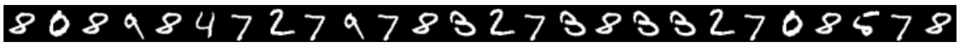
AE1205 Assignment 6: Automatic number recognition

A group of professors seems to be planning something shady in the aerospace faculty. ICT has drawn our attention to the fact that they are exchanging many encrypted messages. Specifically, they exchange images that only contain numbers. The decrypting software of the ICT department cannot be applied to these images, since the software expects text files, not images.

This is why we call on you - our Python expert - to write a program that automatically reads the image and transforms it to a string of numbers that can be decoded. Of course, we secretly hope that you will be able to decipher the secret message in the following image, that the ICT department has seen on numerous occasions; It must contain an important message.



Secret and probably devious message exchanged by Aerospace professors.

Template matching

Your first idea for tackling the problem is to use *template matching*. As the name suggests, you will use a template for each written number and try to match them to the written numbers in the secret message. Of course, the problem is that everyone writes numbers in different ways - a problem that is also faced by programs that automatically sort the physical mail by automatically reading addresses and postal codes. In order to make templates that take into account the natural variation in written numbers, we will download a database of numbers written by many different persons. Please download the files **`MNIST.dat`** and - while you're at it, also **`secret_code.png`**.

The `MNIST.dat` file is a binary file. It contains the variables of the written number database, which were written to disk with the `pickle` function. Since it is not part of the course material, we provide the code of how to load the data below. Remark that we open the file with `'rb'`, meaning that we want to read the file in **binary** format. Although not relevant for this assignment, it is worth to know that saving a variable can be done with the `pickle.dump()` function. The last part of the code below shows a single image in the data set together with its 'label', i.e., the number that was written down.

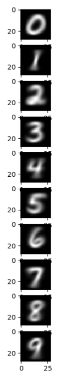
```python
import pickle
from matplotlib import pyplot as plt

# load the database of labeled number images:
# original dataset: http://yann.lecun.com/exdb/mnist/
file = open('MNIST.dat', 'rb')
MNIST = pickle.load(file)
file.close()
images = MNIST[0]
labels = MNIST[1]
shape_image = images[0].shape

# show a single number image plus label:
plt.figure()
plt.imshow(images[0], cmap=plt.get_cmap('gray'))
plt.title(labels[0])
plt.show()
```

Step 1: Making the templates

We propose that you make templates that represent the **average image** of written numbers 0-9, from the image data you loaded with pickle (`MNIST[0]`). With average image, we mean that each pixel in the template represents the average illuminance in that pixel location, i.e., averaged over all examples of that written number in the dataset. When you've made a list of 28x28 pixel templates, which can be represented with a 2d numpy array, you can pass them to the following function to show them.

```python
def show_templates(average_templates):
    n_templates = len(average_templates)
    fig, axs = plt.subplots(n_templates)
    for i in range(n_templates):
        axs[i].imshow(average_templates[i], cmap=plt.get_cmap('gray'))
    plt.show()
```

The result should look like this:



Average images for written numbers.

Step 2: Reading the image containing the secret code

The second step consists of reading the image containing the secret code. You can read the image using `pygame.image.load()` (covered in the course). Be sure to check the read image and adapt the format so that it matches that of the database. Make sure that the image is in the right orientation (you can check that by showing the image with pyplot), and make sure that the resulting image data is grayscale, not RGB. Check PyGame's `Surface` documentation (online) for methods that allow you to access pixel data.

Step 3: Converting the image to a list or string of numbers

When you have the image of the secret code loaded in your code, you can now cut out *image patches* of 28×28 pixels and compare them with the templates. Per image patch, you should determine the *Euclidian* distance between the image patch and each template. Then pick the number with the shortest Euclidian distance as the matching number. Please note that in this case the Euclidian distance is the square root of the sum of squared, per-pixel differences.

Step 4: Decode the message

For decoding the message, we expect the encryption method not to be too hard. Moreover, ICT saw some professors logging on during lecture 3 of the Python course, during the explanation of character representations, and we have the suspicion that some elementary method from that lecture is being used to encode for the letters in the message. When decoding, please take into account that the template matching may not work perfectly. You can check this by simply comparing the written numbers and the numbers detected by your program. So there may be a bit of noise in the message.

Optional:

As mentioned, the decoding may not work optimally. In this optional part of the program, you can try to generate an uncertainty measure for the decoding of each number. One option is to use the matching distance of the closest number template. Another option is to compare the distances to the two closest templates. Can you single out the wrongly decoded number(s)?