In [3]:
```
import torch
import torch.nn as nn  # nn stuff and loss
import torch.optim as optim  # optimization
import torch.nn.functional as F  # relu, tanh, functions with no params (nn also ha
from torch.utils.data import DataLoader  # helps create mini-batches of data to tra
import torchvision.transforms as transforms  # helpful transforms
from customImageSet import CustomImageDataset
from load_dataset import CImgDataset


class CNN(nn.Module):
    def __init__(
        self, input_size=1, num_classes=40
    ):  # input size 784 since 28x28 images
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels=input_size,
            out_channels=10,
            kernel_size=(3, 3),
            stride=(1, 1),
            padding=(1, 1),
        )  # stride and padding are standard
        self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
        self.conv2 = nn.Conv2d(
            in_channels=10,
            out_channels=num_classes,  # out_channels here controls col of mat1
            kernel_size=(3, 3),
            stride=(1, 1),
            padding=(1, 1),
        )
        self.fc1 = nn.Linear(18, num_classes)  # fully connected layer, row of mat2

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.reshape(x.shape[0], -1)  # number of examples sent in
        x = self.fc1(x)
        return x


# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


def train_model():
    # Hyperparameters
    input_size = 1  # row of mat2
    num_classes = 2
    learning_rate = 0.001
    batch_size = 25  # controls row of map1 if correct size or less, controls how m
    # so lower is more accurate but slower
    num_epochs = 50
```

```
    # Load data
    # Since going to load as image, convert to tensor
    # dataset = CustomImageDataset(root_dir="D:/test/data", transform=transforms.To
    dataset = CImgDataset("../test.zip")

    train_set, test_set = torch.utils.data.random_split(
        dataset, [0.8, 0.2]
    )  # first is row of map1
    train_loader = DataLoader(dataset=train_set, batch_size=batch_size, shuffle=Tru
    test_loader = DataLoader(dataset=test_set, batch_size=batch_size, shuffle=True)

    # Initialize network
    model = CNN(input_size=input_size, num_classes=num_classes).to(device)
    # model = CNN()
    x = torch.randn(100, 1, 25, 25)

    # Loss and optimizer
    criterion = nn.CrossEntropyLoss()  # could try MSELoss
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Train network

    for epoch in range(
        num_epochs
    ):  # one epoch = network has seen all images in dataset
        for batch_idx, (data, targets) in enumerate(train_loader):

            # get data to cuda if possible
            data = data.to(device=device)
            targets = targets.to(device=device)

            # forward
            scores = model(data)
            loss = criterion(scores, targets)

            # backward
            optimizer.zero_grad()  # set gradients to 0 for each batch to not store
            loss.backward()

            # gradient descent or adam step
            optimizer.step()
    return model, train_loader, test_loader


# Check training accuracy
def check_accuracy(loader, model, is_training):
    if is_training:
        print("Checking accuracy on training data")
    else:
        print("Checking accuracy on test data")
    num_correct = 0
    num_samples = 0
    model.eval()

    with torch.no_grad():
```

```
        for x, y in loader:
            x = x.to(device=device)
            y = y.to(device=device)

            scores = model(x)

            #  Shape of scores is 64 (originally) images * 10
            #  Want to know which one is the maximum of those 10 digits.
            #  I.e. if max value is first one, digit 0.
            _, predictions = scores.max(1)  #  max of second dimension
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)  #  size of first dimension

        print(
            f"Got {num_correct} / {num_samples} with accuracy {float(num_correct)/f
        )

    model.train()
    #  return float(num_correct)/float(num_samples)*100


def main():
    model, train_loader, test_loader = train_model()
    check_accuracy(train_loader, model, True)
    check_accuracy(test_loader, model, False)

    torch.save(model.state_dict(), "../model_weights_convolutional_example")
    model = CNN(input_size=1, num_classes=2).to(device)
    model.load_state_dict(torch.load("../model_weights_convolutional_example"))
    model.eval()

    check_accuracy(test_loader, model, False)

if __name__ == '__main__':
    main()
```

```
CUDA is available? True
16567 images in dataset
Checking accuracy on training data
Got 10949 / 13254 with accuracy 82.61
Checking accuracy on test data
Got 2748 / 3313 with accuracy 82.95
Checking accuracy on test data
Got 2739 / 3313 with accuracy 82.67
```

