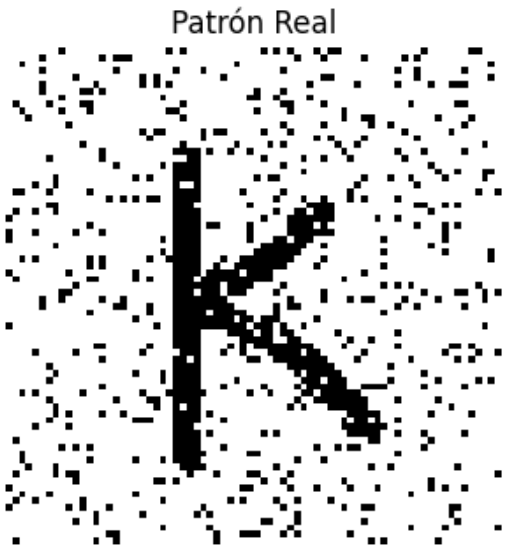
[21]:
```
<matplotlib.image.AxesImage at 0x7f9faea82e10>
```



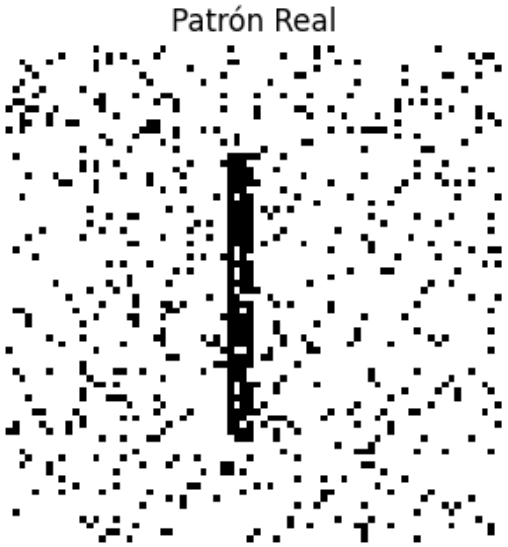
Ejecución de Pruebas

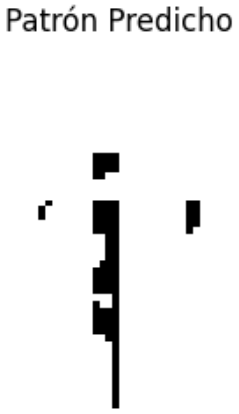
[22]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_noise_simetric[m], activationFunction,␣
 ↪pixels)
```

```
================ Test:  0
[[-1. -1. -1. … -1. -1. -1.]]
```

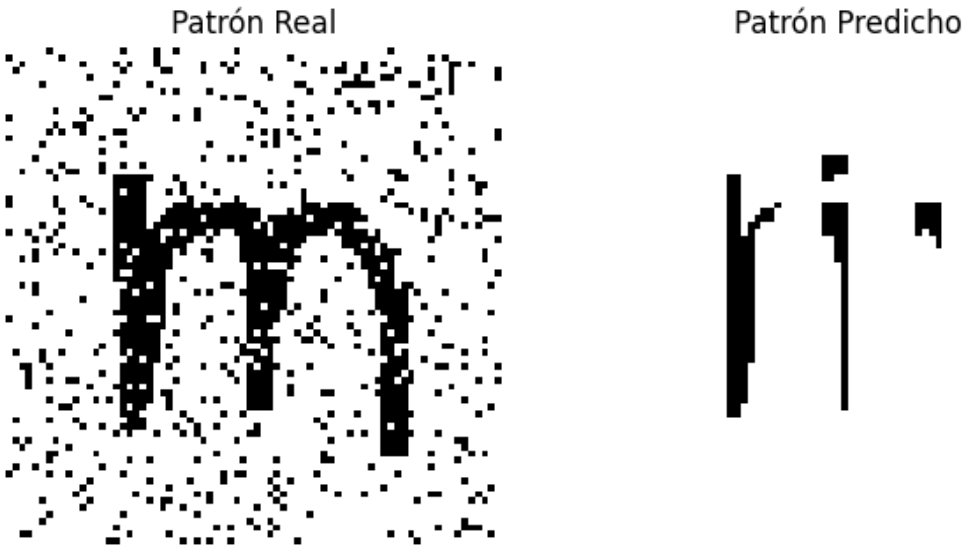



```
================ Test:  1
[[-1. -1. -1. … -1. -1. -1.]]
```

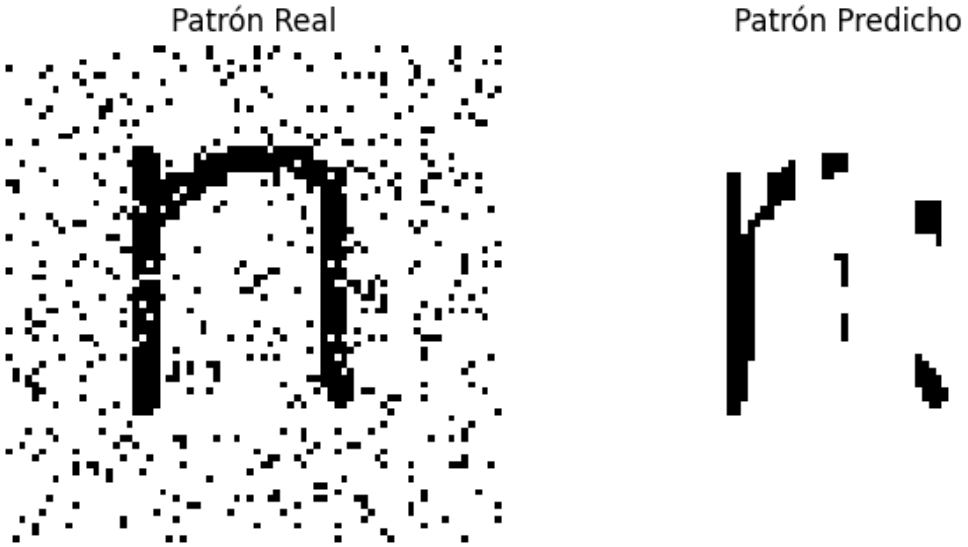





```
================ Test:  2
[[-1. -1. -1. … -1. -1. -1.]]
```




```
================ Test:  3
[[-1. -1. -1. … -1. -1. -1.]]
```



5.4.4 Utilizando las Muestras de Prueba - Ruido de Bits aleatorios - Factor 0.75

Transformar Muestras de entrenamiento a añadiendo ruido a las imagenes de acuerdo con una probabilidad de activación para cada pixel

[23]:
```
Xtest_noise_simetric = []
Ytest_noise_simetric = []
```

```
noiseProb = 0.75

# Construcción de conjunto de entrenamiento
tagIdxDict_test_noise = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        image = processImageWithNoise(train_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y, noiseProb)
        Xtest_noise_simetric.append(image)

        tag = file[0]

        # Obtener etiqueta en formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_test_noise[tag_binario] = tag

        # Añadir etiqueta al conjunto Y
        Ytest_noise_simetric.append(vector)
        index+=1

Xtest_noise_simetric = np.array(Xtest_noise_simetric)
Ytest_noise_simetric = np.array(Ytest_noise_simetric)

print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

```
(26, 5476)
(26, 5)
```

[24]:
```
Xtest_noise_simetric = Xtest_noise_simetric[idx:idx_sup, :]
Ytest_noise_simetric = Ytest_noise_simetric[idx:idx_sup]
print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

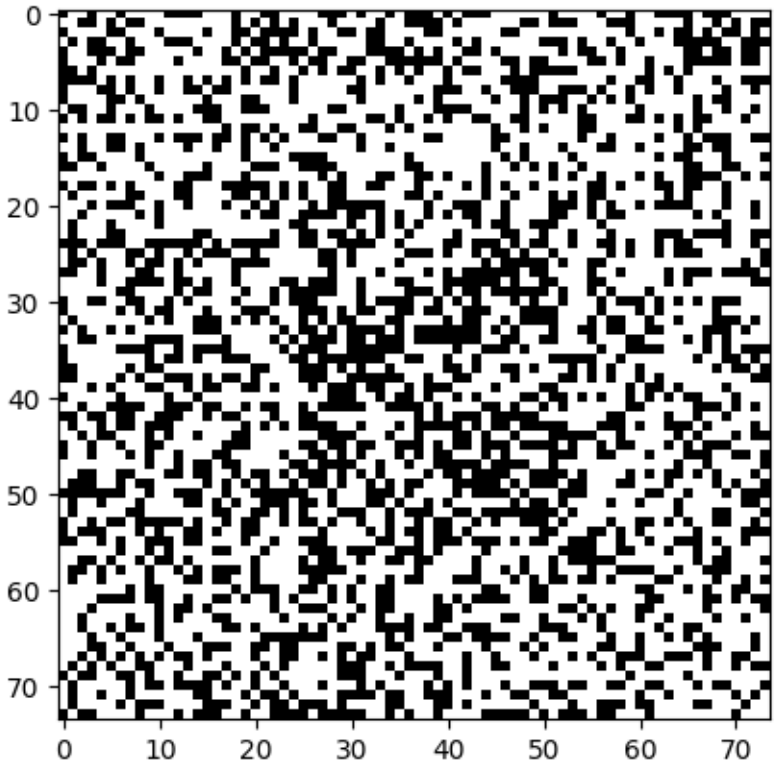
```
(4, 5476)
(4, 5)
```

[25]:
```
input = np.reshape(Xtest_noise_simetric[0], shape=(pixels,pixels))

input2image = np.where(input == -1, 255, 0).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

[25]:
```
<matplotlib.image.AxesImage at 0x7f9f9fe2d450>
```



Ejecución de Pruebas

[26]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_noise_simetric[m], activationFunction,␣
 ↪pixels)
```

```
================ Test:  0
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

Patrón Predicho

```
================ Test:  1
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

Patrón Predicho

```
================ Test:  2
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

Patrón Predicho

```
================ Test:  3
[[-1. -1. -1. … -1. -1. -1.]]
```

Patrón Real

Patrón Predicho

6 Entrenamiento Hopfield | Función de Activación Escalón Asimétrico

[27]:
```
# Función de activación
activationFunction = "asimetrica"
```

6.1 Lectura de Archivos

[28]:
```
# Data sets
Xtrain = []
Ytrain = []
Ytest= []
Xtest = []

# Construcción de conjunto de prueba
tagIdxDict_test = {}
index = 0
for file in sorted(os.listdir(test_base_path)):
        # Procesamiento de imagenes
        image = processImage(test_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y)
        Xtest.append(image)

        tag = file[0]

        # Obtener etiqueta en formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_test[tag_binario] = tag

        # Añadir etiqueta a vector Y
        Ytest.append(vector)
        index+=1

# Construcción de conjunto de entrenamiento
tagIdxDict_train = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        # Procesamiento de imagenes
```

```
        image = processImage(train_base_path+file, activationFunction,␣
↪scale_factor_x, scale_factor_y)
        Xtrain.append(image)

        tag = file[0]

        # Obtener etiqueta en formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_train[tag_binario] = tag

        # Añadir etiqueta a vector Y
        Ytrain.append(vector)
        index+=1

Xtrain = np.array(Xtrain)
Xtest = np.array(Xtest)
Ytrain = np.array(Ytrain)
Ytest = np.array(Ytest)

print(Xtrain.shape)
print(Xtest.shape)
print(Ytrain.shape)
print(Ytest.shape)
```

```
(26, 5476)
(26, 5476)
(26, 5)
(26, 5)
```

[29]:
```
Xtrain_asimetric = Xtrain[idx:idx_sup, :]
Xtest_asimetric = Xtest[idx:idx_sup, :]
Ytrain_asimetric = Ytrain[idx:idx_sup]
Ytest_asimetric = Ytest[idx:idx_sup]
print(Xtrain_asimetric.shape)
print(Xtest_asimetric.shape)
print(Ytrain_asimetric.shape)
print(Ytest_asimetric.shape)
```

```
(4, 5476)
```

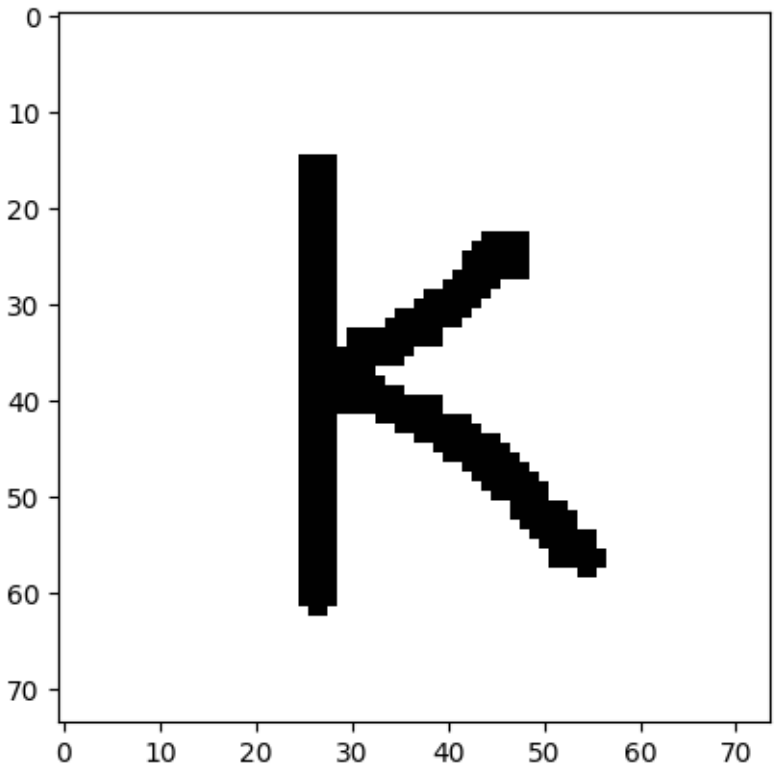
```
(4, 5476)
(4, 5)
(4, 5)
```

[30]:
```
input = np.reshape(Xtrain_asimetric[0], shape=(pixels,pixels))

input2image = np.where(input == 0, 255, 1).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

[30]:
```
<matplotlib.image.AxesImage at 0x7f9f9fb2a910>
```



6.2 Entrenamiento

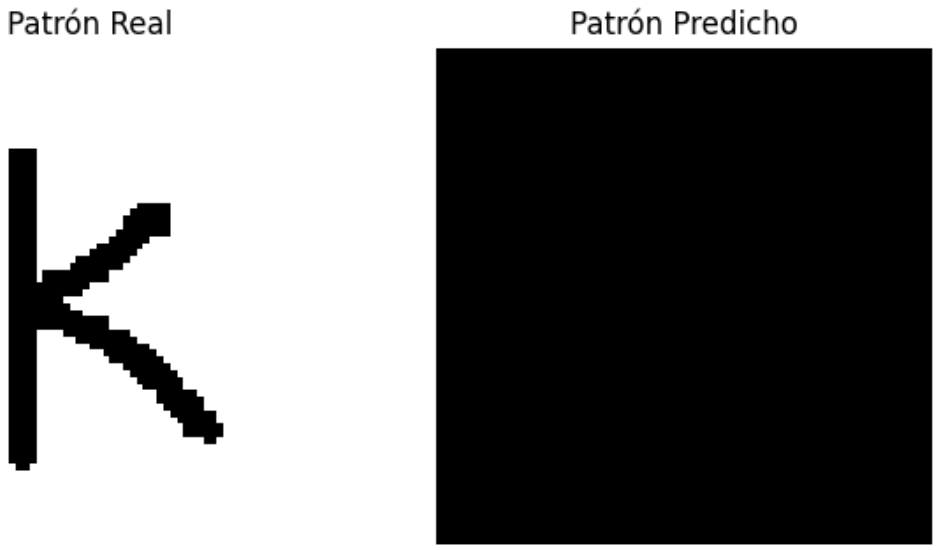
[ ]:
```
optimizedW = execHopefield(Xtrain_asimetric)
```

6.3 Predicciones

6.3.1 Utilizando las Muestras de Entrenamiento

[32]:
```
sample_M = 0
systematicTest(optimizedW, Xtrain_asimetric[sample_M], activationFunction,␣
↪pixels)
```

```
[[1. 1. 1. … 1. 1. 1.]]
```



[33]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtrain_asimetric[m], activationFunction, pixels)
```

```
================ Test:  0
[[1. 1. 1. … 1. 1. 1.]]
```

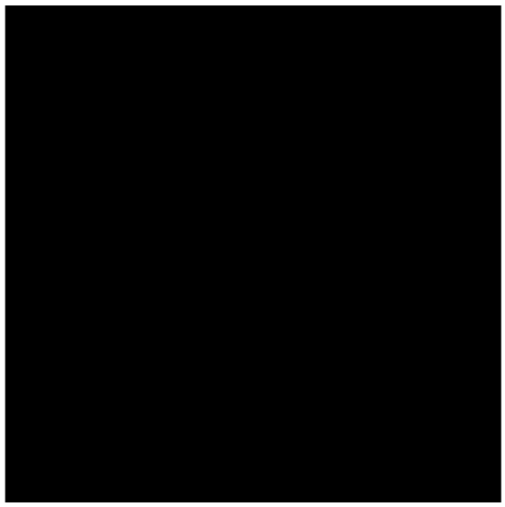
Patrón Real

Patrón Predicho

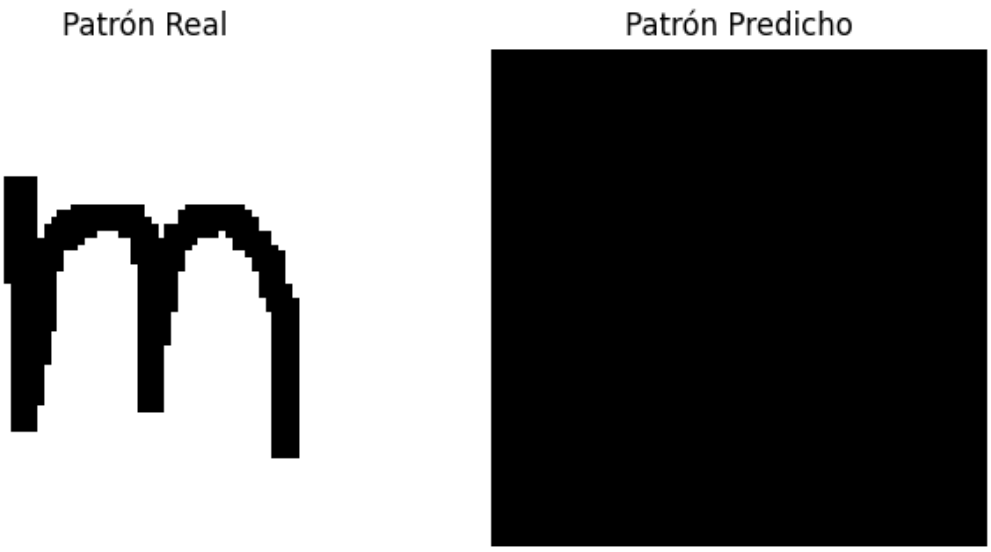
```
================ Test:  1
[[1. 1. 1. … 1. 1. 1.]]
```

