```
!git clone https://github.com/ClawSwipe/lrp_toolbox.git
```

```
fatal: destination path 'lrp_toolbox' already exists and is not an empty directory.
```

```
cd lrp_toolbox/python
```

```
/content/lrp_toolbox/python
```

Model IO

[ ] ↳ 1 cell hidden

Data IO

[ ] ↳ 1 cell hidden

Render

[ ] ↳ 1 cell hidden

Training Test MNIST

[ ] ↳ 1 cell hidden

LRP Demo MNIST

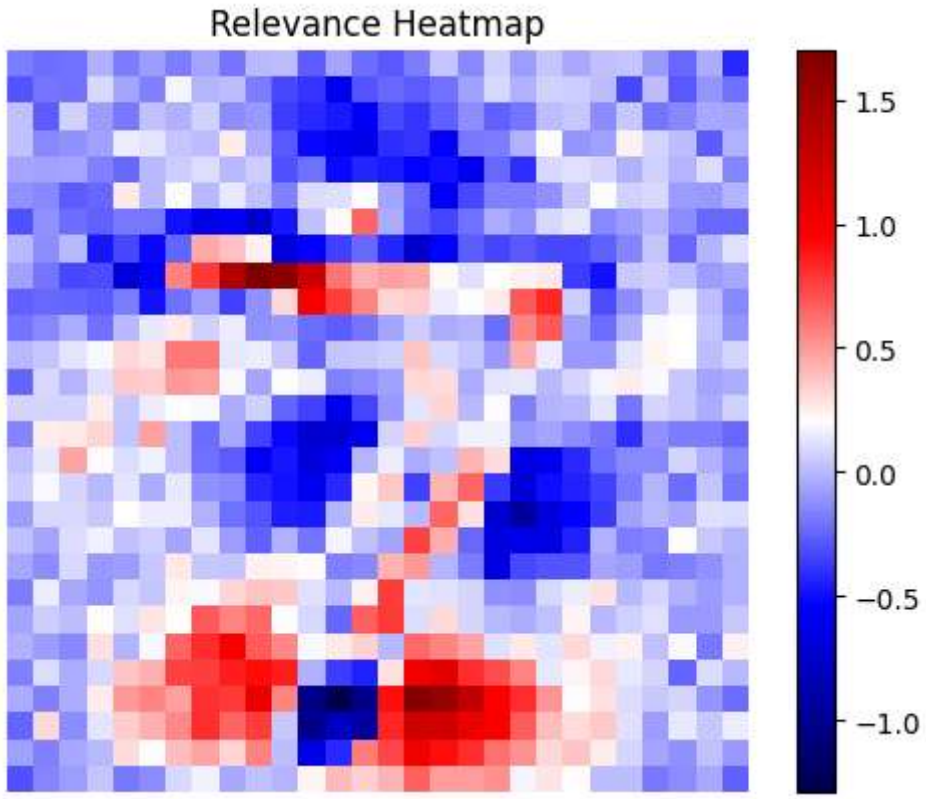
```
'''
@author: Sebastian Lapuschkin
@maintainer: Sebastian Lapuschkin
@contact: sebastian.lapuschkin@hhi.fraunhofer.de, wojciech.samek@hhi.fraunhofer.de
@date: 14.08.2015
@version: 1.0
@copyright: Copyright (c)  2015-2017, Sebastian Lapuschkin, Alexander Binder, Gregoire Mor
@license : BSD-2-Clause

The purpose of this module is to demonstrate the process of obtaining pixel-wise explanati

The module first loads a pre-trained neural network model and the MNIST test set with labe
The data is then randomly permuted and for the first 10 samples due to the permuted order,
```

```
by attributing relevance values to each of the input pixels.

finally, the resulting heatmap is rendered as an image and (over)written out to disk and c

LRP works by:

    Doing a standard forward pass on an input sample to make a prediction.

    Starting with the output prediction as total relevance, then propagating that relevanc

    At each layer, relevance is redistributed to the layer below based on specific rules (

    Finally, each input feature gets a relevance score that tells you how much it contribu
'''



import matplotlib.pyplot as plt
import time
import numpy
import numpy as np
import importlib.util as imp
if imp.find_spec("cupy"): #use cupy for GPU support if available
    import cupy
    import cupy as np
na = np.newaxis
import render
from skimage.transform import resize


#load a neural network, as well as the MNIST test data and some labels
nn = model_io.read('../models/MNIST/long-tanh.nn') # 99.16% prediction accuracy
nn.drop_softmax_output_layer() #drop softnax output layer for analyses

X = data_io.read('../data/MNIST/test_images.npy')
Y = data_io.read('../data/MNIST/test_labels.npy')

# transfer pixel values from [0 255] to [-1 1] to satisfy the expected input / training pa
X =  X / 127.5 - 1

# transform numeric class labels to vector indicator for uniformity. assume presence of al
I = Y[:,0].astype(int)
Y = np.zeros([X.shape[0],np.unique(Y).size])
Y[np.arange(Y.shape[0]),I] = 1

acc = np.mean(np.argmax(nn.forward(X), axis=1) == np.argmax(Y, axis=1))
if not np == numpy: # np=cupy
    acc = np.asnumpy(acc)
print('model test accuracy is: {:0.4f}'.format(acc))

#permute data order for demonstration. or not. your choice.
I = np.arange(X.shape[0])
```

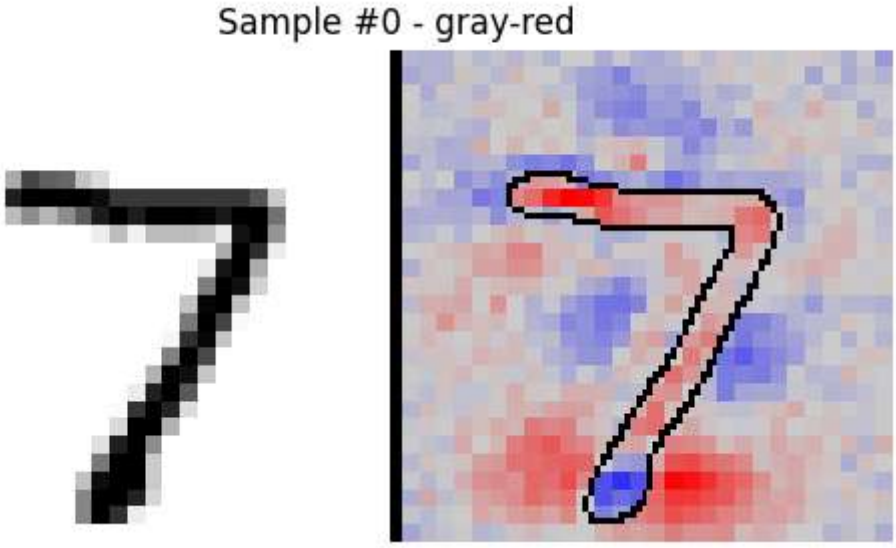
```
#I = np.random.permutation(I)


#predict and perform LRP for the 10 first samples
for i in I[:10]:
    x = X[na,i,:]

    #forward pass and prediction
    ypred = nn.forward(x)
    print('True Class:     ', np.argmax(Y[i]))
    print('Predicted Class:', np.argmax(ypred),'\n')

    #prepare initial relevance to reflect the model's dominant prediction (ie depopulate r
    mask = np.zeros_like(ypred)
    mask[:,np.argmax(ypred)] = 1
    Rinit = ypred*mask

    #compute first layer relevance according to prediction
    #R = nn.lrp(Rinit)                   #as Eq(56) from DOI: 10.1371/journal.pone.013014(
    R = nn.lrp(Rinit,'epsilon',0.01)    #as Eq(58) from DOI: 10.1371/journal.pone.0130140
    #R = nn.lrp(Rinit,'alphabeta',2)    #as Eq(60) from DOI: 10.1371/journal.pone.0130140

    #R = nn.lrp(ypred*Y[na,i]) #compute first layer relevance according to the true class
    '''
    yselect = 3
    yselect = (np.arange(Y.shape[1])[na,:] == yselect)*1.
    R = nn.lrp(ypred*yselect) #compute first layer relvance for an arbitrarily selected cl
    '''

    #undo input normalization for digit drawing. get it back to range [0,1] per pixel
    x = (x+1.)/2.

    if not np == numpy: # np=cupy
        x = np.asnumpy(x)
        R = np.asnumpy(R)

    #render input and heatmap as rgb images
    digit = render.digit_to_rgb(x, scaling=3)


    # Similarly, reshape the relevance map if it's flattened (1D array)
    R_numpy = R.reshape(28, 28)

    # Display the relevance map as a heatmap
    plt.imshow(R_numpy, cmap='seismic', interpolation='none')
    plt.colorbar()  # Show color bar to indicate relevance magnitude
    plt.title("Relevance Heatmap")
    plt.axis('off')
    plt.show()

    # Loop through all custom heatmaps
```

```
        for cmap_name in render.custom_maps.keys():
            print(f'Applying custom heatmap: {cmap_name}')
            hm = render.hm_to_rgb(R, X=x, scaling=3, cmap=cmap_name, sigma=2)
            digit_hm = render.save_image([digit, hm], f'../heatmap_{i}_{cmap_name}.png')

            # Display (optional)
            plt.imshow(digit_hm, interpolation='none')
            plt.title(f'Sample #{i} - {cmap_name}')
            plt.axis('off')
            plt.show()

        digit_hm = render.save_image([digit, hm], f'../heatmap_{i}.png')
        data_io.write(R,'../heatmap.npy')

        #display the image as written to file
        plt.imshow(digit_hm, interpolation = 'none')
        plt.axis('off')
        plt.show()


    #note that modules.Sequential allows for batch processing inputs
    if True:
        N = 256
        t_start = time.time()
        x = X[:N,...]
        y = nn.forward(x)
        R = nn.lrp(y)
        data_io.write(R,'../Rbatch.npy')
        print('Computation of {} heatmaps using {} in {:.3f}s'.format(N, np.__name__, time.tim
```

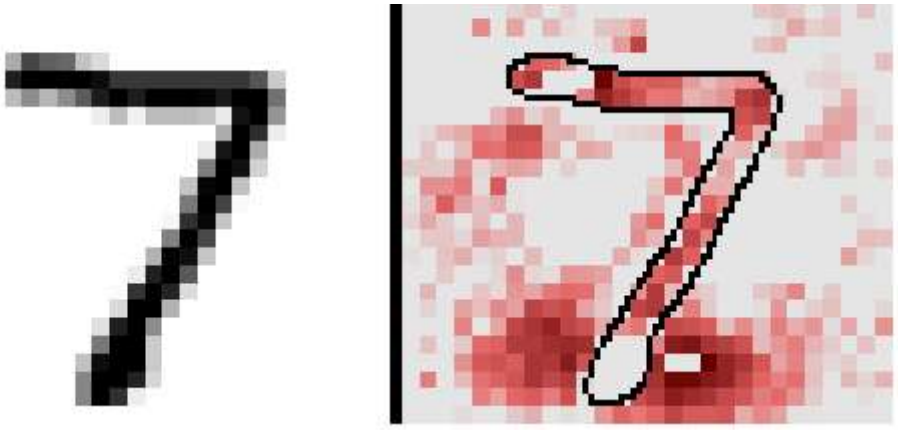
```
loading pickled model from ../models/MNIST/long-tanh.nn
removing softmax output mapping
loading np-formatted data from ../data/MNIST/test_images.npy
loading np-formatted data from ../data/MNIST/test_labels.npy
model test accuracy is: 0.9916
True Class:      7
Predicted Class: 7
```




```
Applying custom heatmap: gray-red
saving image to  ../heatmap_0_gray-red.png
```

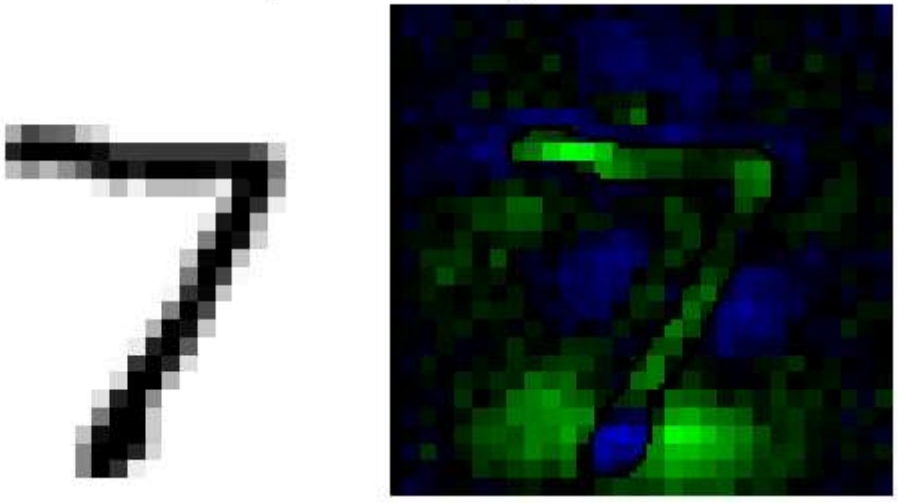



```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_0_gray-red2.png
```



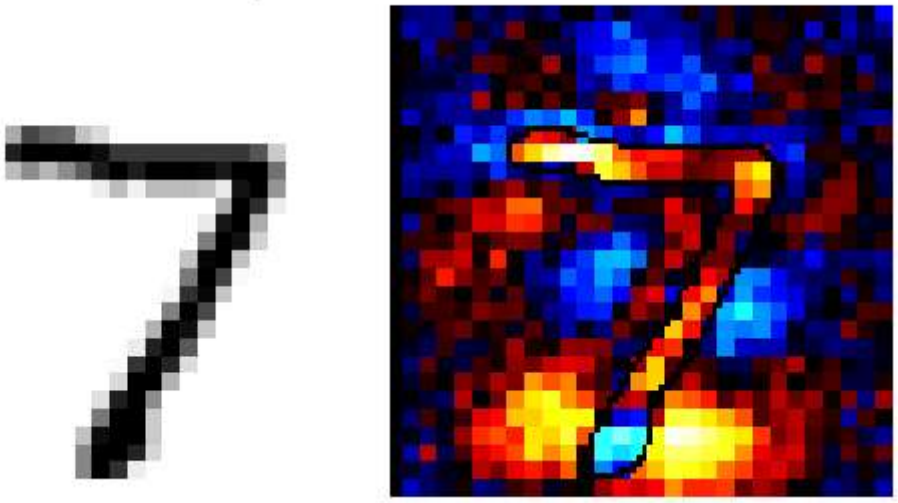
```
Applying custom heatmap: black-green
saving image to  ../heatmap_0_black-green.png
```

Sample #0 - black-green



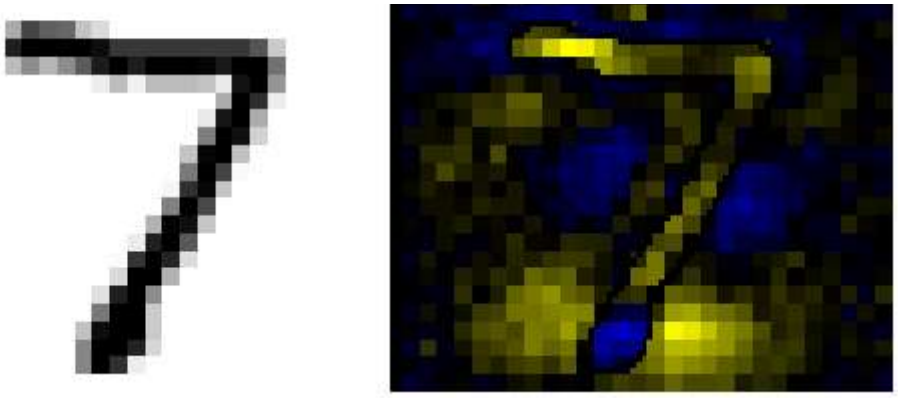
```
Applying custom heatmap: black-firered
saving image to  ../heatmap_0_black-firered.png
```

Sample #0 - black-firered

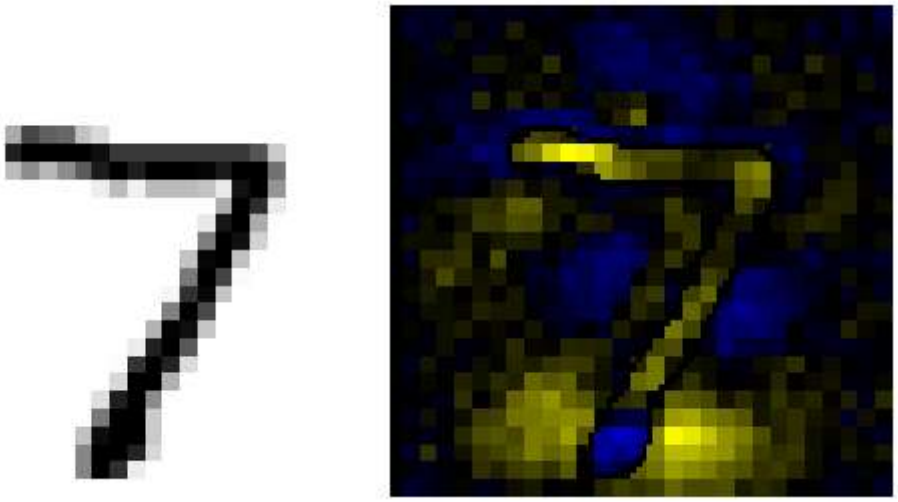


```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_0_blue-black-yellow.png
```

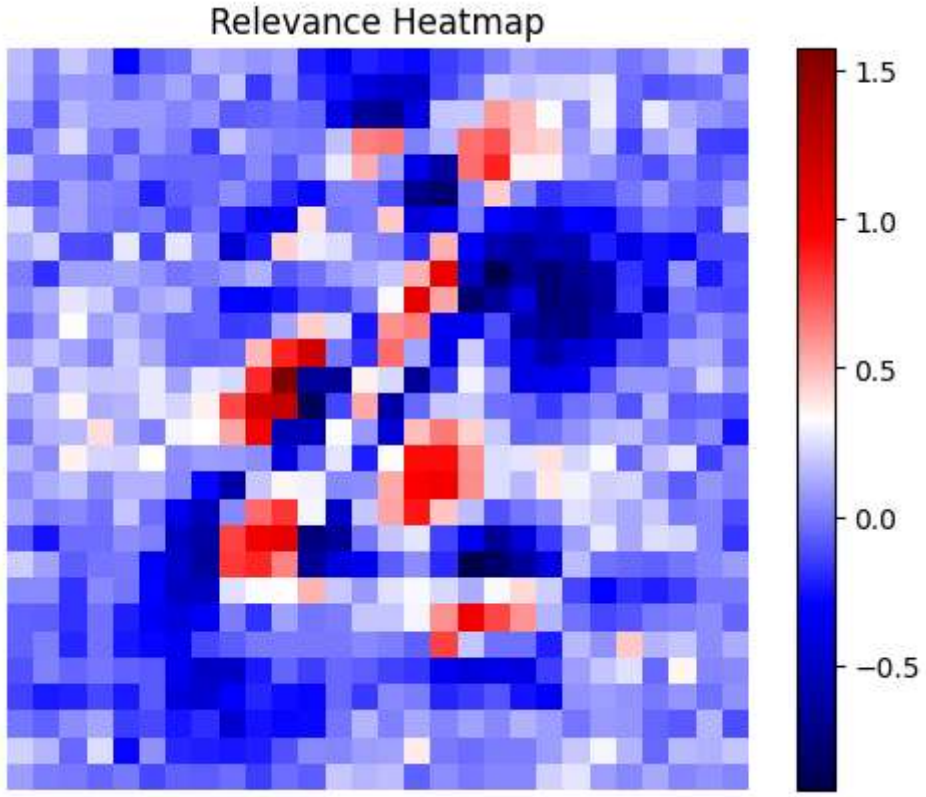
Sample #0 - blue-black-yellow



```
saving image to  ../heatmap_0.png
writing data in npy-format to ../heatmap.npy
```

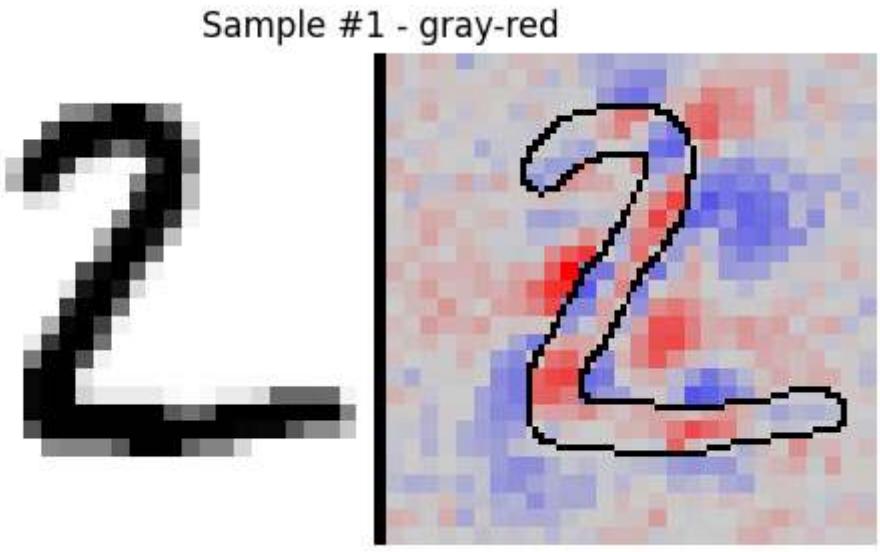


```
True Class:      2
Predicted Class: 2
```

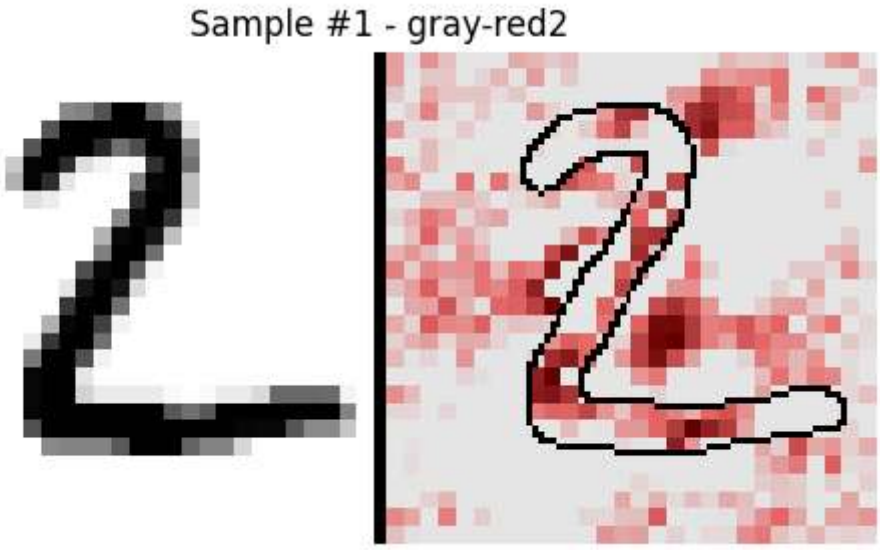



```
Applying custom heatmap: gray-red
```

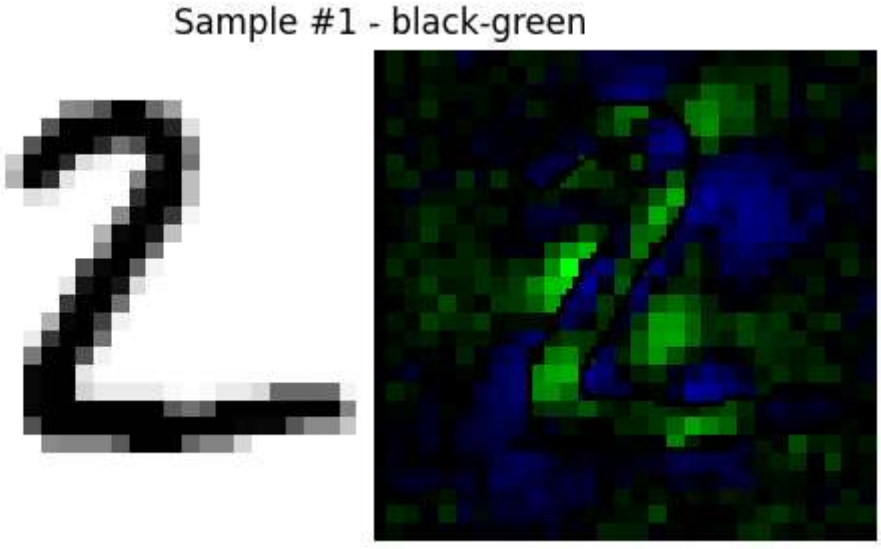
```
saving image to  ../heatmap_1_gray-red.png
```




```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_1_gray-red2.png
```

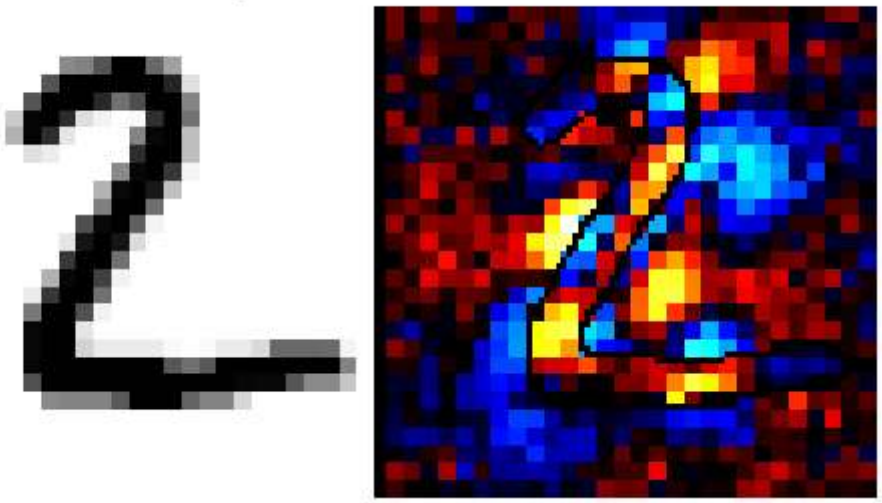



```
Applying custom heatmap: black-green
saving image to  ../heatmap_1_black-green.png
```



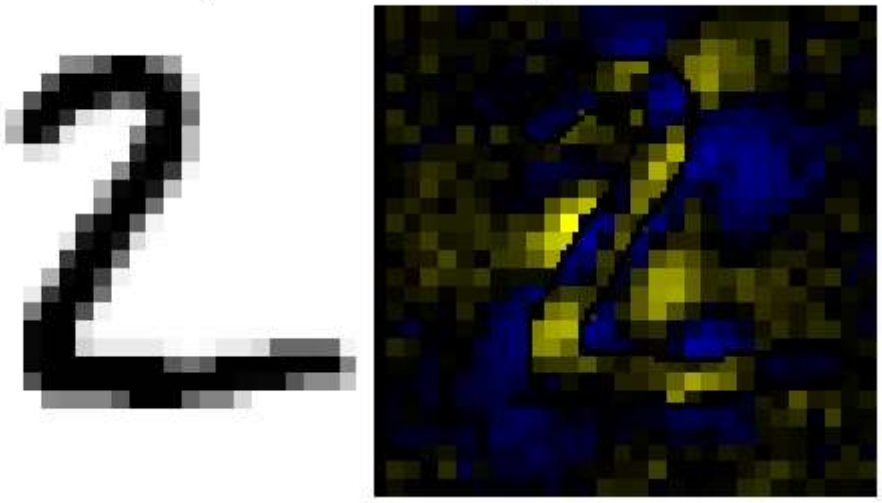

```
Applying custom heatmap: black-firered
saving image to  ../heatmap_1_black-firered.png
```

Sample #1 - black-firered

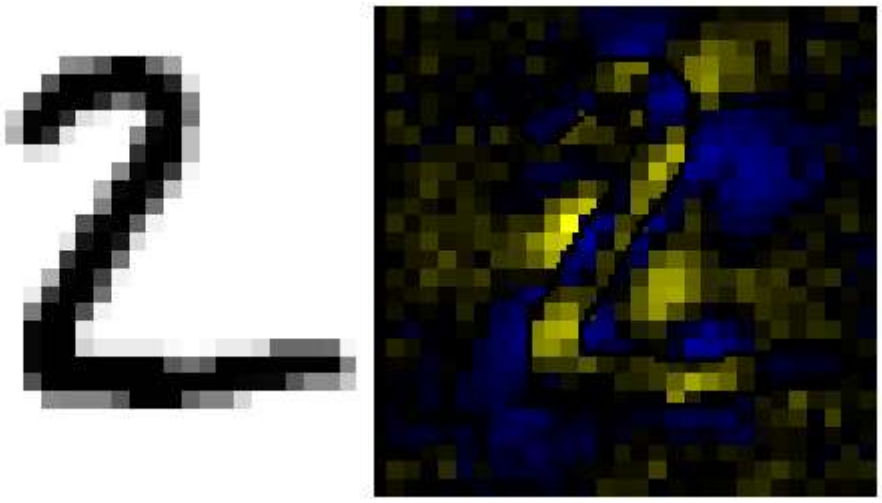


```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_1_blue-black-yellow.png
```

Sample #1 - blue-black-yellow

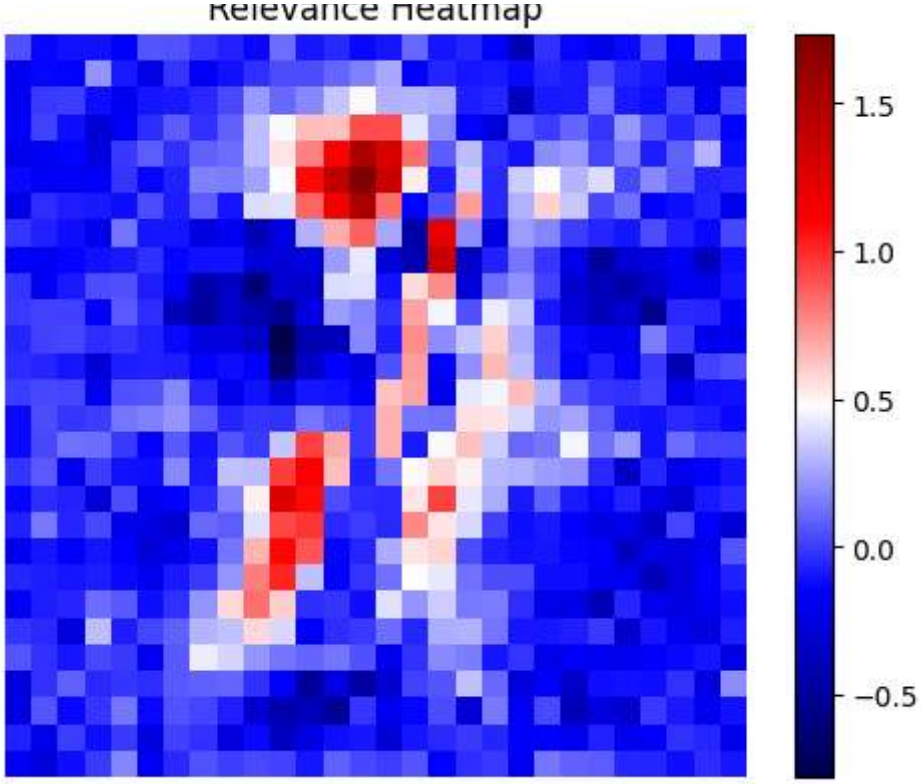


```
saving image to  ../heatmap_1.png
writing data in npy-format to ../heatmap.npy
```

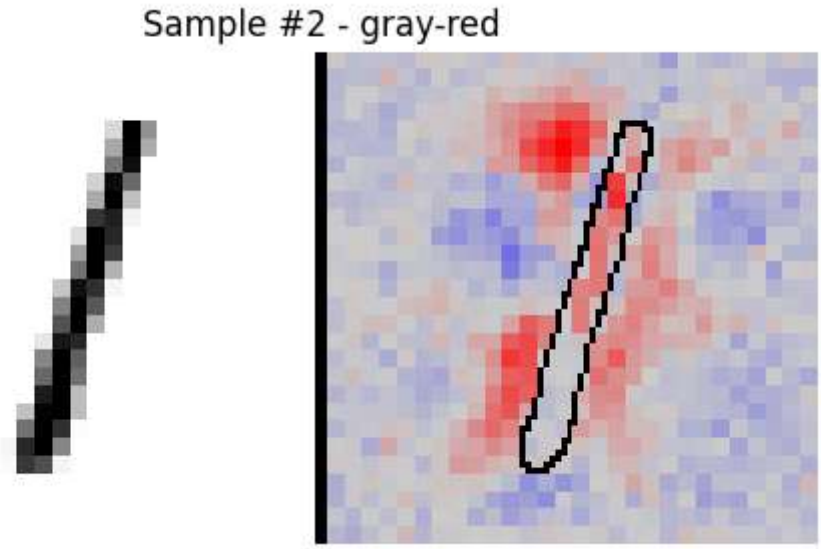


```
True Class:      1
Predicted Class: 1
```

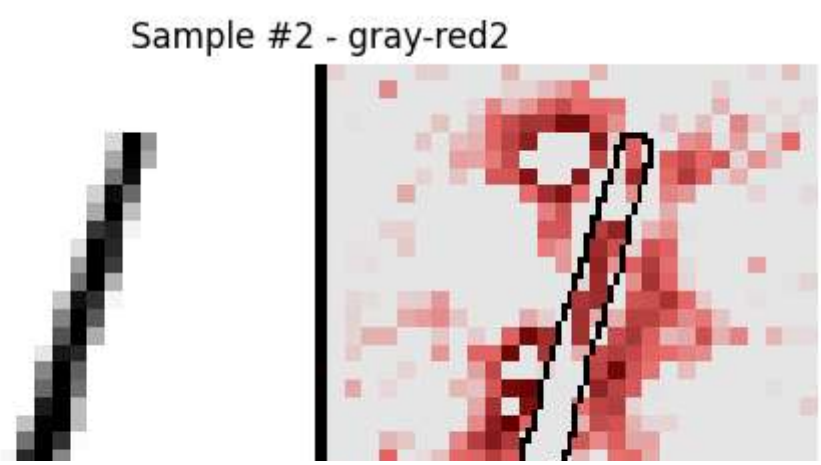
Relevance Heatmap




```
Applying custom heatmap: gray-red
saving image to  ../heatmap_2_gray-red.png
```

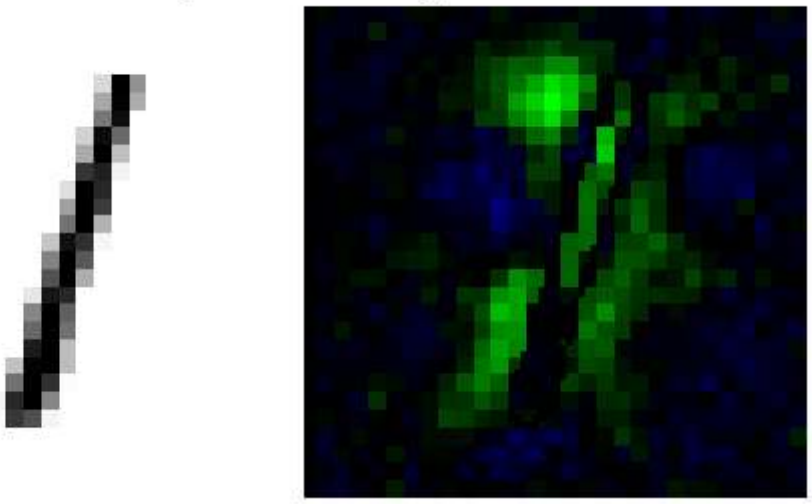



```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_2_gray-red2.png
```



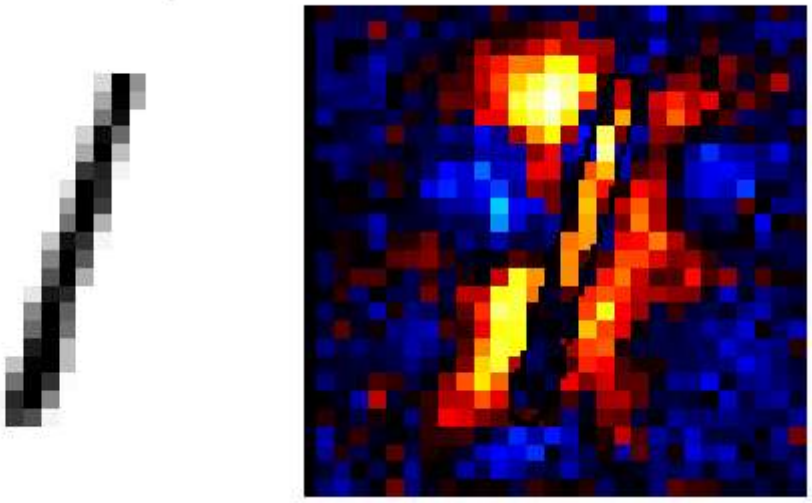
```
Applying custom heatmap: black-green
saving image to  ../heatmap_2_black-green.png
```

Sample #2 - black-green



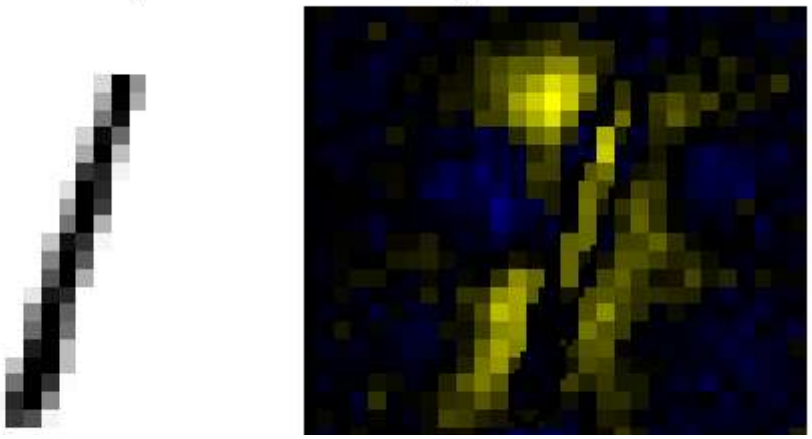
```
Applying custom heatmap: black-firered
saving image to  ../heatmap_2_black-firered.png
```

Sample #2 - black-firered

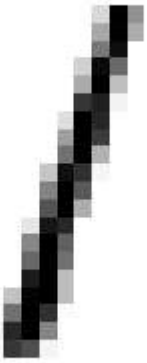


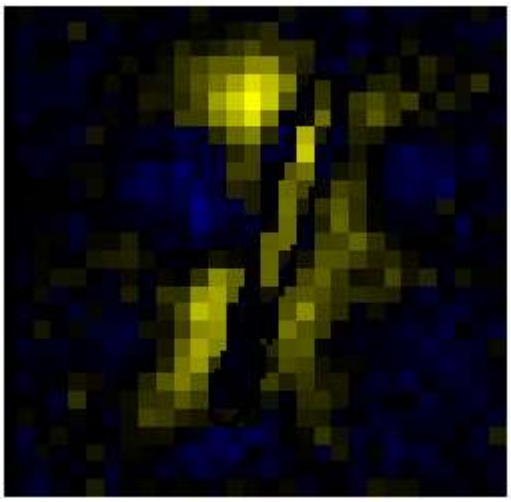
```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_2_blue-black-yellow.png
```

Sample #2 - blue-black-yellow

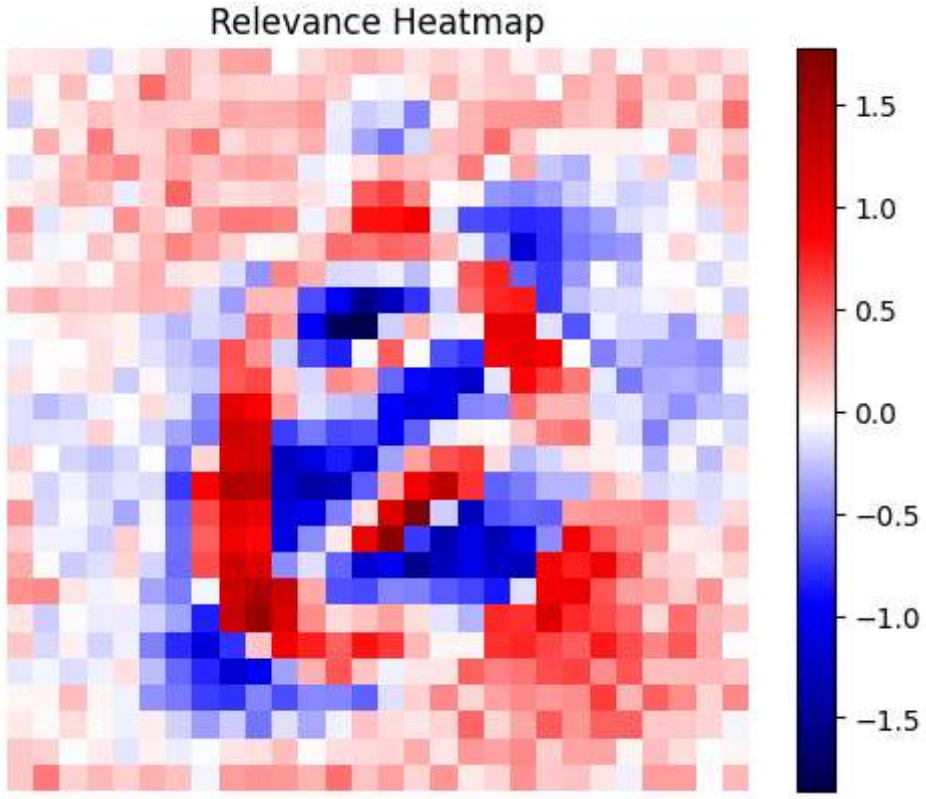


```
saving image to  ../heatmap_2.png
writing data in npy-format to ../heatmap.npy
```

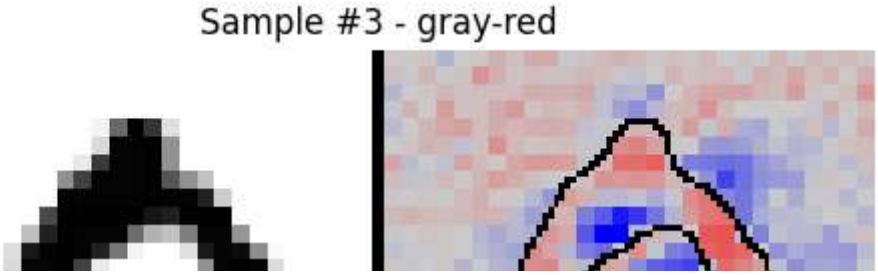


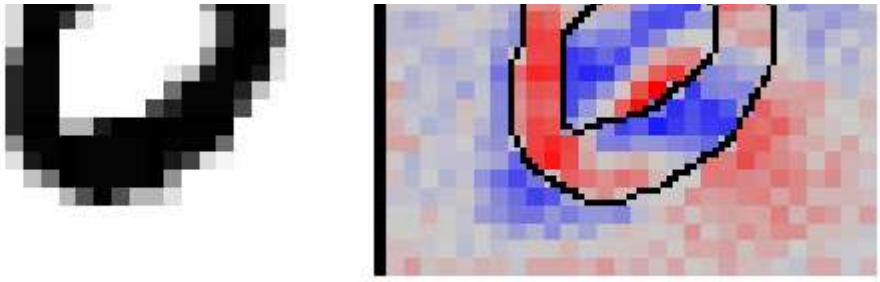


```
True Class:      0
Predicted Class: 0
```



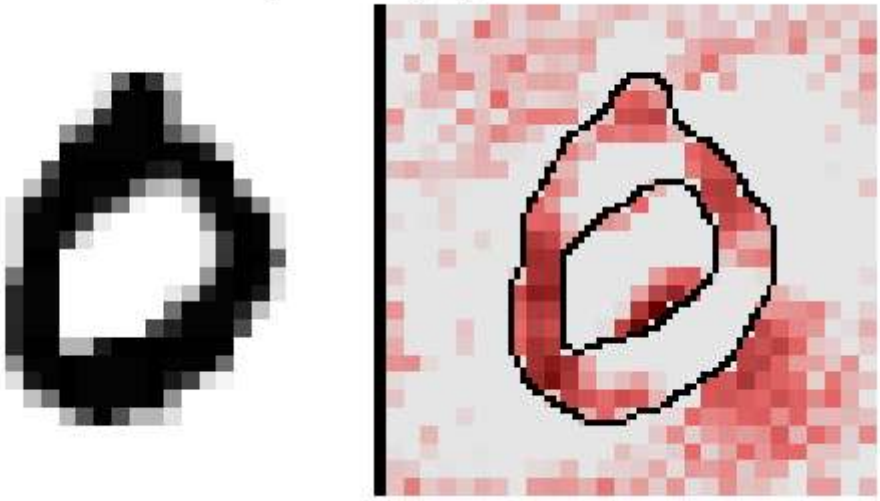

```
Applying custom heatmap: gray-red
saving image to  ../heatmap_3_gray-red.png
```





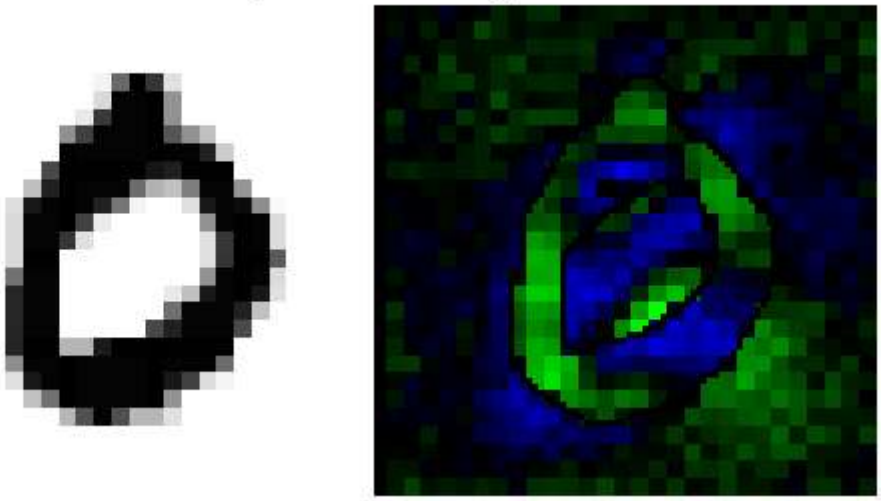
```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_3_gray-red2.png
```

Sample #3 - gray-red2



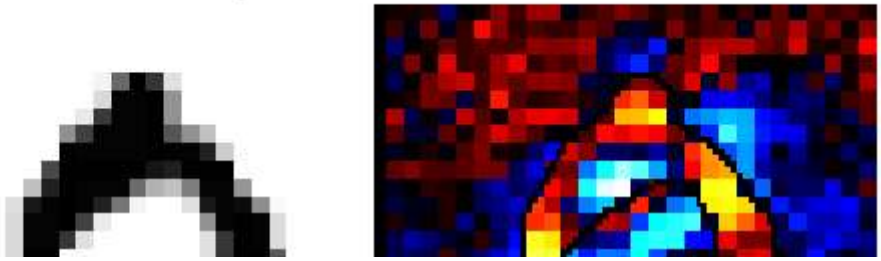
```
Applying custom heatmap: black-green
saving image to  ../heatmap_3_black-green.png
```

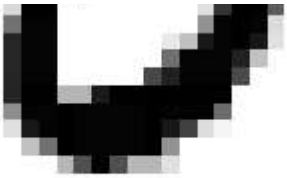
Sample #3 - black-green

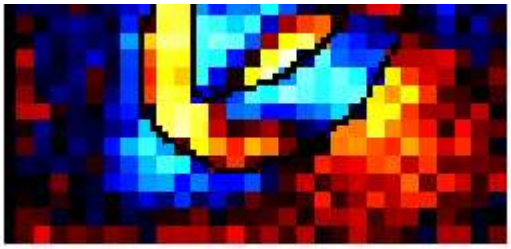


```
Applying custom heatmap: black-firered
saving image to  ../heatmap_3_black-firered.png
```

Sample #3 - black-firered

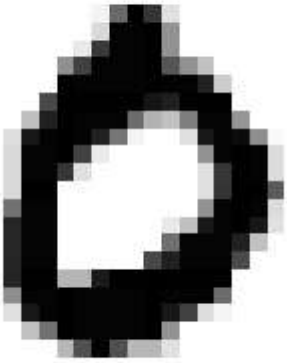


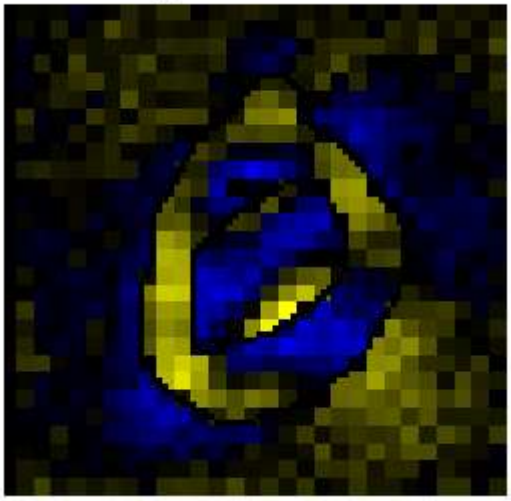




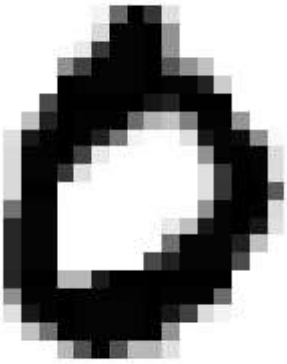
```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_3_blue-black-yellow.png
```

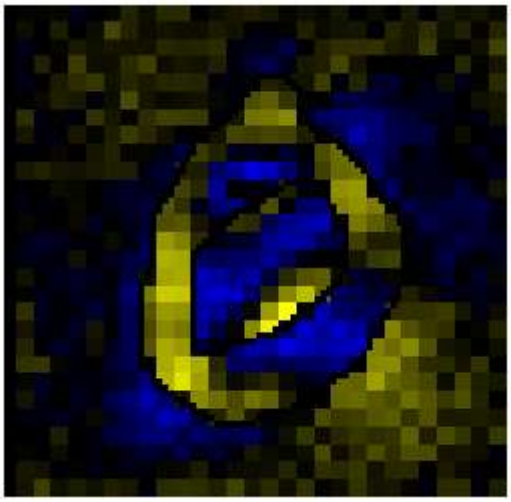
Sample #3 - blue-black-yellow





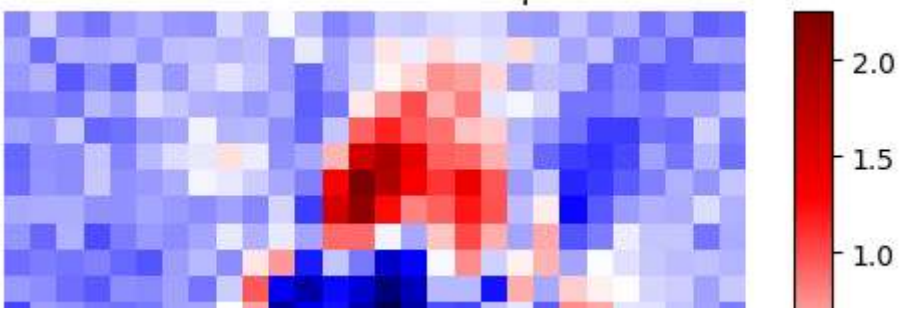
```
saving image to  ../heatmap_3.png
writing data in npy-format to ../heatmap.npy
```

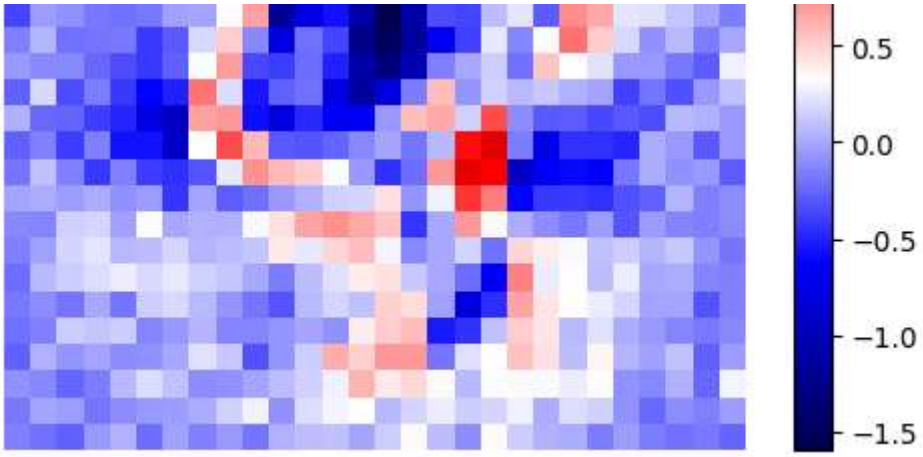




```
True Class:      4
Predicted Class: 4
```

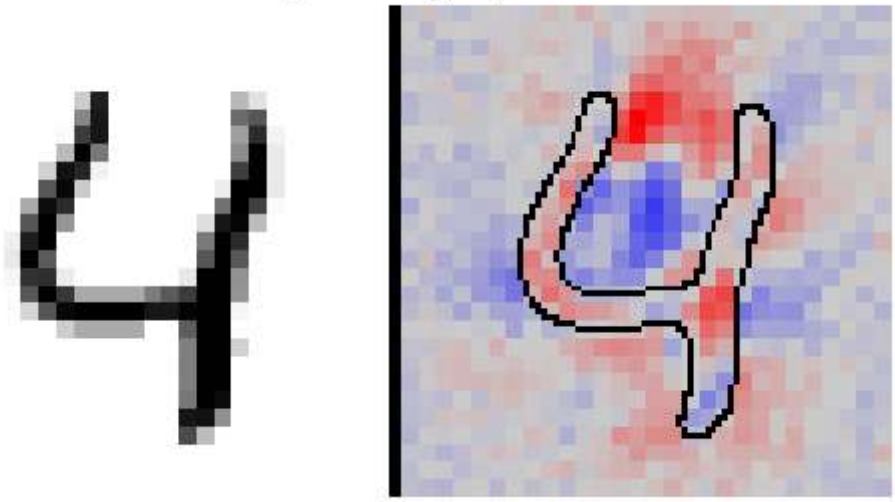
Relevance Heatmap





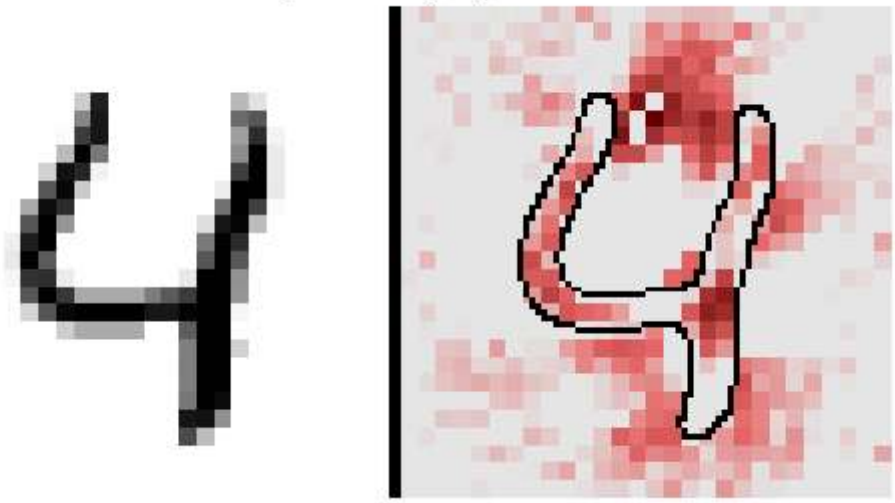

```
Applying custom heatmap: gray-red
saving image to  ../heatmap_4_gray-red.png
```

Sample #4 - gray-red



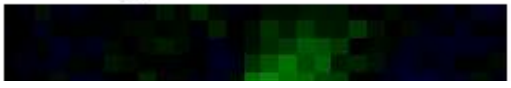
```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_4_gray-red2.png
```

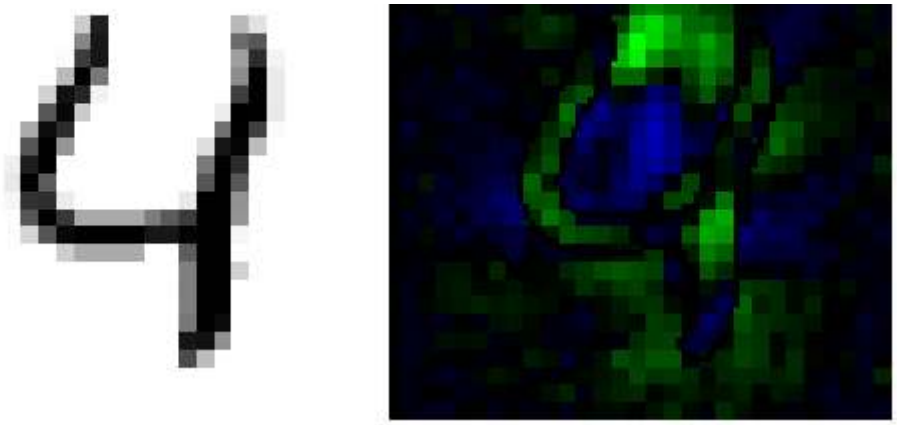
Sample #4 - gray-red2



```
Applying custom heatmap: black-green
saving image to  ../heatmap_4_black-green.png
```

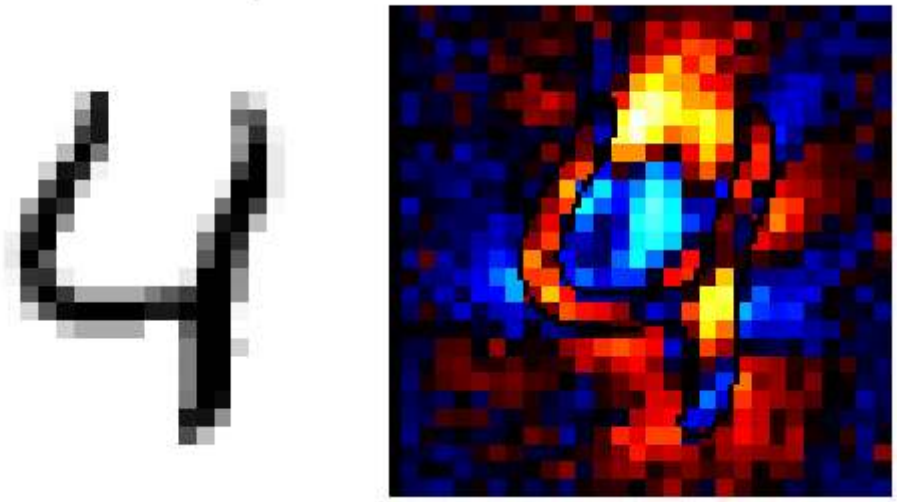
Sample #4 - black-green





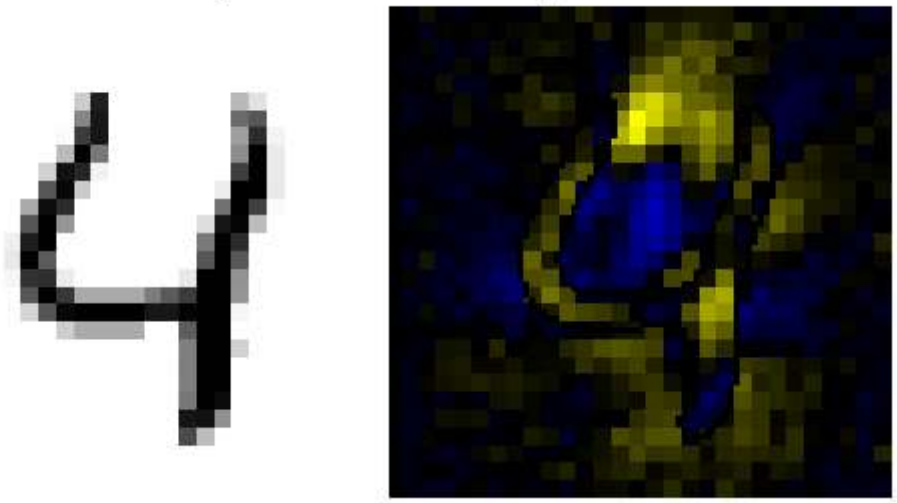
```
Applying custom heatmap: black-firered
saving image to  ../heatmap_4_black-firered.png
```

Sample #4 - black-firered



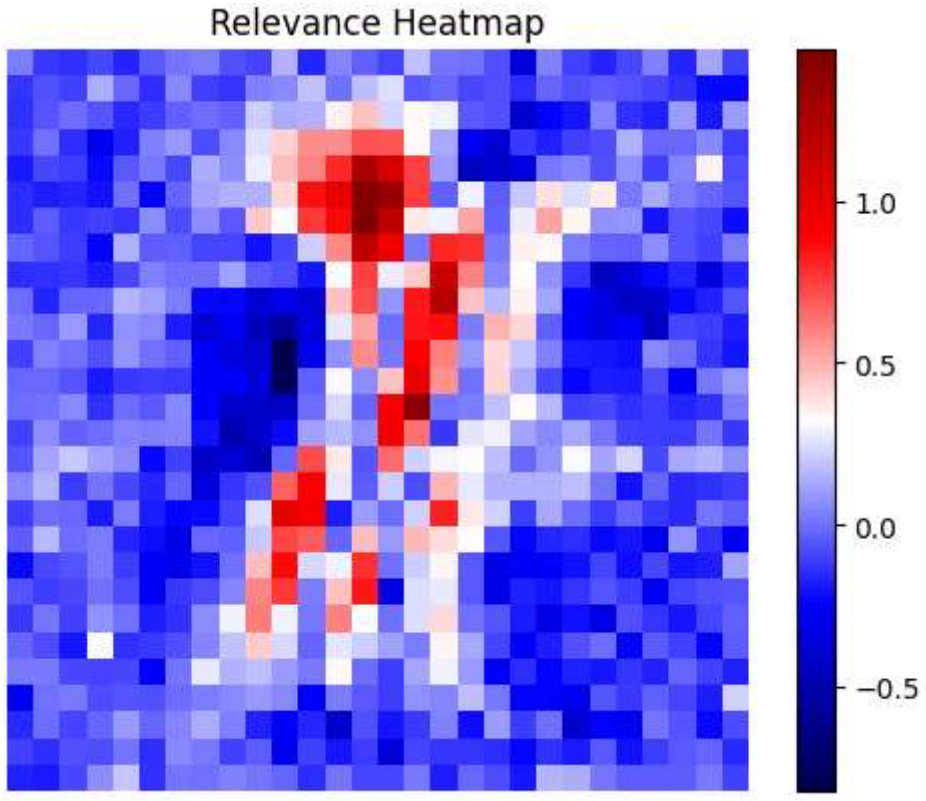
```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_4_blue-black-yellow.png
```

Sample #4 - blue-black-yellow

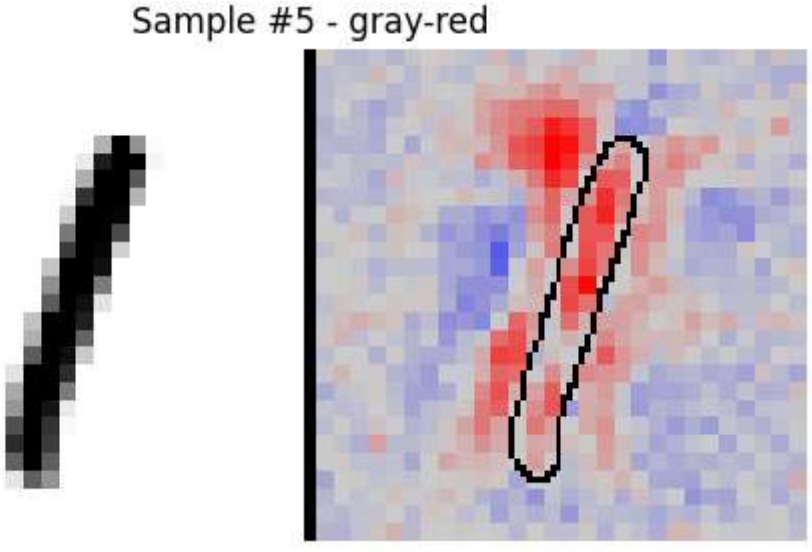


```
saving image to  ../heatmap_4.png
writing data in npy-format to ../heatmap.npy
```

```
True Class:      1
Predicted Class: 1
```

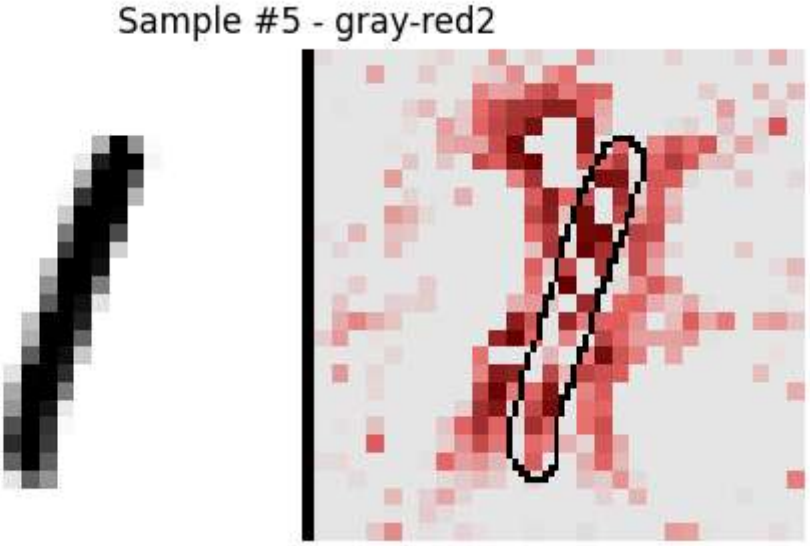



```
Applying custom heatmap: gray-red
saving image to  ../heatmap_5_gray-red.png
```

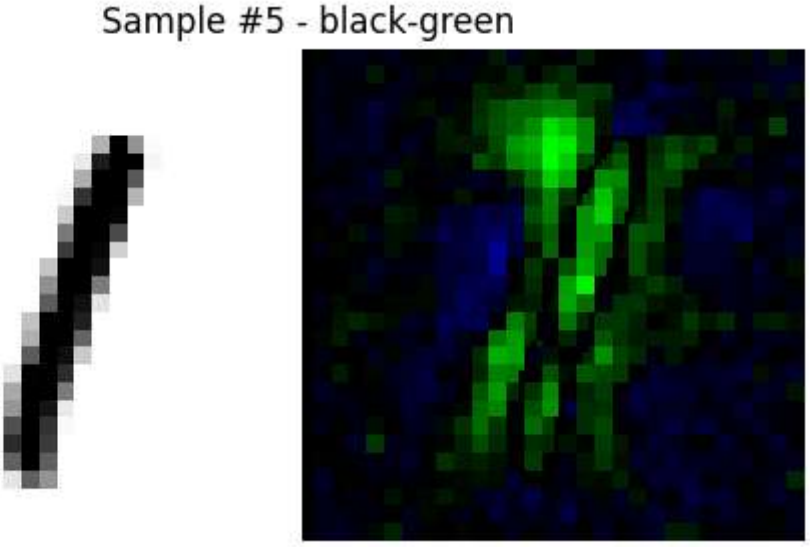



```
Applying custom heatmap: gray-red2
```

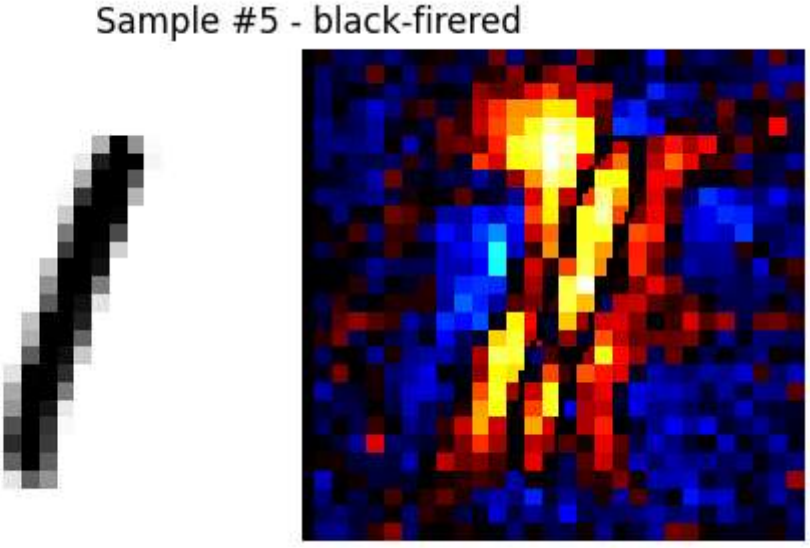
```
saving image to  ../heatmap_5_gray-red2.png
```



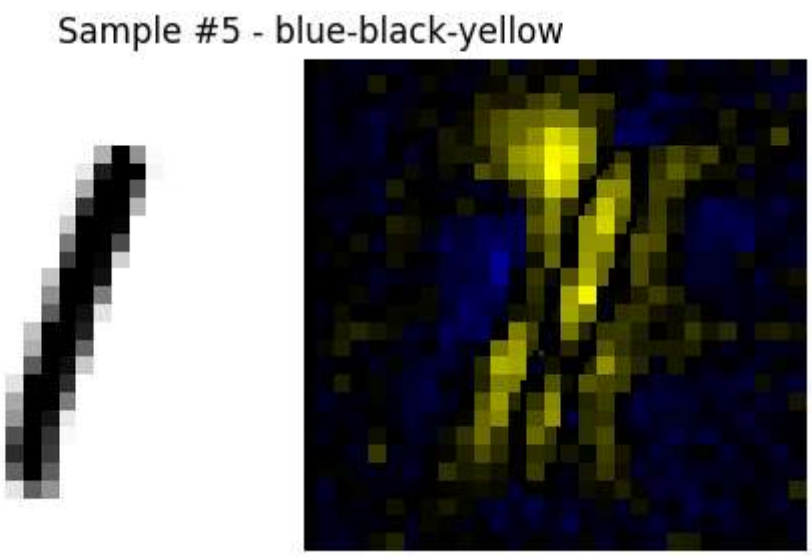

```
Applying custom heatmap: black-green
saving image to  ../heatmap_5_black-green.png
```



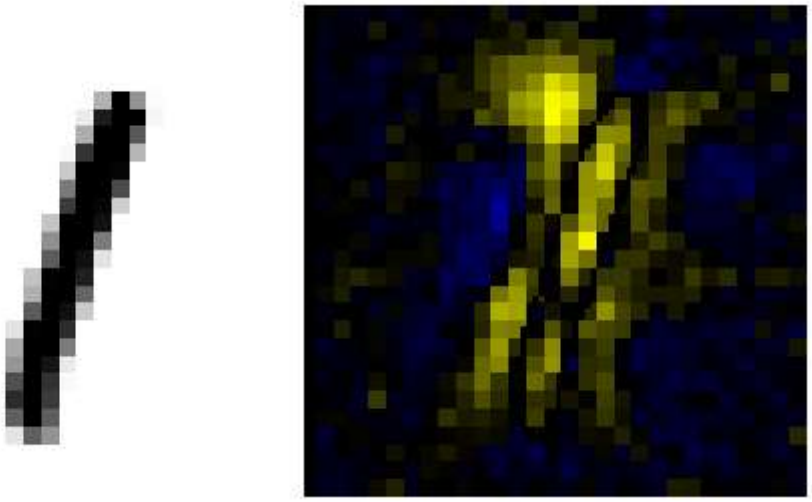

```
Applying custom heatmap: black-firered
saving image to  ../heatmap_5_black-firered.png
```



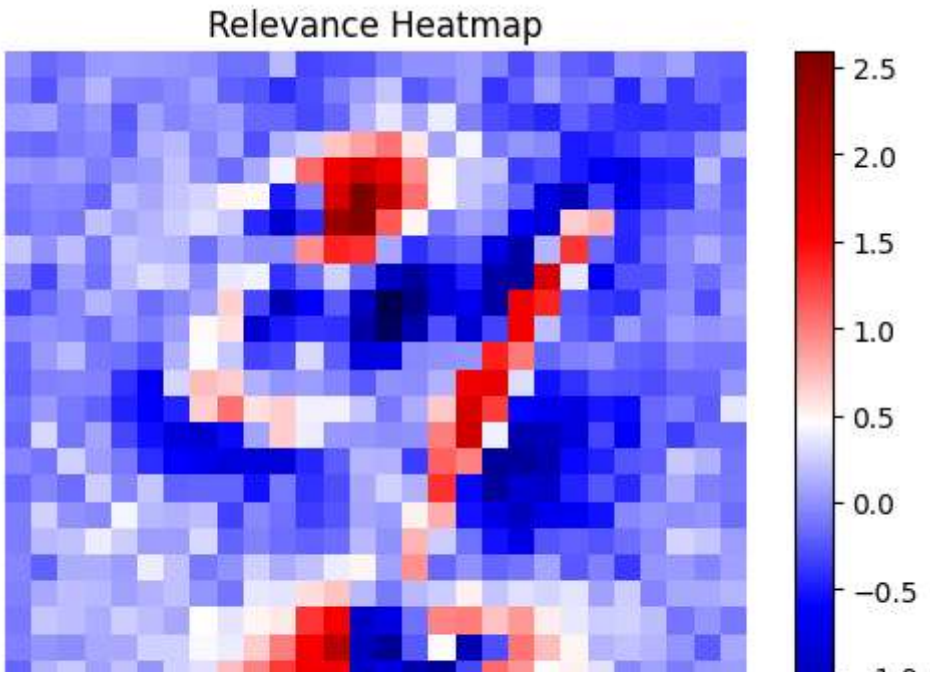

```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_5_blue-black-yellow.png
```

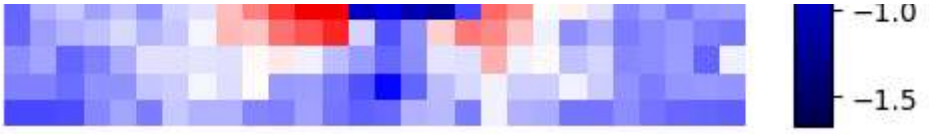



```
saving image to  ../heatmap_5.png
writing data in npy-format to ../heatmap.npy
```



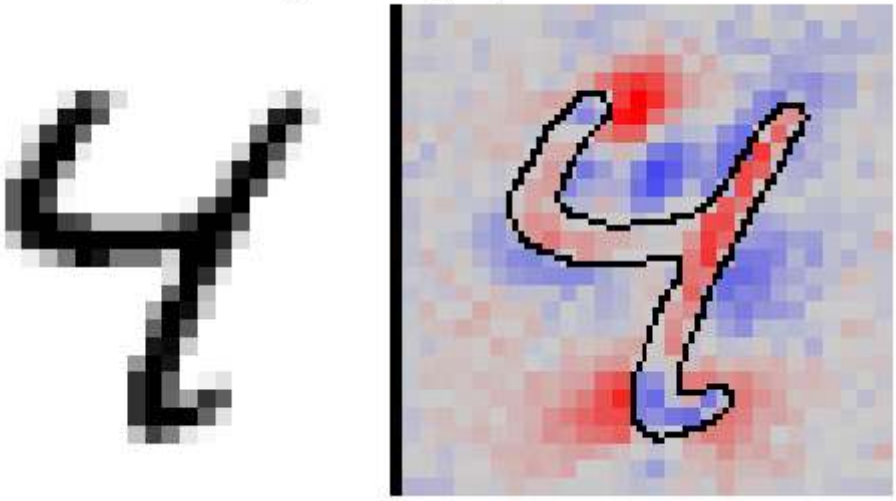
```
True Class:      4
Predicted Class: 4
```





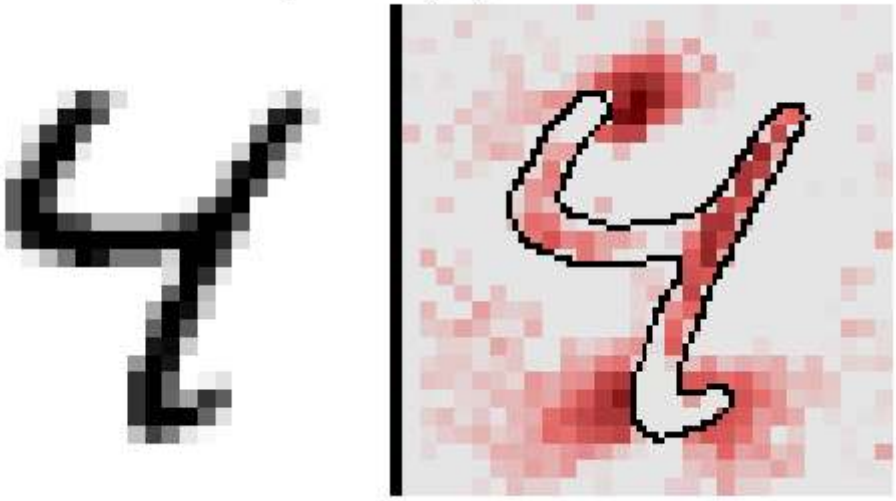

```
Applying custom heatmap: gray-red
saving image to  ../heatmap_6_gray-red.png
```

Sample #6 - gray-red



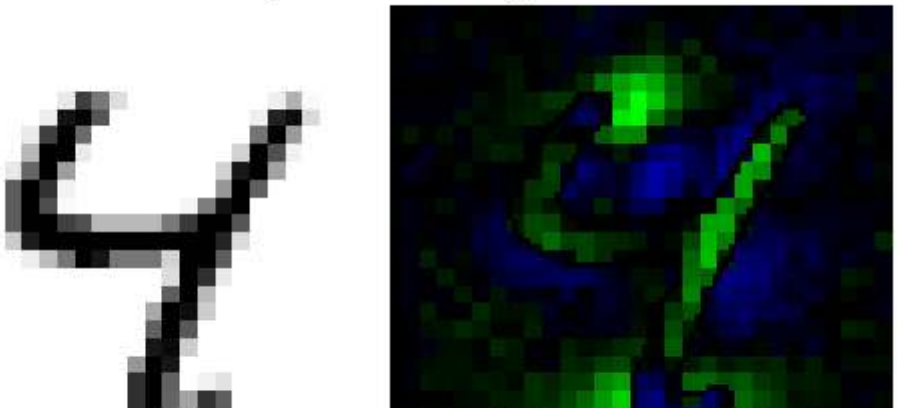
```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_6_gray-red2.png
```

Sample #6 - gray-red2

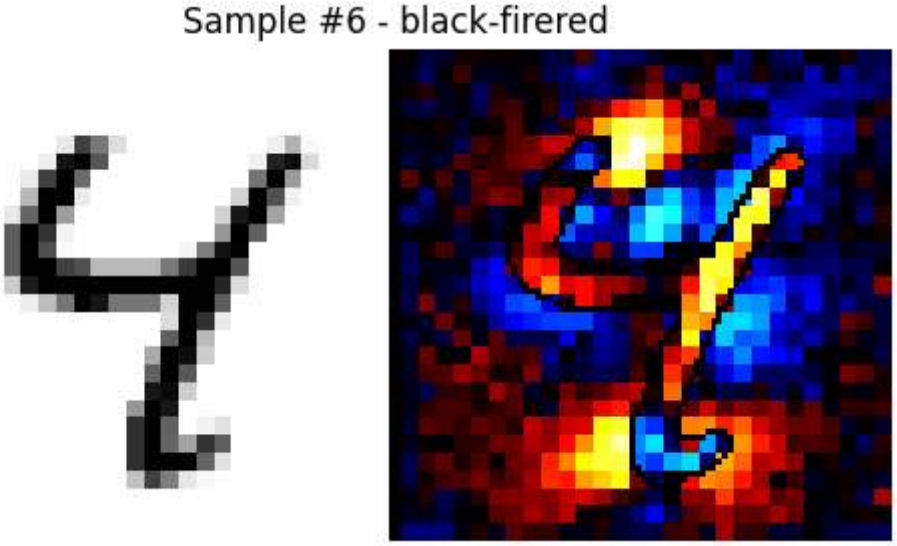


```
Applying custom heatmap: black-green
saving image to  ../heatmap_6_black-green.png
```

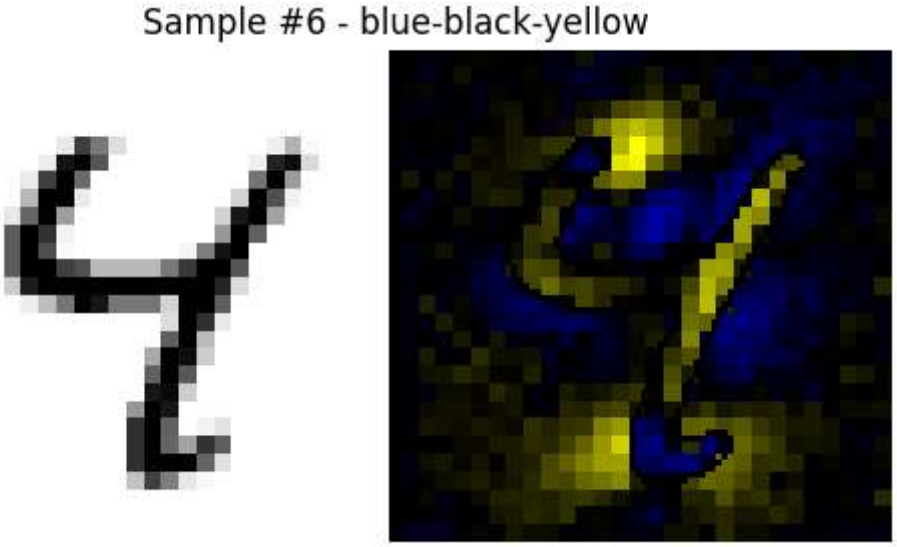
Sample #6 - black-green



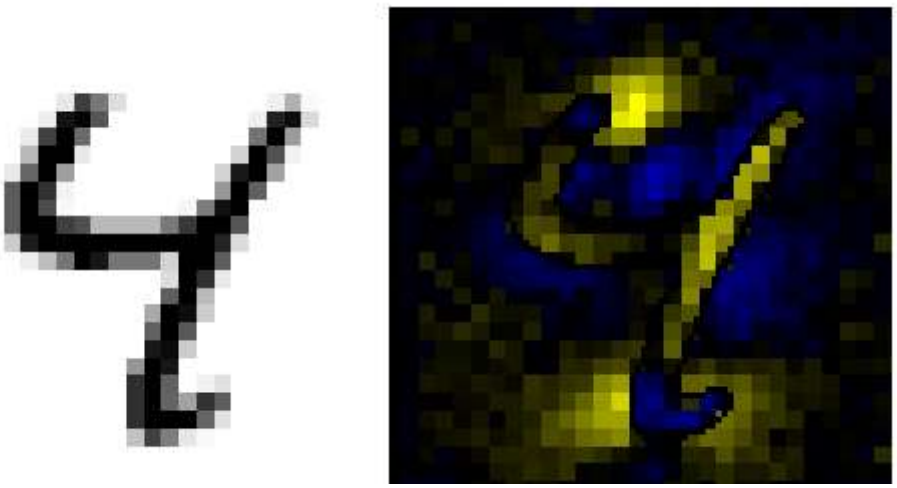
```
Applying custom heatmap: black-firered
saving image to  ../heatmap_6_black-firered.png
```