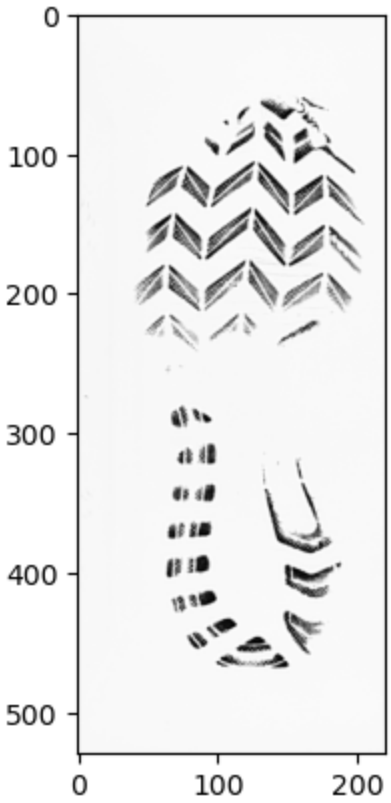
In [25]:

```
import numpy as np
import skimage.io as skio
import skimage.color as skcolor
import skimage.transform as sktrans
import matplotlib.pyplot as plt
import torch
from pathlib import Path
import imageio

from linear_NN import NN as MOD1
from convolutional_NN import CNN as MOD2
```

```
from recurrent_NN import RNN as MOD3
from recurrent_GRU_NN import RNN as MOD4
from recurrent_LSTM_NN import RNN as MOD5
from bidirectional_LSTM_NN import BidirectionalRNN as MOD6

class ModelMaker:
    def __init__(self, initfunc, wts_file, reshape=False):
        self.starter = initfunc
        self.net = initfunc()
        self.net.load_state_dict(torch.load(wts_file, map_location=torch.device("cp
        self.net.eval()
        self.reshape = reshape
        #print(self.net)

    def __call__(self, image):
        x = torch.from_numpy(image)
        # print(x.shape)
        # if there is an error it's likely due to shaping issues
        # add an unsqueeze(0) above if convolutional
        # don't if bidirectional
        if self.reshape:
            x = x.reshape(1, -1)
        else:
            x = x.unsqueeze(0).unsqueeze(0)
        res = self.net(x)
        res_max = torch.max(res)
        res_arg = torch.argmax(res)
        if res_arg == 0:
            res_max *= -1
        return res_max.detach().numpy()


MOD1.make = lambda: MOD1(input_size=225, num_classes=2)
MOD2.make = lambda: MOD2(input_size=1, num_classes=2)
MOD3.make = lambda: MOD3(15, 256, 3, 2, 15)
MOD4.make = lambda: MOD4(15, 256, 3, 2, 15)
MOD5.make = lambda: MOD5(15, 256, 3, 2, 15)
MOD6.make = lambda: MOD6(15, 256, 3, 2)


def run_on_image(maker, img_name, nn_type, num_epochs=None):
    #img = skcolor.rgb2gray(skio.imread(img_name)[:, :, :3])
    img = skio.imread(img_name)
    if len(img.shape) == 3 and img.shape[2] >= 3:
        img = skcolor.rgb2gray(img[:, :, :3])
    tform = sktrans.EuclideanTransform(rotation=0, translation=(7, 7))
    padded_image = np.float32(
        sktrans.warp(
            img,
            tform.inverse,
            output_shape=(img.shape[0] + 14, img.shape[1] + 14),
            mode="reflect",
        )
    )

    fig, axs = plt.subplots(1, 2, sharex=True, sharey=True)
```

```
    #plt.rcParams['figure.figsize'] = 220, 530
    axs[0].imshow(img, "Greys_r")

    res = np.zeros(img.shape, dtype=np.float32)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # print(i,j)
            i2 = i + 7
            j2 = j + 7
            sub_image = padded_image[i2 - 7 : i2 + 8, j2 - 7 : j2 + 8]
            res[i, j] = maker(sub_image)

    axs[1].imshow(res, "Reds")

    plt.show()

    plt.imshow(res, "Reds", aspect='equal')
    name = Path(img_name).stem
    plt.axis("off")
    if (num_epochs):
        plt.imsave("../heatmaps/" + name + "_" + nn_type + "_epochs_" + str(num_epo
    else:
        plt.imsave("../heatmaps/" + name + "_" + nn_type + ".png", res, cmap="Reds"

    #res = np.maximum(res, 0.95*np.max(res))
    res = res * 1.0 * (res > 0.85 * np.max((res)))

    plt.imshow(res, "Reds")
    plt.show()

    plt.imshow(img, "Greys")
    plt.imshow(res, "Reds", alpha=0.6)

    plt.gcf().set_size_inches(220 / 77, 530 / 77)  # 77 is the golden number takes
    plt.axis("off")
    if (num_epochs):
        plt.savefig("../heatmaps/" + name + "_" + nn_type + "_epochs_"
                    + str(num_epochs) + "_compare.png", bbox_inches='tight', pad_in
    else:
        plt.savefig("../heatmaps/" + name + "_" + nn_type + "_compare.png", bbox_in

    plt.show()

def main():
    wts_file = "../model_weights_convolutional"
    make = ModelMaker(MOD2.make, wts_file, False)
    f = "../002_07_L_01.png"
    print("Creating heatmap...")
    run_on_image(make, f, wts_file[17:])

if __name__ == "__main__":
    main()
```

```
Creating heatmap...
```


0
100
200
300
400
500
0
100
200


0
100
200

In [ ]: