UNIVERSITÀ DI SIENA 1240

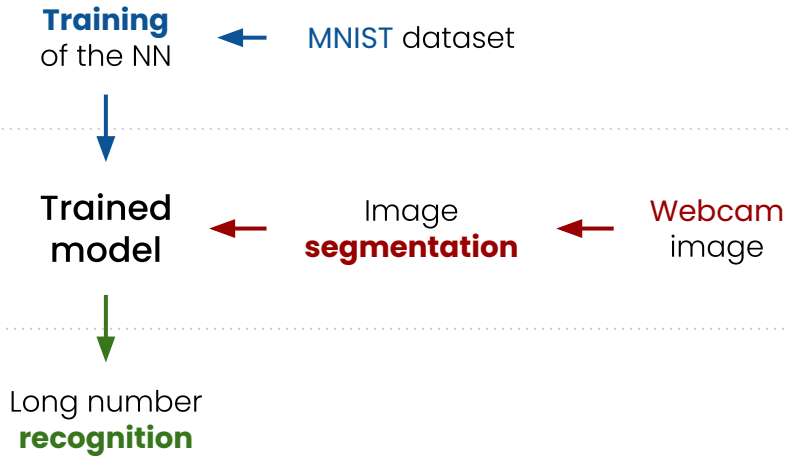
Handwritten (long) **numbers recognition**

Neural Networks project

Filippo Guerranti
filippo.guerranti@student.unisi.it

Goal and Workflow (README.md)

The aim of this project is to build a **CNN model** trained on **MNIST** dataset and to exploit its classification capabilities to recognize a sequence of several single handwritten digits (that can be considered as a **long number**) given as an input image that the user can take from the **webcam**.



Phase 1 Training of the model

- MNIST dataset decoding and management
- CNN model implementation
- Training of the model

Phase 2 Input image segmentation and digits extraction

- Webcam image capture
- Image segmentation
- Digits extraction

Phase 3 Long number recognition

- Input of the network: extracted digits
- Output of the network: long number
- Results

Github Repository

See README.md for further details.

MNIST dataset decoding and management

MNIST dataset is downloaded from the source (IDX file format), decoded accordingly and stored in a `torch.Tensor`. The MNIST class provides several method to split the dataset, set preprocessing operations and get some statistics.

Download, decode and store procedures

- Dataset download `(modules.dataset.download)`
- File decoding and storing to `torch.Tensor` `(modules.dataset.store_file_to_tensor)`

USAGE EXAMPLE

```
download(url, folder, name)

data = store_file_to_tensor(image_filepath)
labels = store_file_to_tensor(labels_filepath)
```

MNIST class `(modules.dataset.MNIST)`

- Subclass of `torch.utils.data.Dataset`
- `__len__()` and `__getitem__()` overloading
- Dataset split according to proportions
- Set preprocess (**data augmentation**) operation

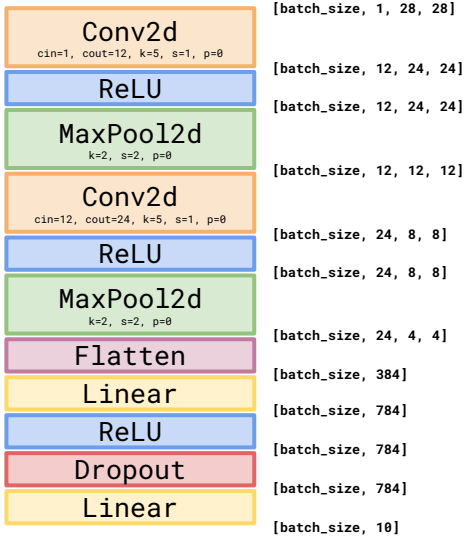
USAGE EXAMPLE

```
data_set = MNIST(folder, train=True, download=True)
test_set = MNIST(folder, train=False)

train_set, val_set = data_set.splits(proportions,
shuffle=True)

train_set.set_preprocess(data_augmentation_operation)
```

CNN model implementation

The classifier is a CNN having the architecture shown below and is handled by the CNN class which implements several methods to train and evaluate the classifier along with the possibility of classifying an input image (or batch of images).

Model Architecture



CNN class `(modules.cnn.CNN)`

- Loss: **cross entropy**
- Decision: **argmax**
- Optimizers: **Adam**
- Training procedure: `train_cnn()`
- Evaluation procedure: `eval_cnn()`

USAGE EXAMPLE

```
classifier = cnn.CNN(data_augmentation=True, device)

classifier.train_cnn(train_set, val_set, batch_size, lr, ep)

train_accuracy = classifier.eval_cnn(train_set)
val_accuracy   = classifier.eval_cnn(val_set)
test_accuracy  = classifier.eval_cnn(test_set)
```

Training

The training results are shown in the result slide

The training procedure is done both with data augmentation and without it. In this project the data augmentation technique consists of a **random rotation** (between -30° and +30°), followed by a **crop of random scale** (between 0.9 and 1.1) and **of random ratio** (between 3/4 and 4/3) of the original size which is then **resized** to the original 28x28 size.