Patrón Real

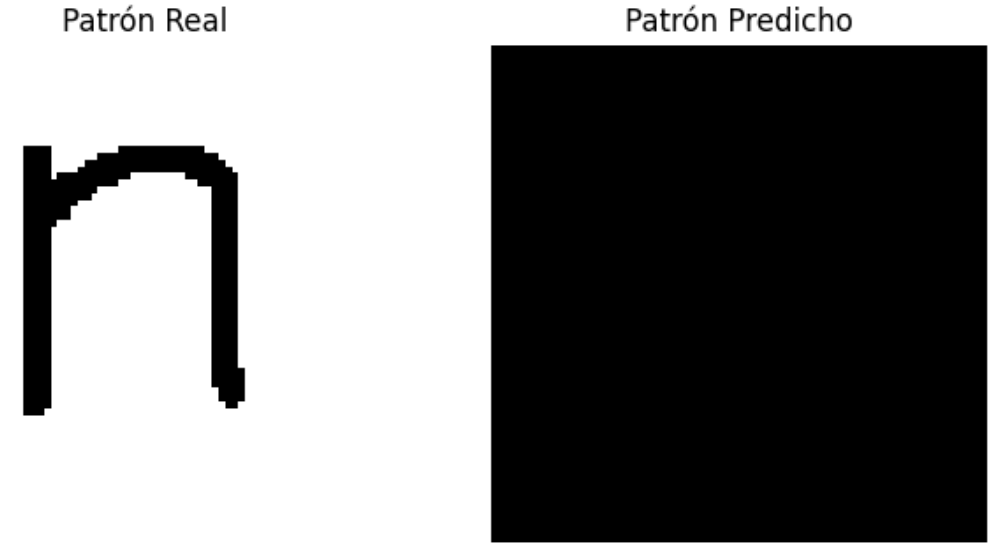
Patrón Predicho



```
================ Test:  2
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  3
[[1. 1. 1. … 1. 1. 1.]]
```



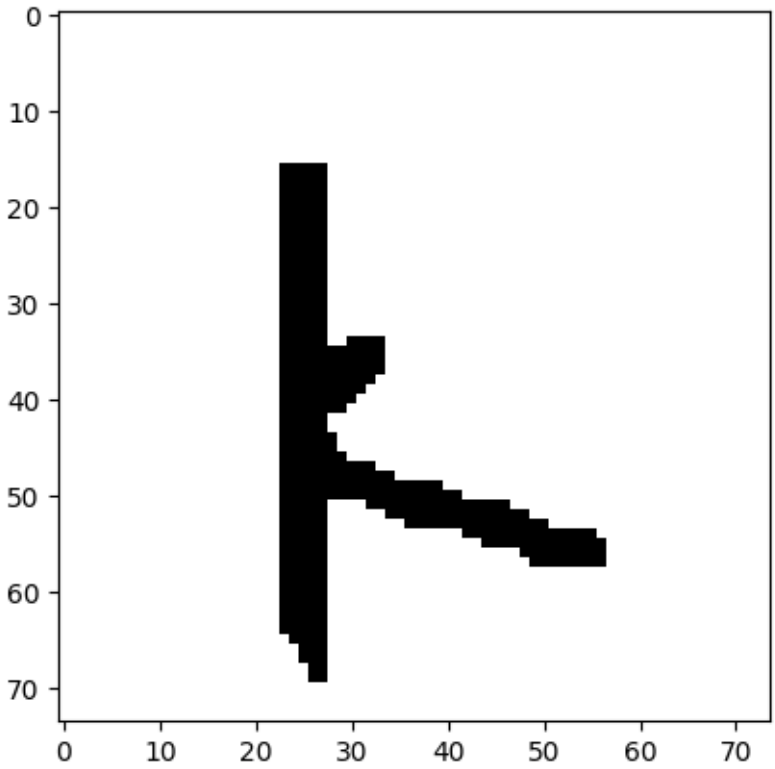
6.3.2 Utilizando las Muestras de Prueba - Ruido de Forma

[34]:
```
input = np.reshape(Xtest_asimetric[0], shape=(pixels,pixels))

input2image = np.where(input == 0, 255, 0).astype(np.uint8)
```

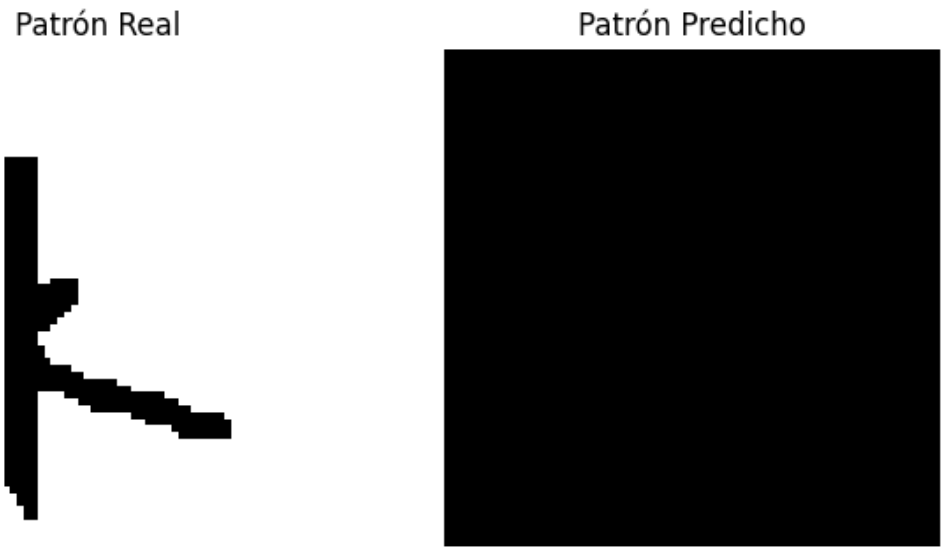
```
plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

[34]: <matplotlib.image.AxesImage at 0x7f9f9fea2510>

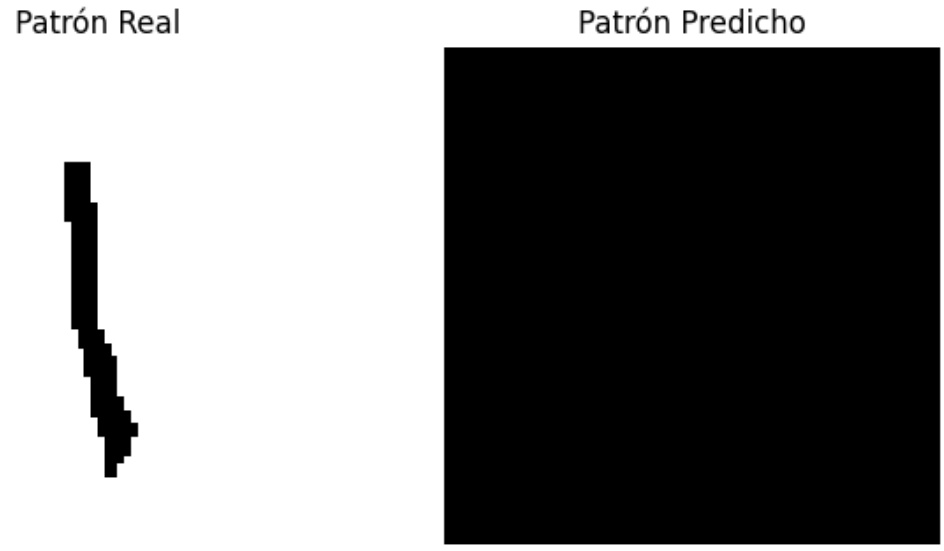


[35]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_asimetric[m], activationFunction, pixels)
```

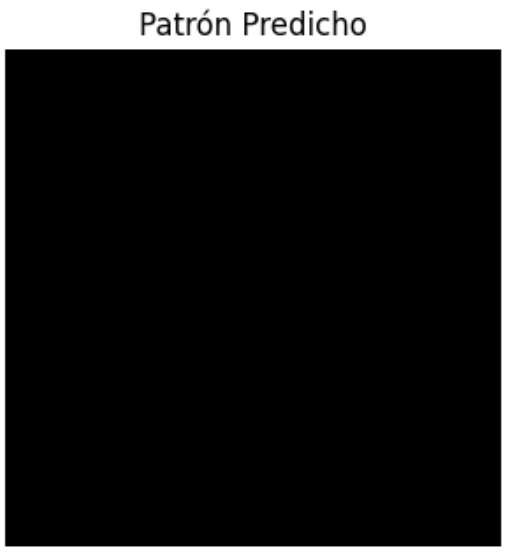
```
================ Test:  0
[[1. 1. 1. … 1. 1. 1.]]
```



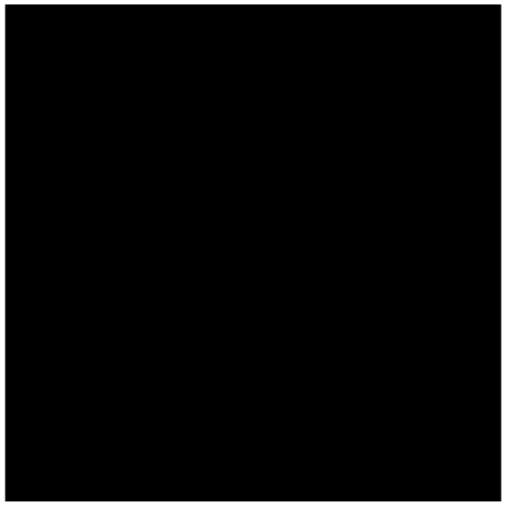

```
================ Test:  1
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  2
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  3
[[1. 1. 1. … 1. 1. 1.]]
```



6.3.3 Utilizando las Muestras de Prueba - Ruido de Bits aleatorios - Factor 0.2

Transformar Muestras de entrenamiento a añadiendo ruido a las imagenes de acuerdo con una probabilidad de activación para cada pixel

[36]:
```
Xtest_noise_asimetric = []
Ytest_noise_asimetric = []
```

```
noiseProb = 0.2

# COnstrucción de conjunto de prueba con ruido
tagIdxDict_test_noise = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        image = processImageWithNoise(train_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y, noiseProb)
        Xtest_noise_asimetric.append(image)

        tag = file[0]

        # Obtener etiqueta en formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_test_noise[tag_binario] = tag

        # Añadir etiqueta al conjunto Y
        Ytest_noise_asimetric.append(vector)
        index+=1

Xtest_noise_asimetric = np.array(Xtest_noise_asimetric)
Ytest_noise_asimetric = np.array(Ytest_noise_asimetric)

print(Xtest_noise_asimetric.shape)
print(Ytest_noise_asimetric.shape)
```

```
(26, 5476)
(26, 5)
```

[37]:
```
Xtest_noise_asimetric = Xtest_noise_asimetric[idx:idx_sup, :]
Ytest_noise_asimetric = Ytest_noise_asimetric[idx:idx_sup]
print(Xtest_noise_asimetric.shape)
print(Ytest_noise_asimetric.shape)
```

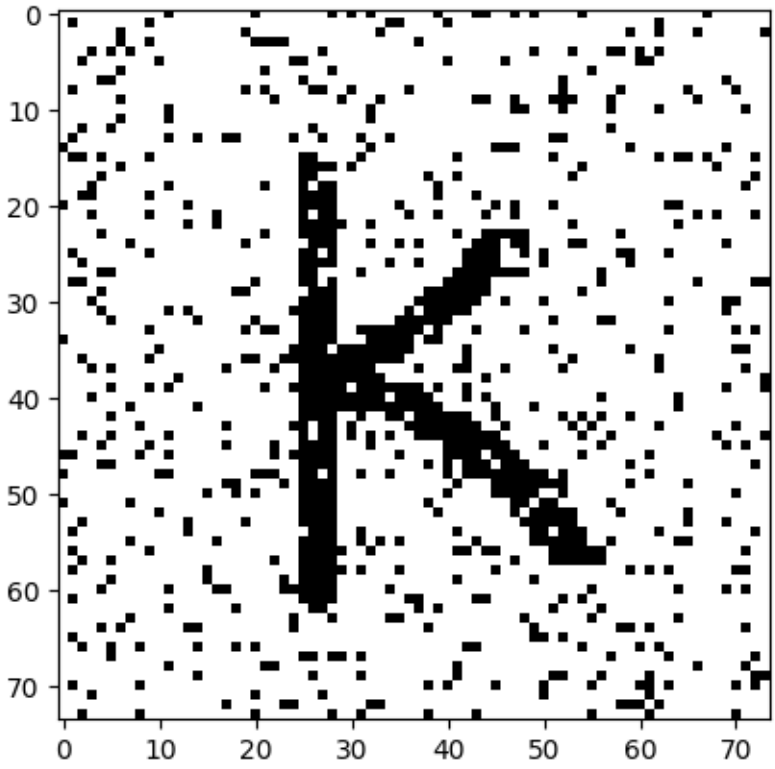
```
(4, 5476)
(4, 5)
```

[38]:
```
input = np.reshape(Xtest_noise_asimetric[0], shape=(pixels,pixels))

input2image = np.where(input == 0, 255, 0).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

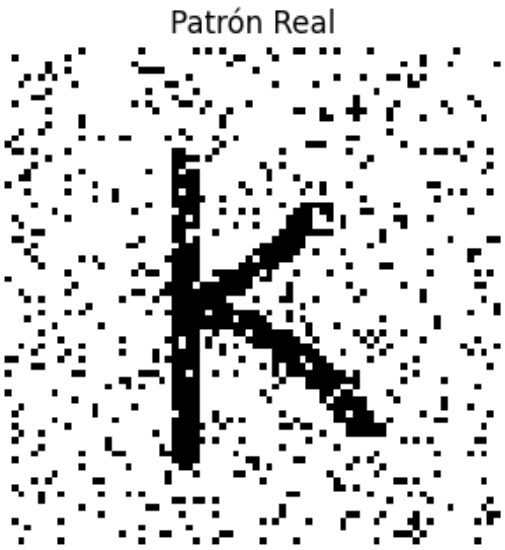
[38]:
```
<matplotlib.image.AxesImage at 0x7f9fa030c0d0>
```

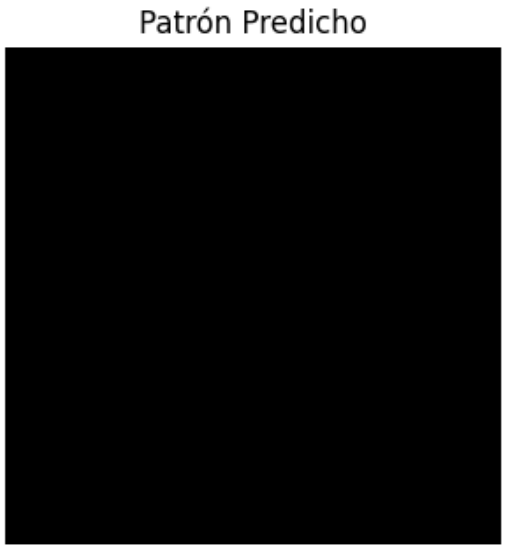


Ejecución de Pruebas

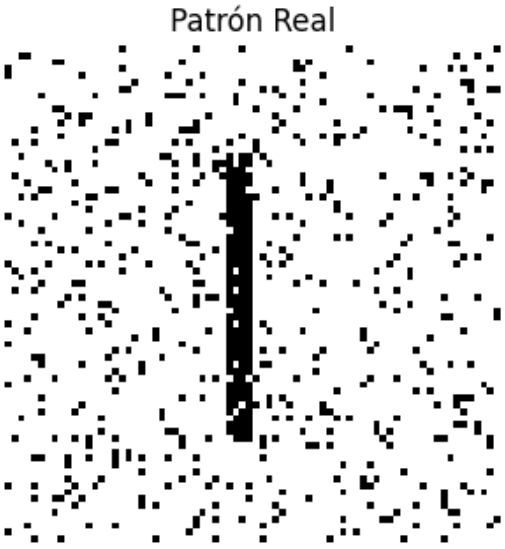
[39]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_noise_asimetric[m], activationFunction,␣
 ↪pixels)
```

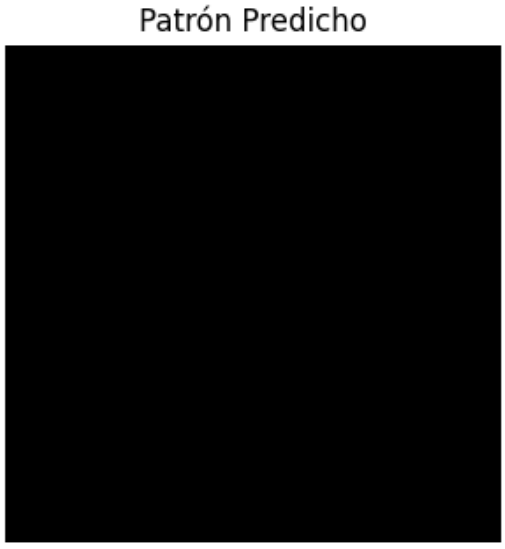
```
================ Test:  0
[[1. 1. 1. … 1. 1. 1.]]
```