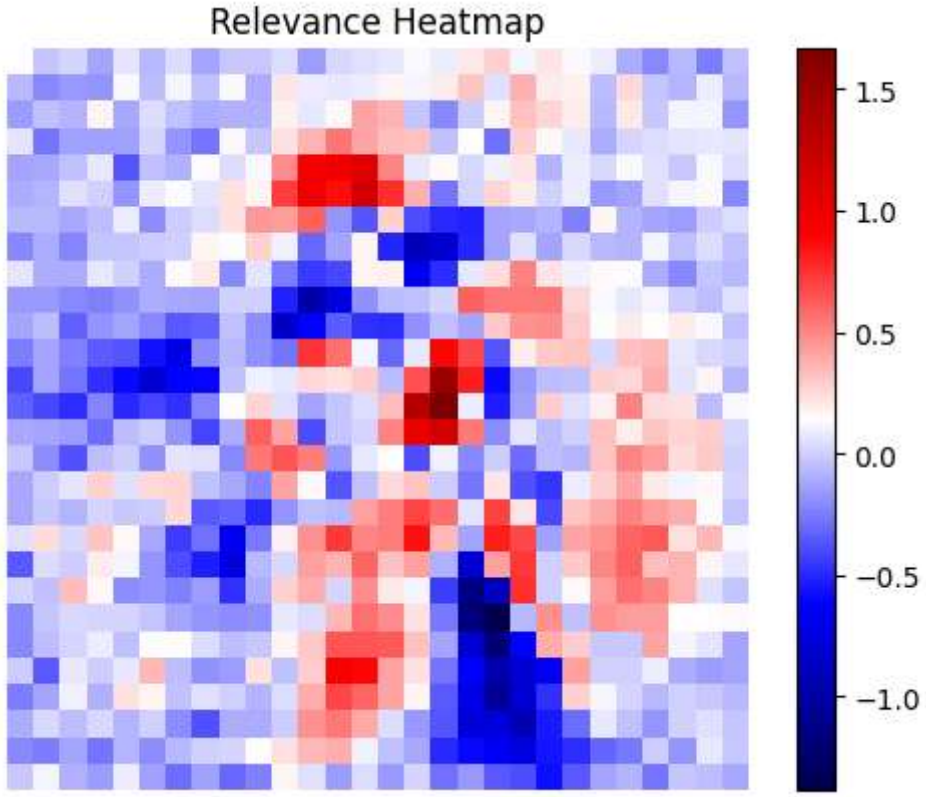

```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_6_blue-black-yellow.png
```



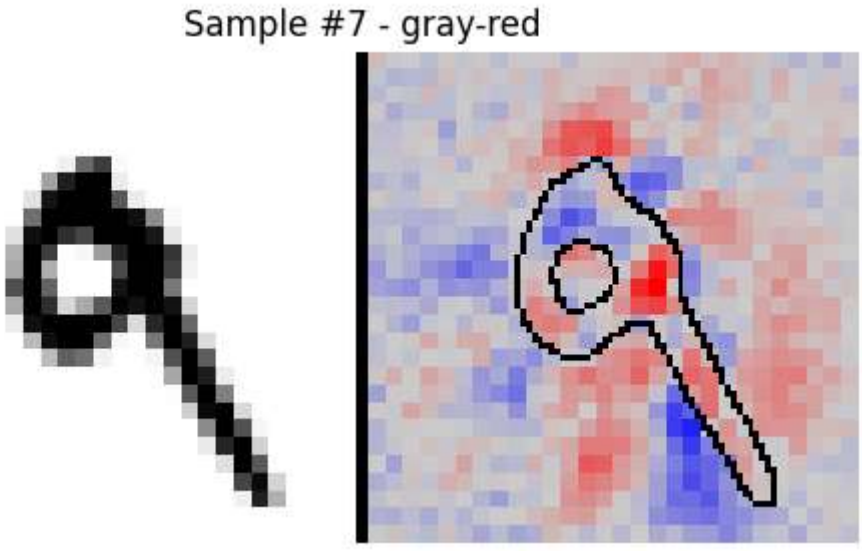

```
saving image to  ../heatmap_6.png
writing data in npy-format to ../heatmap.npy
```



```
True Class:      9
Predicted Class: 9
```

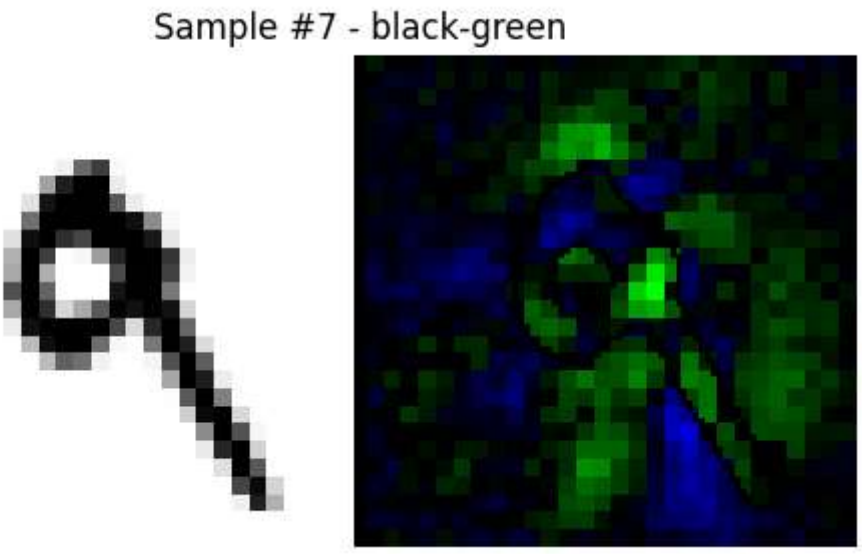



```
Applying custom heatmap: gray-red
saving image to  ../heatmap_7_gray-red.png
```

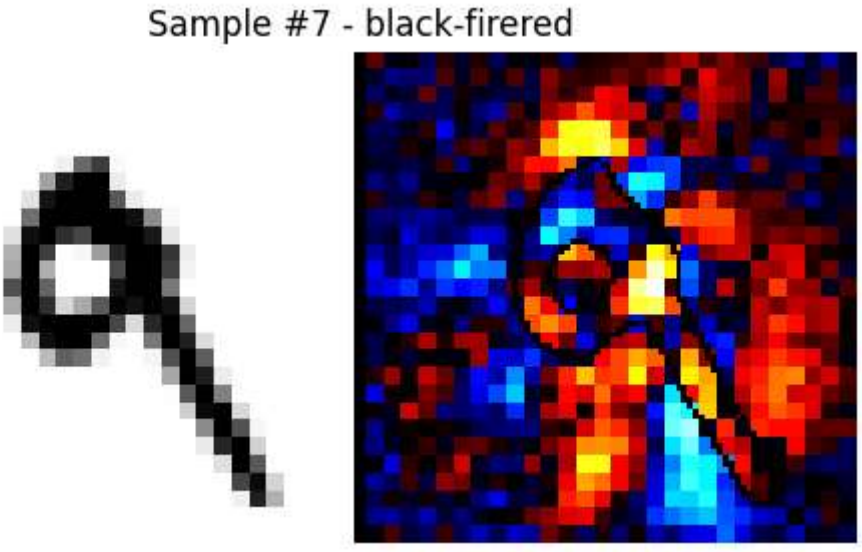



```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_7_gray-red2.png
```

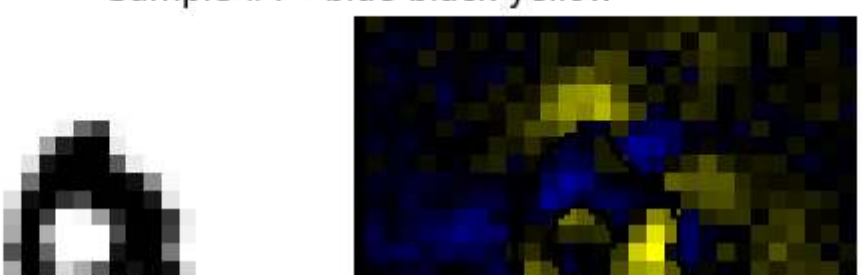
```
Applying custom heatmap: black-green
saving image to  ../heatmap_7_black-green.png
```

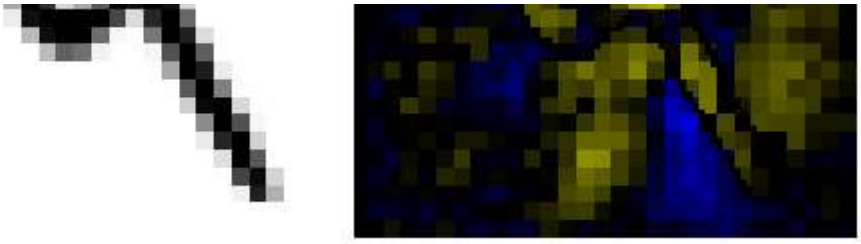



```
Applying custom heatmap: black-firered
saving image to  ../heatmap_7_black-firered.png
```

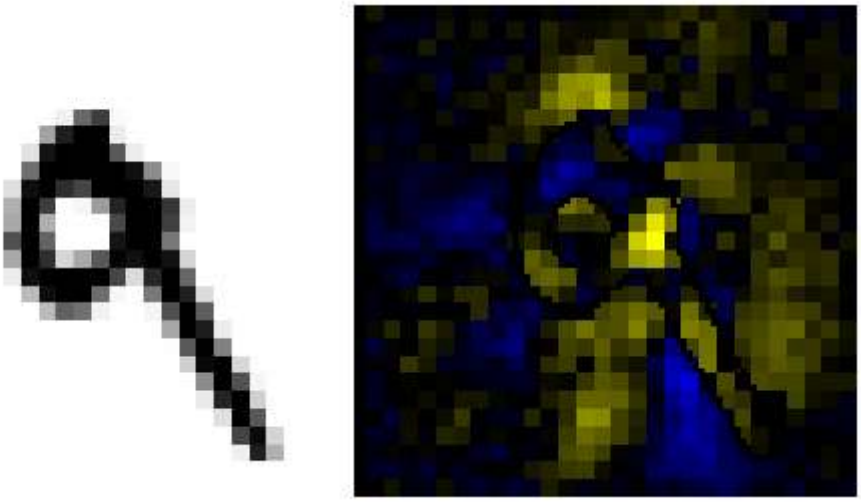



```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_7_blue-black-yellow.png
```

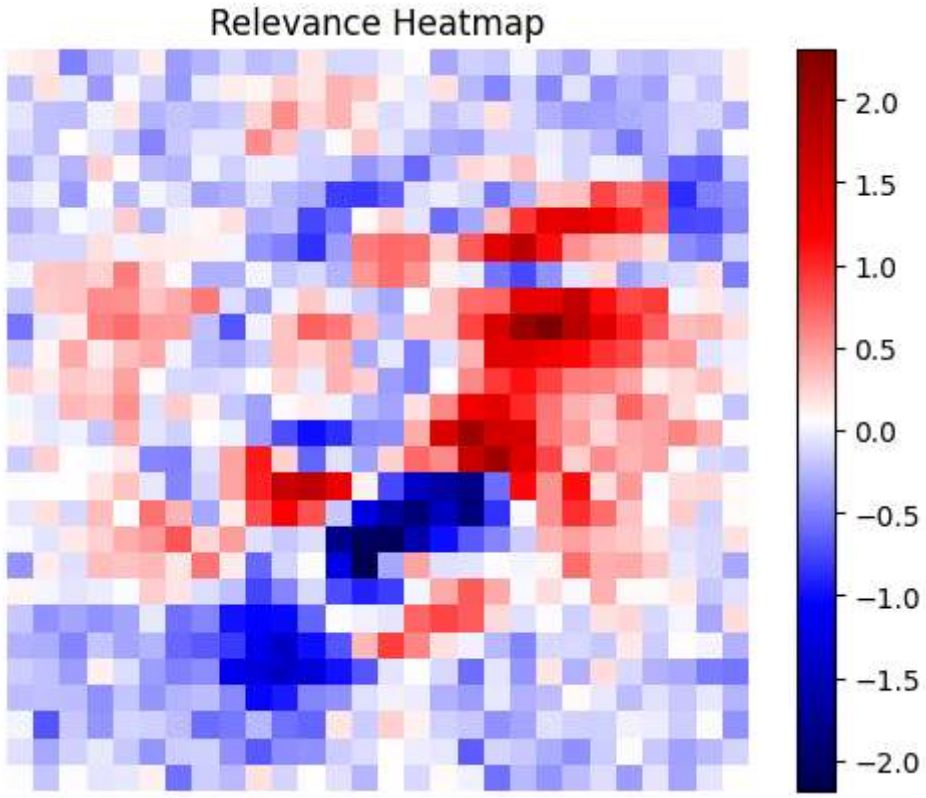




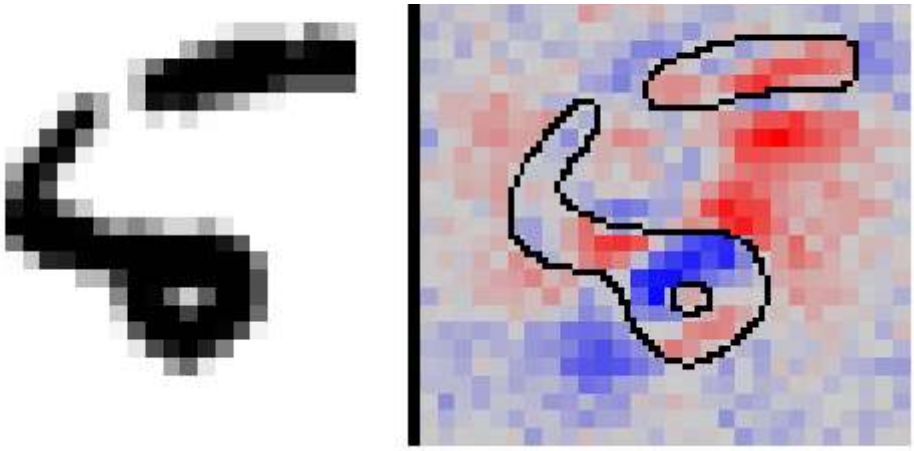
```
saving image to  ../heatmap_7.png
writing data in npy-format to ../heatmap.npy
```



```
True Class:      5
Predicted Class: 5
```

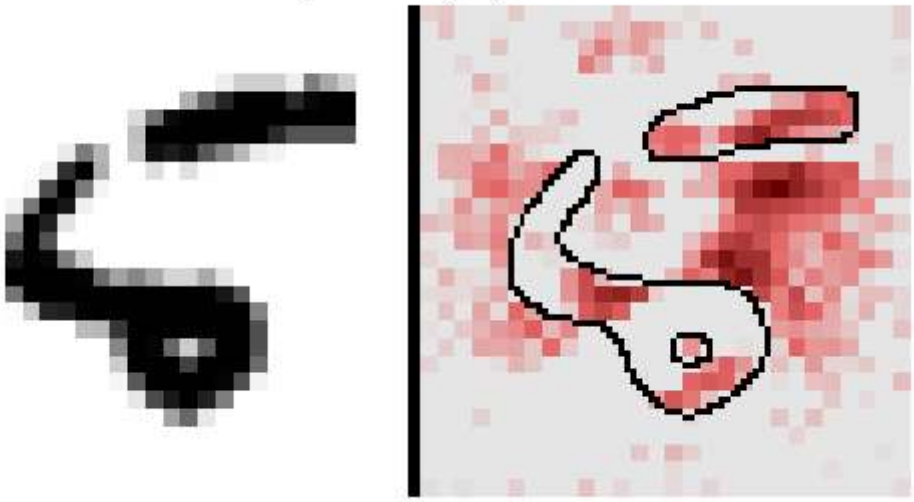



```
Applying custom heatmap: gray-red
saving image to  ../heatmap_8_gray-red.png
```



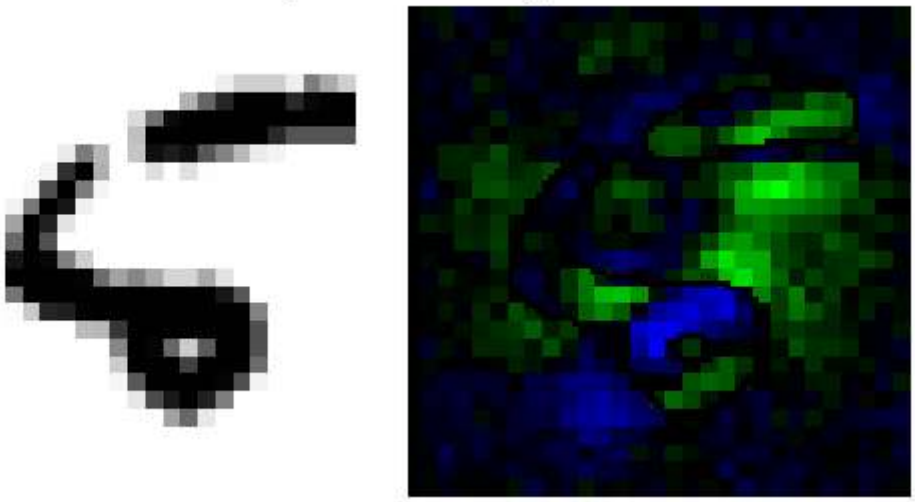
```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_8_gray-red2.png
```