Training phase `(modules.utils.train())`

- CNN classifier initialization
- MNIST dataset preparation into training, validation and test sets
- CNN training: `train_cnn()`
- CNN evaluation on datasets: `eval_cnn()`

In terminal `$ python3 hlnr.py train -h`

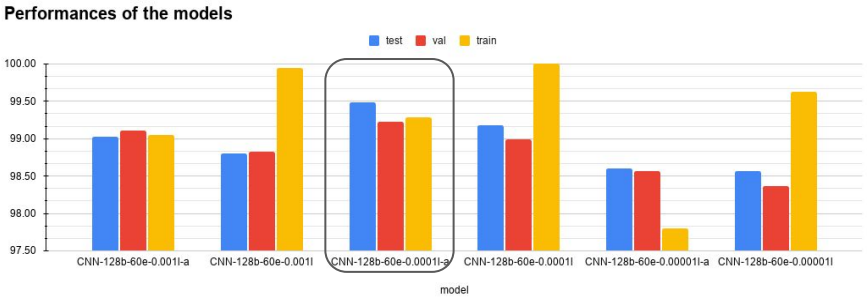
```
usage: hlnr.py train [-h] [-a] [-s TRAIN VAL] [-b BATCH_SIZE]
[-e EPOCHS] [-l LEARNING_RATE] [-w NUM_WORKERS] [-d DEVICE]
```

`$ python3 hlnr.py train`

data augmentation: **false**
parameters: **default**

`$ python3 hlnr.py train -a`

data augmentation: **true**
parameters: **default**



Webcam capture

The webcam capture is strongly based on the OpenCV library. When called by the `modules.utils.classify()` function, it **opens the camera** and allows the user to **take an image** of the handwritten digits. In the same function, along with the possibility of taking a webcam picture, it is also possible to provide an image coming from a **user folder**.

Real-time image capture (`modules.utils.webcam_capture()`)

- opens the webcam (`cv2.VideoCapture(0)`)
- shows the captured frames in a while loop until:
  - `SPACE` key is pressed: **take a snapshot**
  - `ESC` key is pressed: **close webcam and exit**
- once the snapshot is taken, it is directly send to the CNN model in order to be classified

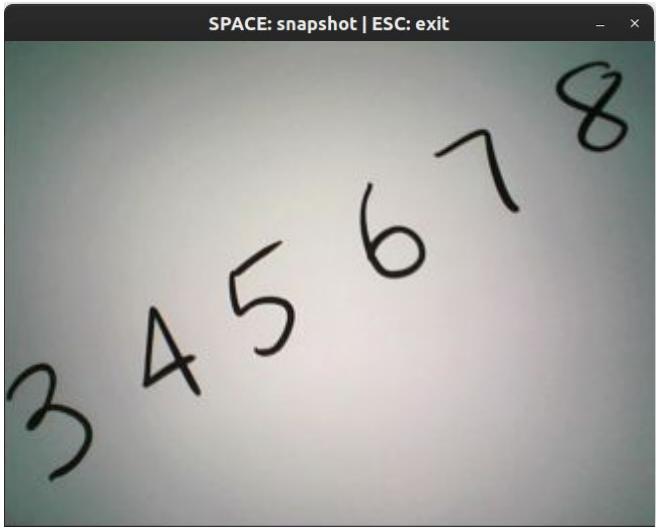
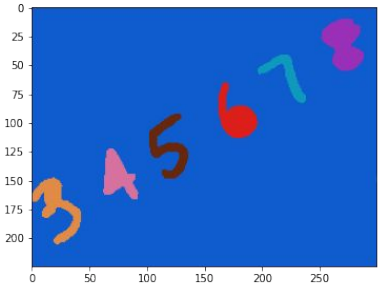
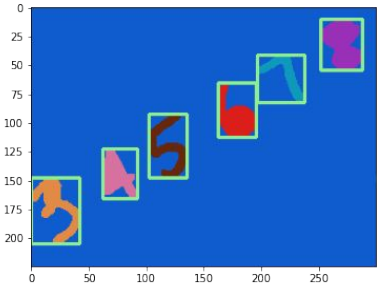


Image segmentation

DisjointSetForest class

- Used as base data structure for segmentation
- Stores disjoint non-overlapping sets
- Provides operations for:
  - add new sets
  - `merge` sets (replacing them by their union)
  - `find` representative member of a set





GraphBasedSegmentation class

- Graph-based image segmentation (paper)
- Exploits disjoint-set forest data structure