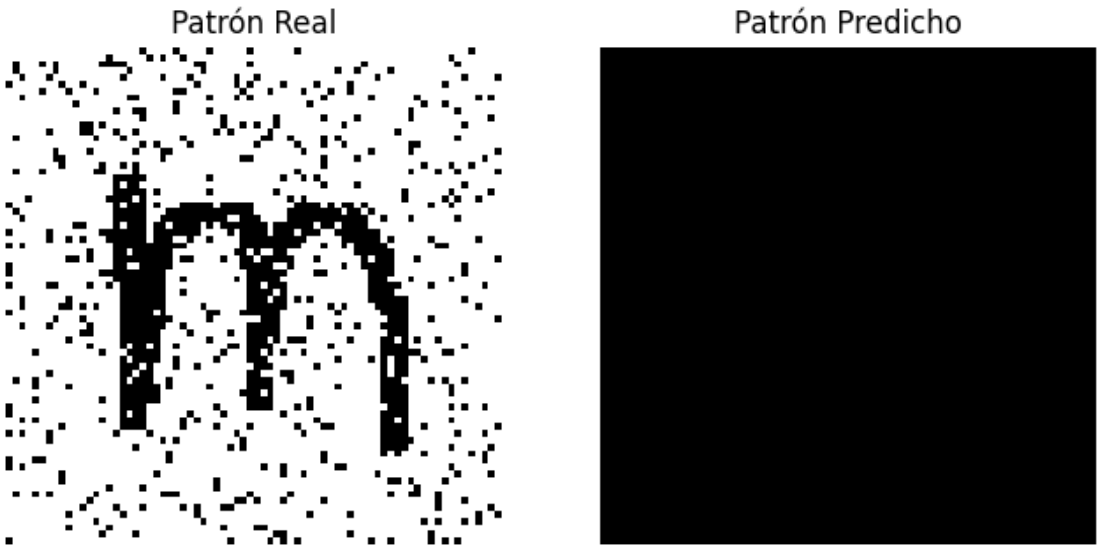



```
================ Test:  1
[[1. 1. 1. … 1. 1. 1.]]
```

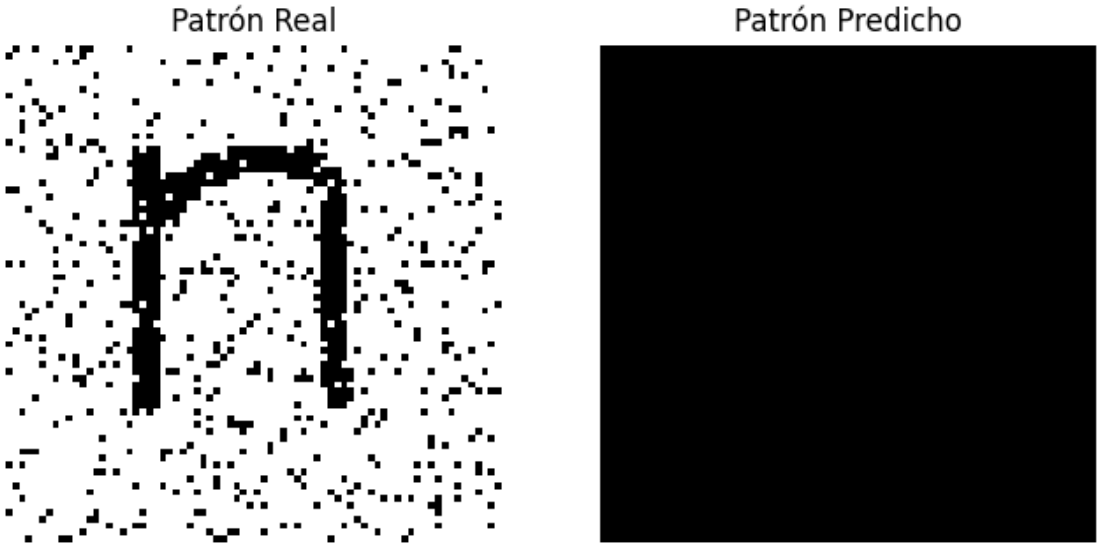





```
================ Test:  2
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  3
[[1. 1. 1. … 1. 1. 1.]]
```



6.3.4 Utilizando las Muestras de Prueba - Ruido de Bits aleatorios - Factor 0.75

Transformar Muestras de entrenamiento a añadiendo ruido a las imagenes de acuerdo con una probabilidad de activación para cada pixel

[40]:
```
Xtest_noise_simetric = []
Ytest_noise_simetric = []
```

```
noiseProb = 0.75

# Construcción del conjunto de prueba con ruido
tagIdxDict_test_noise = {}
index = 0
for file in sorted(os.listdir(train_base_path)):
        # Procesamiento de imagenes
        image = processImageWithNoise(train_base_path+file, activationFunction,␣
 ↪scale_factor_x, scale_factor_y, noiseProb)
        Xtest_noise_simetric.append(image)

        tag = file[0]

        # Obtener etiqueta con formato binario
        tag_binario = format(index, '05b')

        # Tag_binario -> Vector
        vector = []
        for i in tag_binario:
            if i == "0" and activationFunction == "simetrica":
                vector.append(-1)
            else:
                vector.append(int(i))

        tagIdxDict_test_noise[tag_binario] = tag

        # Añadir etiqueta al conjunto Y
        Ytest_noise_simetric.append(vector)
        index+=1

Xtest_noise_simetric = np.array(Xtest_noise_simetric)
Ytest_noise_simetric = np.array(Ytest_noise_simetric)

print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