Sample #8 - gray-red2



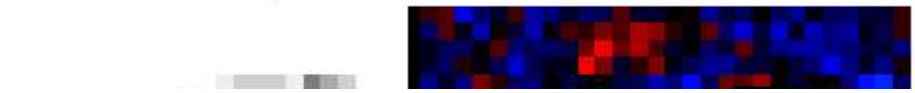
```
Applying custom heatmap: black-green
saving image to  ../heatmap_8_black-green.png
```

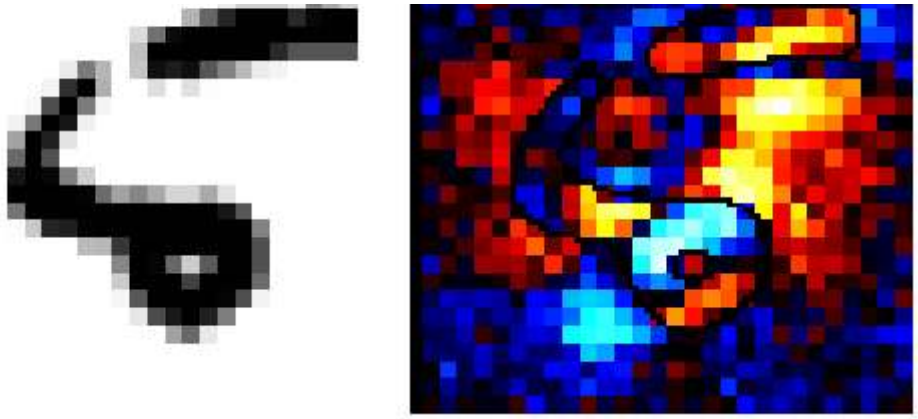
Sample #8 - black-green



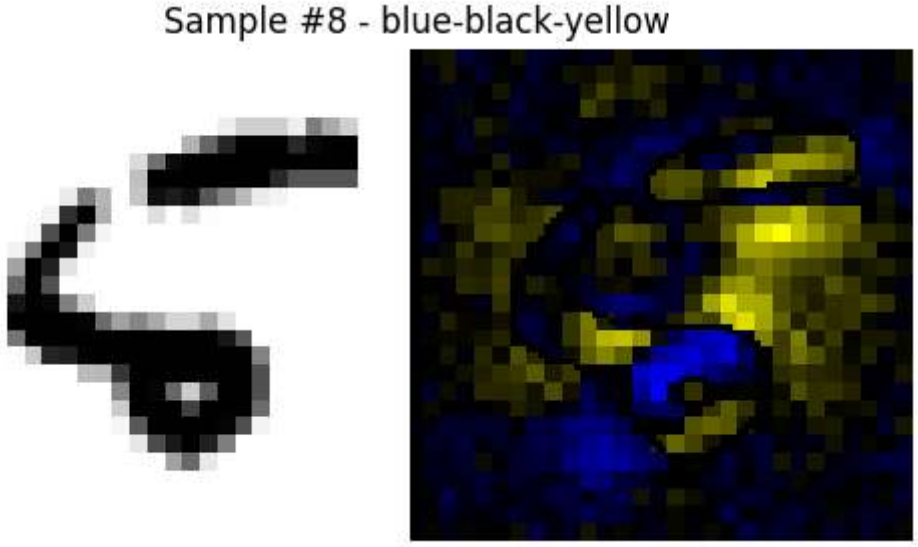
```
Applying custom heatmap: black-firered
saving image to  ../heatmap_8_black-firered.png
```

Sample #8 - black-firered

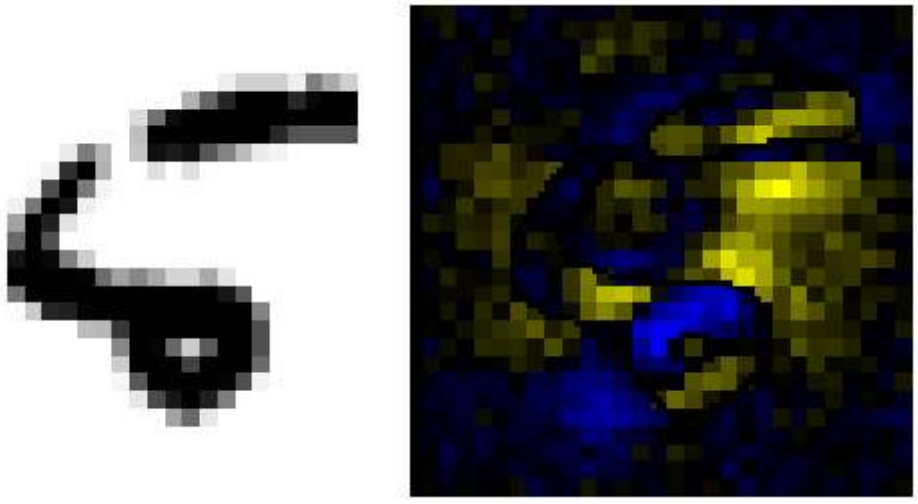




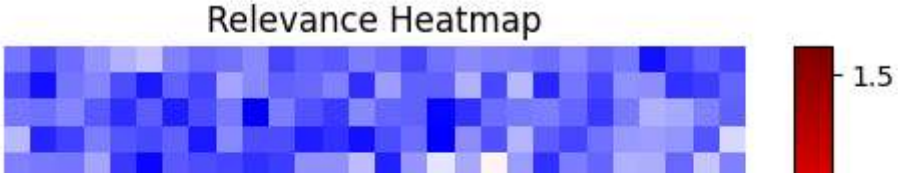
```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_8_blue-black-yellow.png
```

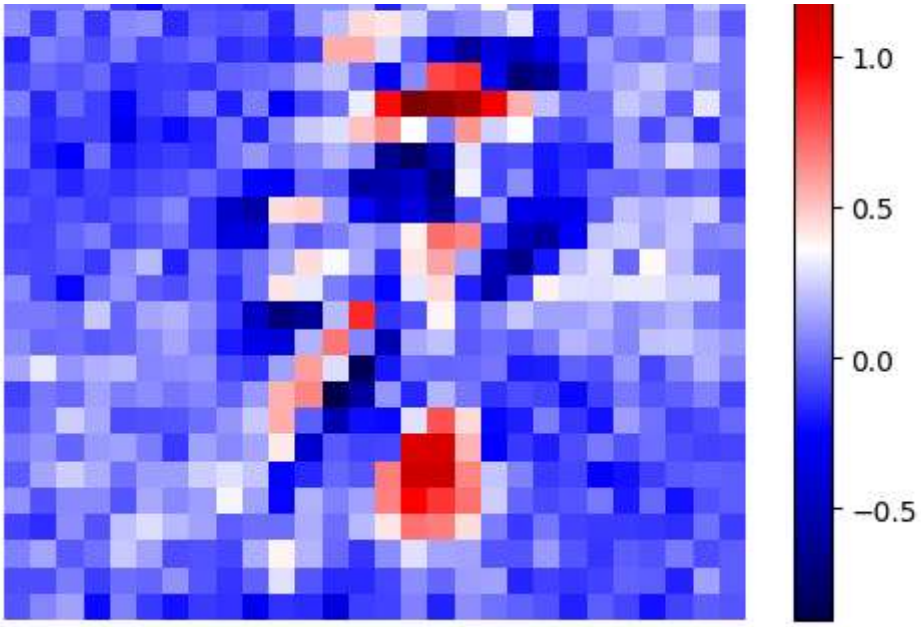



```
saving image to  ../heatmap_8.png
writing data in npy-format to ../heatmap.npy
```



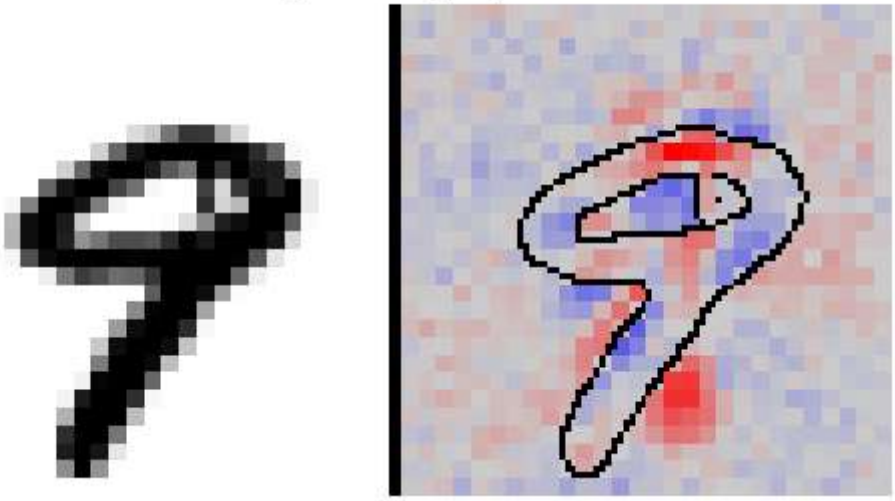
```
True Class:      9
Predicted Class: 9
```





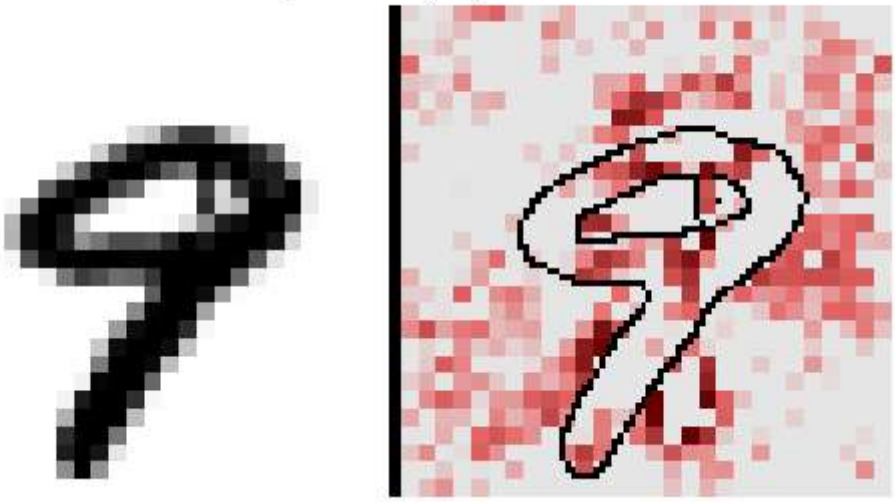

```
Applying custom heatmap: gray-red
saving image to  ../heatmap_9_gray-red.png
```

Sample #9 - gray-red



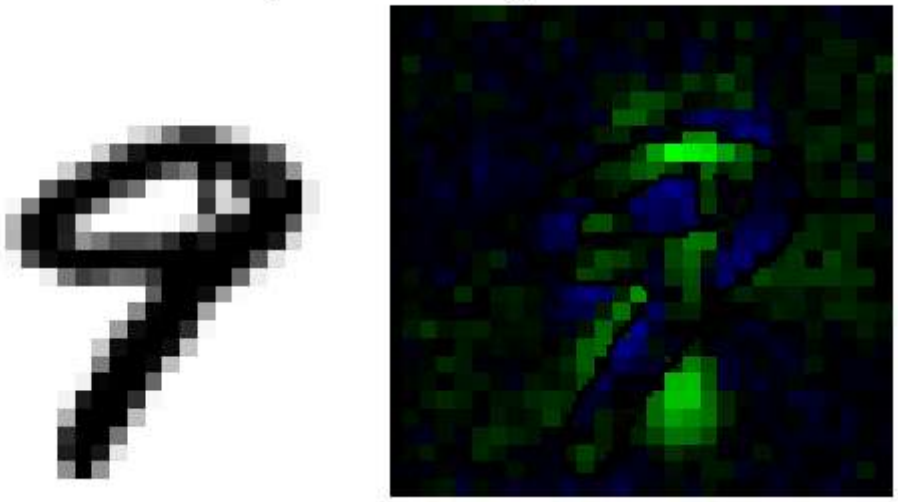
```
Applying custom heatmap: gray-red2
saving image to  ../heatmap_9_gray-red2.png
```

Sample #9 - gray-red2

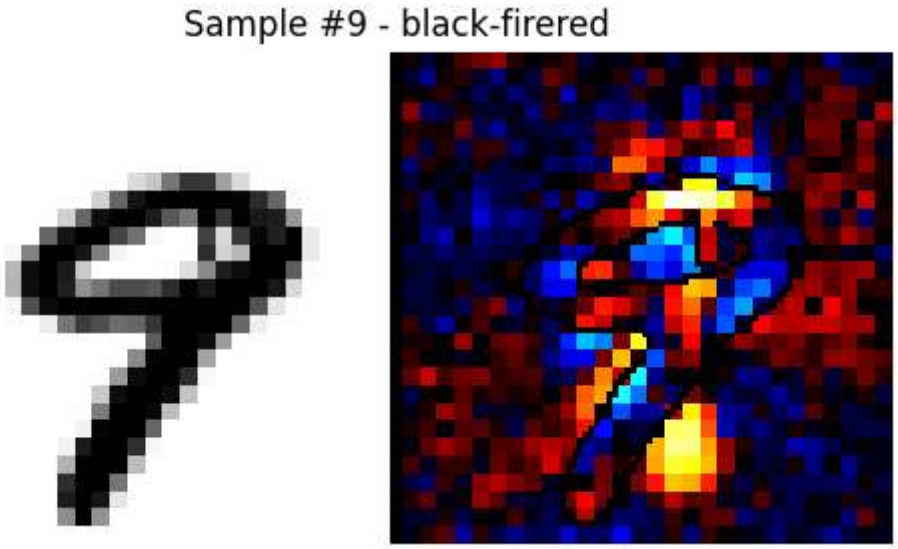


```
Applying custom heatmap: black-green
```

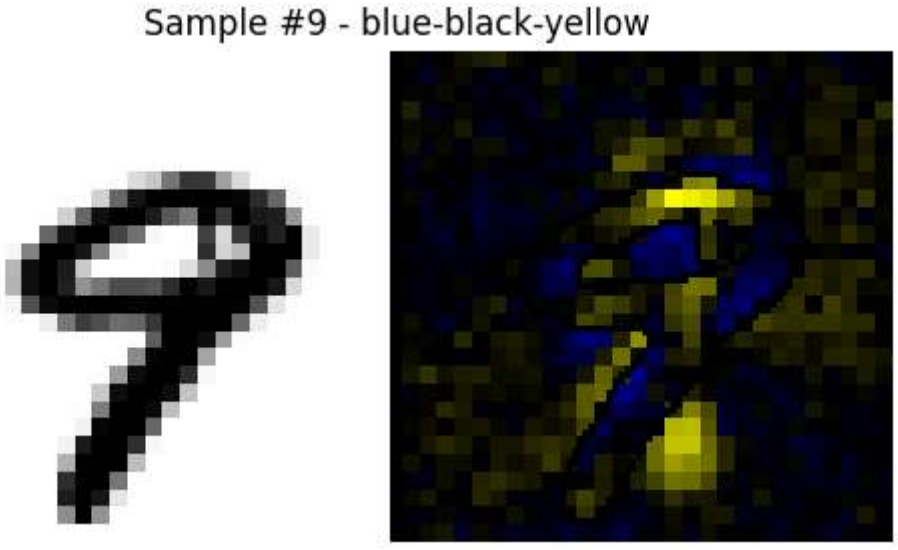
```
saving image to  ../heatmap_9_black-green.png
```



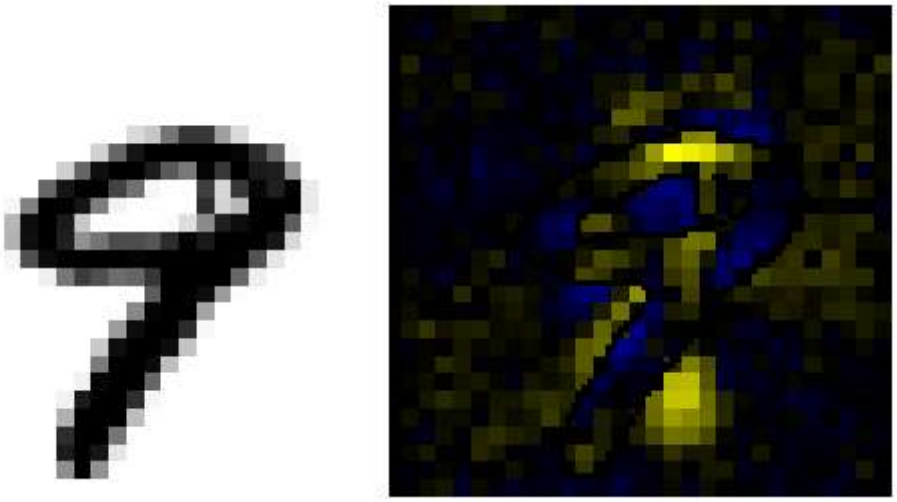

```
Applying custom heatmap: black-firered
saving image to  ../heatmap_9_black-firered.png
```




```
Applying custom heatmap: blue-black-yellow
saving image to  ../heatmap_9_blue-black-yellow.png
```

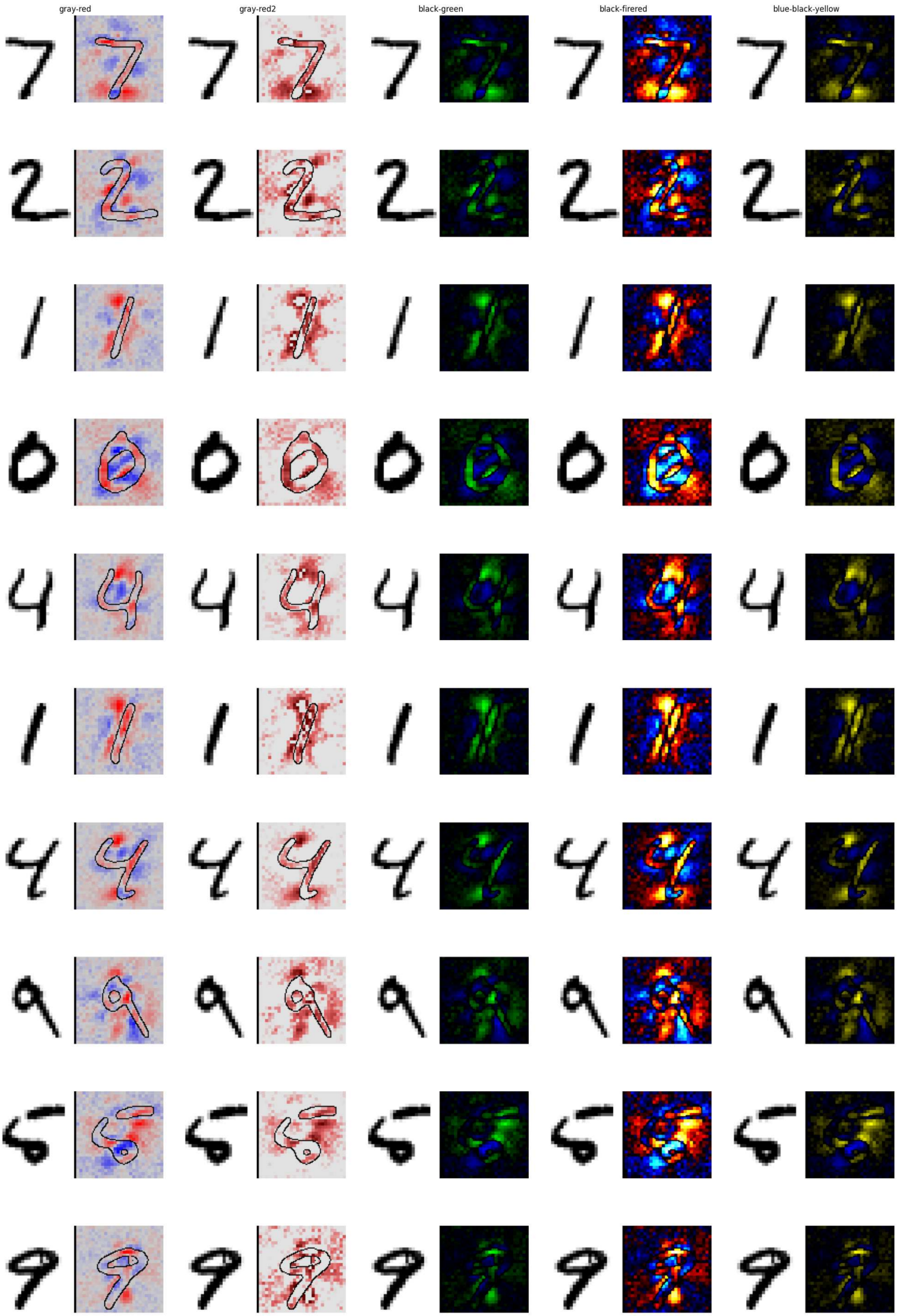



```
saving image to  ../heatmap_9.png
writing data in npy-format to ../heatmap.npy
```



```
writing data in npy-format to ../Rbatch.npy
Computation of 256 heatmaps using cupy in 0.261s
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

# Path to the folder where heatmaps were saved
base_path = '../'  # adjust if needed

num_samples = 10
cmap_names = [
    'gray-red', 'gray-red2', 'black-green', 'black-firered', 'blue-black-yellow'
]  # Same as keys in render.custom_maps

# Set up a figure grid: rows = samples, columns = colormaps
fig, axes = plt.subplots(num_samples, len(cmap_names), figsize=(4 * len(cmap_names), 3 * num

for i in range(num_samples):
    for j, cmap in enumerate(cmap_names):
        img_path = os.path.join(base_path, f'heatmap_{i}_{cmap}.png')
        img = mpimg.imread(img_path)

        ax = axes[i, j] if num_samples > 1 else axes[j]
        ax.imshow(img)
        ax.axis('off')
        if i == 0:
            ax.set_title(cmap)

plt.tight_layout()
plt.show()
```


gray-red
gray-red2
black-green
black-firered
blue-black-yellow

Timing MNIST

```
'''
@author: Sebastian Lapuschkin
@maintainer: Sebastian Lapuschkin
@contact: sebastian.lapuschkin@hhi.fraunhofer.de, wojciech.samek@hhi.fraunhofer.de
@date: 08.11.2017
@version: 1.0
@copyright: Copyright (c)  2015-2017, Sebastian Lapuschkin, Alexander Binder, Gregoire Mont
@license : BSD-2-Clause

compute execution times for different lrp computation variants, from naive to optimized.
'''

import time
import matplotlib.pyplot as plt
import importlib.util as imp
import numpy
import numpy as np
if imp.find_spec("cupy"): #use cupy for GPU support if available
    import cupy
    import cupy as np
na = np.newaxis




#load a neural network, as well as the MNIST test data and some labels
nn = model_io.read('../models/MNIST/long-rect.nn') # 99.17% prediction accuracy
X = data_io.read('../data/MNIST/test_images.npy')
Y = data_io.read('../data/MNIST/test_labels.npy')

# transfer pixel values from [0 255] to [-1 1] to satisfy the expected input / training par
X =  X / 127.5 - 1

# transform numeric class labels to vector indicator for uniformity. assume presence of al
I = Y[:,0].astype(int)
Y = np.zeros([X.shape[0],np.unique(Y).size])
Y[np.arange(Y.shape[0]),I] = 1

#permute data order for demonstration. or not. your choice.
I = np.arange(X.shape[0])



# do some benchmarking.
def benchmark():
    for B in [1 ,16, 64]: #batch sizes for work laptop with only 8GB RAM
    #for B in [1, 16, 64, 256]:
```

```
forward_times_old = []
forward_times_new = []
forward_times_aware = []

lrp_times_old = []  #simple
lrp_times_old2 = [] #epsilon
lrp_times_old3 = [] #alpha 2, beta -1
lrp_times_old4 = [] #alpha 1, beta 0
lrp_times_old5 = [] #alpha 0, beta 1

lrp_times_new = []  #s
lrp_times_new2 = [] #e
lrp_times_new3 = [] #a2b-1
lrp_times_new4 = [] #a1b0
lrp_times_new5 = [] #a0b1

lrp_times_aware = []  #s
lrp_times_aware2 = [] #e
lrp_times_aware3 = [] #a2b-1
lrp_times_aware4 = [] #a1b0
lrp_times_aware5 = [] #a0b1


print('#######################################')
print('Measuring Speed Gain for batch of {} on FCNN using {}'.format(B, np.__name_
print('#######################################')
for i in range(10):
    x = X[:B,:]
    # old code
    t_start = time.time();  yold = nn.forward(x);                              for
    t_start = time.time();  Rold = nn.lrp(yold, 'simple_slow');                lr
    t_start = time.time();  REold = nn.lrp(yold, 'epsilon_slow', 0.01);        lr
    t_start = time.time();  RAold2 = nn.lrp(yold, 'alphabeta_slow', 2.);          l
    t_start = time.time();  RAold1 = nn.lrp(yold, 'alphabeta_slow', 1.);          l
    t_start = time.time();  RAold0 = nn.lrp(yold, 'alphabeta_slow', 0.);          l
    # newer lrp code
    t_start = time.time();  ynew = nn.forward(x);                              for
    t_start = time.time();  Rnew = nn.lrp(ynew, 'simple');                     lr
    t_start = time.time();  REnew = nn.lrp(ynew, 'epsilon', 0.01);             lr
    t_start = time.time();  RAnew2 = nn.lrp(ynew, 'alphabeta', 2.);              l
    t_start = time.time();  RAnew1 = nn.lrp(ynew, 'alphabeta', 1.);              l
    t_start = time.time();  RAnew0 = nn.lrp(ynew, 'alphabeta', 0.);              l

    # lrp aware code and forward pass
    t_start = time.time();  yaw = nn.forward(x, lrp_aware=True);               for
    t_start = time.time();  Raw = nn.lrp(yaw, 'simple');                       lr
    t_start = time.time();  REaw= nn.lrp(yaw, 'epsilon', 0.01);                lr
    t_start = time.time();  RAaw2= nn.lrp(yaw, 'alphabeta', 2.);                l
    t_start = time.time();  RAaw1= nn.lrp(yaw, 'alphabeta', 1.);                l
    t_start = time.time();  RAaw0= nn.lrp(yaw, 'alphabeta', 0.);                l
```

```
        tolerance = 1e-8
        np.testing.assert_allclose(yold, ynew, rtol=tolerance) # predictions
        np.testing.assert_allclose(yold, yaw, rtol=tolerance)

        np.testing.assert_allclose(Rold, Rnew, rtol=tolerance) # simple lrp maps
        np.testing.assert_allclose(Rold, Raw, rtol=tolerance)

        np.testing.assert_allclose(REold, REnew, rtol=tolerance) # eps lrp maps
        np.testing.assert_allclose(REold, REaw, rtol=tolerance)

        np.testing.assert_allclose(RAold2, RAnew2, rtol=tolerance) # alpha2 lrp maps
        np.testing.assert_allclose(RAold2, RAaw2, rtol=tolerance)

        np.testing.assert_allclose(RAold1, RAnew1, rtol=tolerance) # alpha1 lrp maps
        np.testing.assert_allclose(RAold1, RAaw1, rtol=tolerance)

        np.testing.assert_allclose(RAold0, RAnew0, rtol=tolerance) # alpha0 lrp maps
        np.testing.assert_allclose(RAold0, RAaw0, rtol=tolerance)

        print('.',end='')

    print()
    print('    Mean Forward pass times:')
    print('      old:  ', numpy.mean(forward_times_old), '({}% speedup vs old)'.format(
    print('      new:  ', numpy.mean(forward_times_new), '({}% speedup vs old)'.format(
    print('      aware:', numpy.mean(forward_times_aware), '({}% speedup vs old)'.forma
    print('    Mean LRP times 1 (simple lrp):')
    print('      old:  ', numpy.mean(lrp_times_old), '({}% speedup vs old)'.format(int(
    print('      new:  ', numpy.mean(lrp_times_new), '({}% speedup vs old)'.format(int(
    print('      aware:', numpy.mean(lrp_times_aware), '({}% speedup vs old)'.format(in
    print('    Mean LRP times 2 (epsilon lrp):')
    print('      old:  ', numpy.mean(lrp_times_old2), '({}% speedup vs old)'.format(int
    print('      new:  ', numpy.mean(lrp_times_new2), '({}% speedup vs old)'.format(int
    print('      aware:', numpy.mean(lrp_times_aware2), '({}% speedup vs old)'.format(i
    print('    Mean LRP times 3 (alpha=2 lrp):')
    print('      old:  ', numpy.mean(lrp_times_old3), '({}% speedup vs old)'.format(int
    print('      new:  ', numpy.mean(lrp_times_new3), '({}% speedup vs old)'.format(int
    print('      aware:', numpy.mean(lrp_times_aware3), '({}% speedup vs old)'.format(i
    print('    Mean LRP times 4 (alpha=1 lrp):')
    print('      old:  ', numpy.mean(lrp_times_old4), '({}% speedup vs old)'.format(int
    print('      new:  ', numpy.mean(lrp_times_new4), '({}% speedup vs old)'.format(int
    print('      aware:', numpy.mean(lrp_times_aware4), '({}% speedup vs old)'.format(i
    print('    Mean LRP times 5 (alpha=0 lrp):')
    print('      old:  ', numpy.mean(lrp_times_old5), '({}% speedup vs old)'.format(int
    print('      new:  ', numpy.mean(lrp_times_new5), '({}% speedup vs old)'.format(int
    print('      aware:', numpy.mean(lrp_times_aware5), '({}% speedup vs old)'.format(i
    print('    Mean Total times with LRP once:')
    oldtotal = numpy.mean(numpy.array(lrp_times_old) + numpy.array(forward_times_old))
    newtotal = numpy.mean(numpy.array(lrp_times_new) + numpy.array(forward_times_new))
    awaretotal = numpy.mean(numpy.array(lrp_times_aware) + numpy.array(forward_times_a
```

```
        print('      old:  ', oldtotal, '({}% speedup vs old)'.format(int(100*(1 - oldtotal
        print('      new:  ', newtotal, '({}% speedup vs old)'.format(int(100*(1 - newtotal
        print('      aware:', awaretotal, '({}% speedup vs old)'.format(int(100*(1 - awaret
        print('    Mean Total times with LRP twice (simple+epsilon):')
        oldtotaltwice = numpy.mean(numpy.array(lrp_times_old) + numpy.array(lrp_times_old2)
        newtotaltwice = numpy.mean(numpy.array(lrp_times_new) + numpy.array(lrp_times_new2)
        awaretotaltwice = numpy.mean(numpy.array(lrp_times_aware) + numpy.array(lrp_times_a
        print('      old:  ', oldtotaltwice, '({}% speedup vs old)'.format(int(100*(1 - old
        print('      new:  ', newtotaltwice, '({}% speedup vs old)'.format(int(100*(1 - new
        print('      aware:', awaretotaltwice, '({}% speedup vs old)'.format(int(100*(1 - a
        print('    Mean Total times with LRP five times(simple,epsilon,alpha=2,alpha=1,alph
        oldtotalfive = oldtotaltwice + numpy.mean(numpy.array(lrp_times_old3) + numpy.array
        newtotalfive = newtotaltwice +  numpy.mean(numpy.array(lrp_times_new3) + numpy.arra
        awaretotalfive = awaretotaltwice +  numpy.mean(numpy.array(lrp_times_aware3) + nump
        print('      old:  ', oldtotalfive, '({}% speedup vs old)'.format(int(100*(1 - oldt
        print('      new:  ', newtotalfive, '({}% speedup vs old)'.format(int(100*(1 - newt
        print('      aware:', awaretotalfive, '({}% speedup vs old)'.format(int(100*(1 - aw
        print('')
        print('')

#run benchmark
```

```
      aware: 0.0016381502151489257 (8% speedup vs old)
    Mean Total times with LRP once:
      old:   0.0032855749130249025 (0% speedup vs old)
      new:   0.001794886589050293 (45% speedup vs old)
      aware: 0.0017532587051391601 (46% speedup vs old)
    Mean Total times with LRP twice (simple+epsilon):
      old:   0.004994511604309082 (0% speedup vs old)
      new:   0.003051614761352539 (38% speedup vs old)
      aware: 0.003071498870849609 (38% speedup vs old)
    Mean Total times with LRP five times(simple,epsilon,alpha=2,alpha=1,alpha=0):
      old:   0.011979770660400391 (0% speedup vs old)
      new:   0.00930490493774414 (22% speedup vs old)
      aware: 0.008894991874694825 (25% speedup vs old)


##################################
asuring Speed Gain for batch of 64 on FCNN using cupy
##################################
.......
  Mean Forward pass times:
    old:   0.0020311355590820314 (0% speedup vs old)
    new:   0.0007555484771728516 (62% speedup vs old)
    aware: 0.001950240135192871 (3% speedup vs old)
  Mean LRP times 1 (simple lrp):
    old:   0.0031230688095092774 (0% speedup vs old)
    new:   0.0027878046035766 (10% speedup vs old)
```