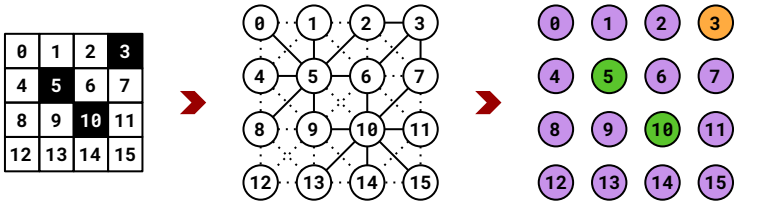


Image pixels
(black = 0, white = 255)

Graph
(weights are the difference between pixels' intensities)

Segmented regions
(each region is a subset in the disjoint-set forest)

USAGE EXAMPLE

```
segmented = segmentation.GraphBasedSegmentation(image)
segmented.segment(k=4500, min_size=100,preprocessing=True)
segmented.generate_image()
segmented.digits_boxes_and_areas()
```

Digit extraction

Digit extraction procedure

Carried out by the `extract_digits()` method of the `GraphBasedSegmentation` class. Given the region boundaries:

- regions are sliced out from the original image
- slices are resized according to the MNIST dataset samples dimensions (28x28)
- resized slices are modified in order to obtain an image which is as close as possible to the one that the network saw in training phase
- modified slices are converted into a `torch.tensor` which will be used as input to the network

Extracted digits

Note: having trained the network using data augmentation (random rotations and random crop and resize) should help in correctly classifying the digits which are extracted from a number that has been written in diagonal.

Input of the network: extracted digits

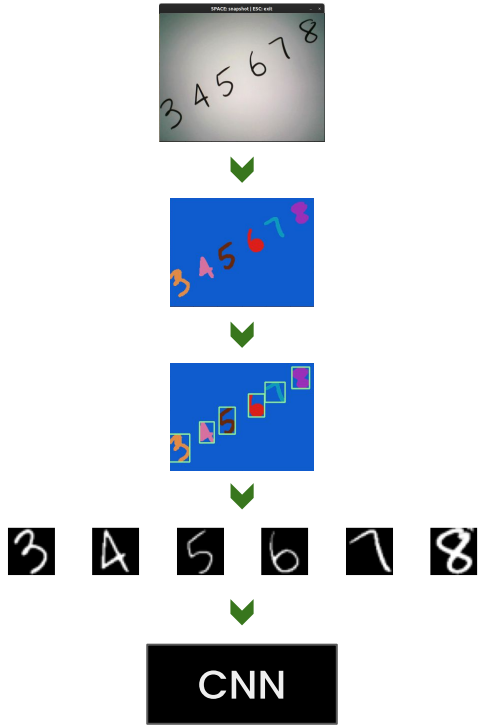
From input image to extracted digits `(modules.utils.classify())`

Takes the input arguments from the `main_args_parse()` function and:

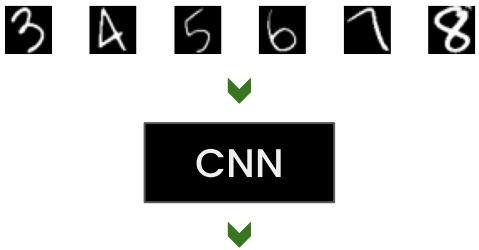
- starts the webcam image capture procedure (**default** behaviour) or takes an input image from a folder defined by the user
- initializes the CNN classifier
- loads the pre-trained model with data augmentation or the one without it
- segments the image via the `segment()` method of the `GraphBasedSegmentation()` class
- extracts the digits via the `extract_digits()` method of the `GraphBasedSegmentation()` class

In terminal `$ python3 hlnr.py classify -h`

```
usage: hlnr.py classify [-h] [-f PATH_TO_IMAGE] [-a | -m PATH_TO_MODEL] [-d DEVICE]
```



Output of the network: long number



`>>> The recognized number is: 345678`

From extracted digits to recognized long number `(modules.utils.classify())`

The extracted digits are passed as input parameters to the `classify()` method of the `CNN()` class (since they are `torch.tensor`s) and the CNN returns the classification for each digit.

In terminal

```
$ python3 hlnr.py classify
```

image from: **webcam**
model: default pre-trained model **without** data augmentation

```
$ python3 hlnr.py classify -a
```

image from: **webcam**
model: default pre-trained model **with** data augmentation

Results

Considering the model that performed better on the test set (`CNN-128b-60e-0.0001l-a`) the following **results** have been noticed:

- Handwritten numbers which appear in diagonal are generally recognized from the networks, due to the **data augmentation** techniques that apply random rotations and random crops and resizes (this helps the network to recognize digits which are slightly rotated or zoomed)

```
>>> The recognized number is: 23745
```

- The network still has troubles recognizing the 1, 7 and 9: they are often **misclassified**. The reason for this could be that MNIST dataset is composed of 1s which are written as vertical single lines, while in the test the network saw 1s composed of more than one line (one at the to and one at the bottom) which could lead to misclassification with 2 or 7.

```
>>> The recognized number is: 2777
```