

# COMP 2012H Honors Object-Oriented Programming and Data Structures

## Assignment 10 AI Digit Recognition: Data Preprocessing and Network Testing

Menu

- [Honor Code](#)
- [Objectives & ILOs](#)
- [Introduction](#)
- [Description](#)
- [Tasks](#)
- [Resources & Sample I/O](#)
- [Submission & Grading](#)
- [Change Log](#)
- [FAQ](#)

Page maintained by

CHEN, Jierun  
Email: [jierunchen@ust.hk](mailto:jierunchen@ust.hk)  
Last Modified: 11/20/2022 00:21:46

Homepage

[Course Homepage](#)

Honor Code

We value academic integrity very highly. Please read the [Honor Code](#) section on our course webpage to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment. Please read the [Honor Code](#) thoroughly.
- Serious offenders will fail the course immediately, and there will be additional disciplinary actions from the department and university, upto and including expulsion.

End of Honor Code


Objectives & Intended Learning Outcomes

The objective of this assignment is to familiarise you with Standard Template Library (STL). After this assignment you should be able to:

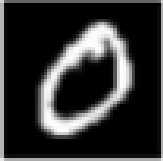
- Choose and use proper STL containes to store data efficiently, and do manipulation in the stored data.
- Use STL iterators to iterate through the elements of a container.
- Use various STL algorithms to manipulate STL containers.
- Develop and use function objects to further leverage STL library.

End of Objectives & Intended Learning Outcomes

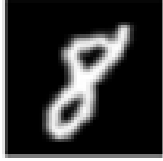
label = 5



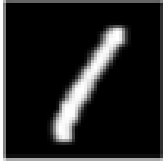
label = 0




label = 8




label = 1



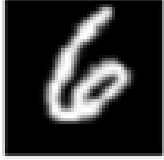
label = 9




label = 2



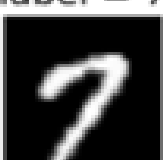
label = 6




label = 3



label = 7



label = 4



MNIST digit dataset

Source: <https://ejleep1.tistory.com/932>

Introduction

The goal of this assignment is to do data preprocessing and network testing for digit recognition. Digit recognition is the process to provide the ability to machines to recognize digits (human handwritten digits here). To achieve this, we leverage a simple neural network

and a digit dataset. Before the network be ready to perform digit recognition, it will have to go through a training phase and a testing phase.

- During the training phase, "train" split of the dataset is used to teach the network how to make predictions. The "train" split contains a collection of image and label pairs. For example, as shown in the figure aboved, each digit image is accompanied with a corresponding label or ground truth. We train a network by feeding images to it and encouraging it to make predictions that are close to the corresponding labels. Therefore, enough and high-quality "train" data are crital for training an accurate network. You will explore and build a sense on this throughout the assignment. Beisides, you can treat the network as a black box and skip how the neural network learns from the dataset.
- During the testing phase, "test" split of the dataset is used to evaluate how well the trained network makes the correct predictions. The "test" split also contains a collection of image and label pairs. Images are fed into the trained network as input one by one. Then the predictions from the trained network are compared with the label. A prediction is called to be correct if it matches with the corresponding label.

You will explore and build a sense on how a complete and class-balanced "train" data plays an important role in obtaining an accurate network throughout the assignment. "Class-balanced" means that each class in the dataset has roughly the same number of images and labels. The program pipeline is as follows:

1. Train a network with the complete and class-balanced "train" data, test it on the "test" data and report the overall and digit-wise accuracy.
2. Remove part of the "train" data for a digit, making the dataset class-imbalanced. You then retrain a network, test it again and report the accuracy. Now the accuracy for that specific digit is expected to be lower than other digits.
3. Generate more "train" data for the above digit by augmenting the original images. Augmenting an image can be done by applying some geometrical transform without affecting the fidelity of the label, such as image shifting, resizing and cropping.

In this assignment, you will implement part of the above pipeline mainly on data preprocessing and network testing by making use of STL containers, iterators, algorithms and function objects.

End of Introduction

## Description

Please read the [FAQ](#) and [Change Log](#) section regularly, and do check it one day before the deadline to make sure you don't miss any clarification, even if you have already submitted your work by then. You can also raise questions on Piazza and remember to use the "pa10" tag.

## Code structure

The skeleton code structure is as follows:

```
PA10
├── dataset.cpp
├── dataset.h
├── main.cpp
├── nn.cpp
├── nn.h
├── train_test.cpp
├── t10k-images-idx3-ubyte
├── t10k-labels-idx1-ubyte
├── train-images-idx3-ubyte
└── train-labels-idx1-ubyte
```

`main.cpp` contains the pipeline interface;

`nn.cpp` and `nn.h` hold the NN struct definition and the member function definitions as well as some layer structs definition that are used in the NN struct.

`dataset.cpp` and `dataset.h` hold the dataset class definition and the member function definitions as well as some global const, enum and new data type definitions.

train\_test.cpp holds the training and testing related function definitions.

The last four files are the "train" and "test" split of the digit dataset, each split with one image file and one label file.

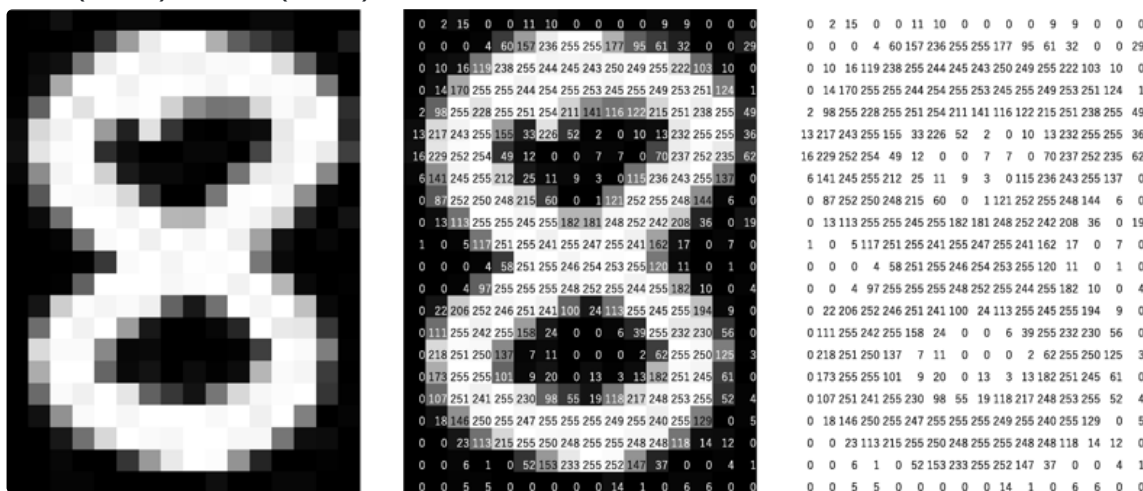
Your job is to complete `dataset.h`, `dataset.cpp` and `train_test.cpp` so that the pipeline can run through all the intended steps.

## How is the digit dataset loaded, stored and accessed?

The dataset is loaded into a `dataset` class object by calling the `void load_dataset()` member function, and in particular loaded into two variables for labels and two variables for images:

- `train_labels` and `test_labels`  
vector<uint8\_t> objects. Each element in the vector represents a label. The `uint8_t` data type is defined as unsigned char with only 8-bits to save memory space.
- `train_images` and `test_images`  
vector<array<uint8\_t, PIXELS>> objects. Each array<uint8\_t, PIXELS> element represents an image with height ROWS, width COLS and thereby ROWS\*COLS=PIXELS pixels. ROWS, COLS, and PIXELS are constant variables defined in `dataset.h`.

Each pixel's brightness is represented by a `uint8_t` (unsigned char) number, whose range is from 0 (black) to 255 (white):



Source: <https://github.com/NehaCkumari/MNIST-Handwritten-Digit-Recognition-using-CNN>

In an array, the pixels are arranged in the up-bottom and left-right order, as shown in the figure below:

How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

0	1	2	3	4	5	6	7	8	9	.	.	.		
---	---	---	---	---	---	---	---	---	---	---	---	---	--	--

Source: <https://processing.org/tutorials/pixels>

Most of the functions you need to implement are class member functions, which can directly access the four private dataset variables. At rare cases, you need to access these four private data members outside the class. For that, we provide the following accessor member functions:

```
uint8_t get_a_label(const enum data_type mode, const size_t index) const;
```

**Description** - Return a label from train labels or test labels at a specific position.

**Parameters**

- `mode` - To indicate the dataset spilt to get a label, assumed to be either `TRAIN_LABEL` OR `TEST_LABEL`.
- `index` - The position at which the label is retrieved, assumed to be in valid range.

**Return value** - The label.

```
array<uint8_t, PIXELS> get_an_image(const enum data_type mode, const size_t index) const;
```

**Description** - Return an image from `train_images` OR `test_images` at a specific position.

**Parameters**

- `mode` - To indicate the dataset spilt to get an image, assumed to be either `TRAIN_IMAGE` OR `TEST_IMAGE`.
- `index` - The position at which the image is retrieved, assumed to be in valid range.

**Return value** - The image.

```
vector<float> get_a_normalized_image(const enum data_type mode, const size_t index) const;
```

**Description** - Return an normalized image from `train_images` OR `test_images` at a specific position. Unlike an original image having integral pixel value in the range [0, 255], a normalized image has decimal pixel value in the range [0, 1].

**Parameters**

- `mode` - To indicate the dataset spilt to get an image, assumed to be either `TRAIN_IMAGE` OR `TEST_IMAGE`.
- `index` - The position at which the image is retrieved, assumed to be in valid range.