```
(26, 5476)
(26, 5)
```

[41]:
```
Xtest_noise_simetric = Xtest_noise_simetric[idx:idx_sup, :]
Ytest_noise_simetric = Ytest_noise_simetric[idx:idx_sup]
print(Xtest_noise_simetric.shape)
print(Ytest_noise_simetric.shape)
```

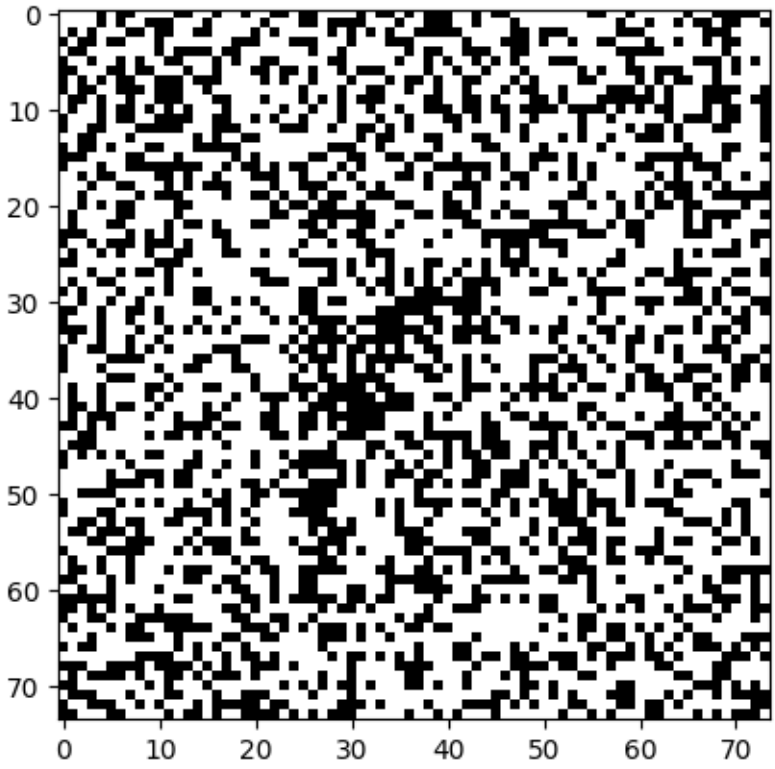
```
(4, 5476)
(4, 5)
```

[42]:
```
input = np.reshape(Xtest_noise_simetric[0], shape=(pixels,pixels))

input2image = np.where(input == 0, 255, 0).astype(np.uint8)

plt.imshow(input2image, cmap='gray', vmin=0, vmax=255)
```

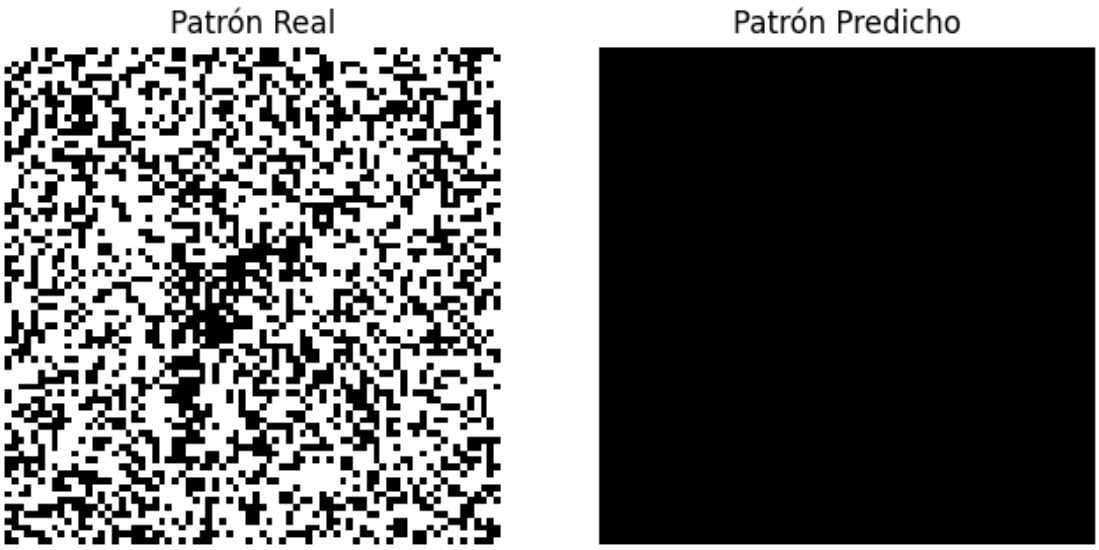
[42]:
```
<matplotlib.image.AxesImage at 0x7f9fae96b790>
```



Ejecución de Pruebas

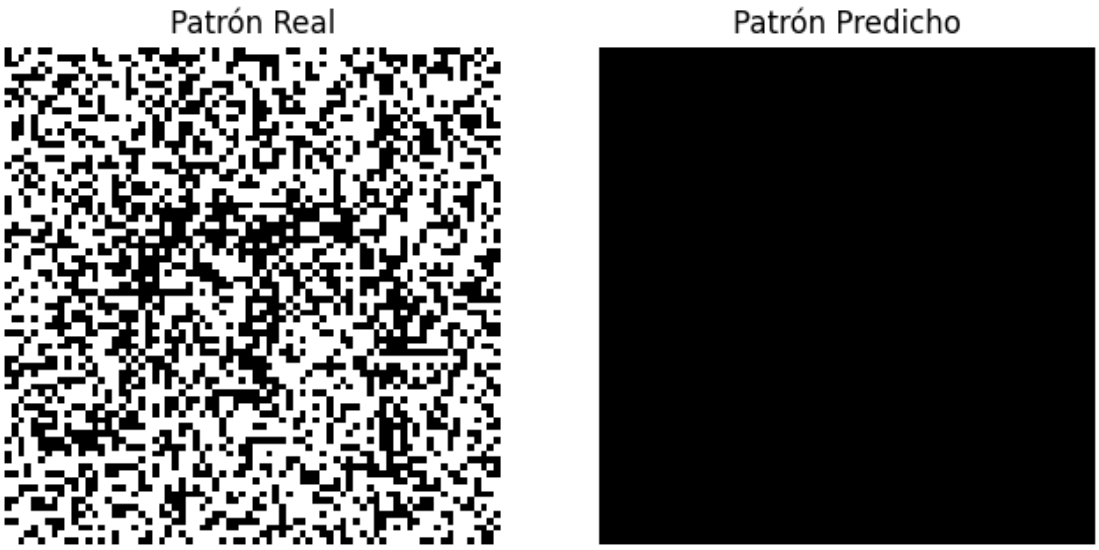
[43]:
```
for m in range(M):
    print("================ Test: ", m)
    systematicTest(optimizedW, Xtest_noise_simetric[m], activationFunction,␣
 ↪pixels)
```

```
================ Test:  0
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  1
[[1. 1. 1. … 1. 1. 1.]]
```


```
================ Test:  2
[[1. 1. 1. … 1. 1. 1.]]
```




```
================ Test:  3
[[1. 1. 1. … 1. 1. 1.]]
```