**Return value** - The image.

**Notes** - By default, an image should be normalized before being fed into a neural network.

```
size_t get_data_size(const enum data_type mode) const;
```

**Description** - Return the data size.

**Parameters**

- `mode` - To indicate the dataset spilt to return the size, can be `TRAIN_LABEL`, `TEST_LABEL`, `TRAIN_IMAGE` OR `TEST_IMAGE`.

**Return value** - The size.

## Choose suitable STL containers for image manipulation

Though the images are stored as an STL array after loading the dataset, they may not be ready for image augmentation, such as shifting, resizing, cropping and padding. Becuase the STL array length has to be set as compile-time constant. Thus, before applying these operations, an image in the STL array format need to be converted into other format with proper STL containers.

**Note that you have to define a new data type with name `img_dt` using typedef definitions in `dataset.h`,** before implementing some of the following functions involving `img_dt` usages.

Though this `img_dt` definition is not graded directly, choosing STL containers wisely and defining it properly will ease your implementaion on related functions. Here are some general rules on choosing the right STL containers (particularly the sequential containers):

- Use `std::vector` as your default sequential container, especially as an alternative to built-in arrays.
- If size is known in advance, use `std::array` instead of a built-in array.

- If you add or remove elements frequently at both the front and back of a container, use `std::deque`.
- Use a `std::list` (not `std::deque`) if you need to insert/remove elements in the middle of the sequence
- Do not use `std::list` if you need random access to objects.
- You can define multi-dimensional containers by combining multiple same or different containers in a nested manner.

As the purpose of PA10 is to familiarize yourself with various STL templates, we strongly encourage you to walk through the following related documents, be aware of various off-the-shell operations, and figure out ideal choices that fit in this assignment. Here are some related links for your reference:

- Containers - <https://cplusplus.com/reference/stl/>, <https://www.geeksforgeeks.org/containers-cpp-stl/>
- Algorithm - <https://cplusplus.com/reference/algorithm/>
- Iterator - <https://cplusplus.com/reference/iterator/>

End of Description

## Tasks

Implement the following functions in the `dataset.cpp` except for the function `void testing(NN& nn, dataset& data)` in the `train_test.cpp`. Note that you have to define a new data type with name `img_dt` using typedef definitions in `dataset.h` for related functions.

```
void print_statistic(const enum data_type mode) const;
```

**Description** - Calculate how many images are there for the train and test split and print the result.

Example output:

```
Number of train images for digit 0: 1994
Number of train images for digit 1: 2281
Number of train images for digit 2: 1929
Number of train images for digit 3: 2076
Number of train images for digit 4: 1945
Number of train images for digit 5: 1775
Number of train images for digit 6: 1971
Number of train images for digit 7: 2093
Number of train images for digit 8: 1922
Number of train images for digit 9: 2014
```

Another example output:

```
Number of test images for digit 0: 305
Number of test images for digit 1: 378
Number of test images for digit 2: 352
Number of test images for digit 3: 349
Number of test images for digit 4: 352
Number of test images for digit 5: 307
Number of test images for digit 6: 309
Number of test images for digit 7: 347
Number of test images for digit 8: 319
Number of test images for digit 9: 315
```

### Parameters

- `mode` - specify the dataset split.

### Notes

- There's one space after each colon regardless of the specific number.



- You can assume the parameter `mode` to be either `TRAIN_IMAGE` OR `TEST_IMAGE` only.
- It's ok to have trailing spaces at the end of each row and new empty rows at the end of the output.

```
void testing(NN& nn, dataset& data);
```

**Description** - The testing phase to test the accuray of the trained neural network. The **normalized** images of the test split will be fed through the network one by one to get the output. The output is a vector<float> with length 10. The index of the maximum element is called the prediction, which is then further compared with the corresponding label to check whether the prediction is correct. By iterating through all images, and do the comparisons, we can calculate and print the overall and digit-wise accuracy.

Example output:

```
Overall accuracy: 78.94%
Accuracy for digit 0: 95.41%
Accuracy for digit 1: 96.30%
Accuracy for digit 2: 82.10%
Accuracy for digit 3: 84.24%
Accuracy for digit 4: 84.94%
Accuracy for digit 5: 48.21%
Accuracy for digit 6: 79.94%
Accuracy for digit 7: 79.83%
Accuracy for digit 8: 62.07%
Accuracy for digit 9: 71.11%
```

#### Parameters

- `nn` - the trained neural network.
- `data` - the dataset object for reference.

#### Notes

- The accuracy values are round to two decimal places. For example, 95.405% should become 95.41% and 95.404% should become 95.40%.
- It's ok to have trailing spaces at the end of each row and new empty rows at the end of the output.
- Please assume that each digit has at least one occurrence.

```
void remove_some_train_data(const uint8_t digit, const size_t size);
```

**Description** - Remove the last `size` data of `digit` from the `train_labels` and `train_images` vectors while keeping other data in their original order.

#### Parameters

- `digit` - the digit to remove.
- `size` - the number of data to remove.

#### Notes

- You can assume the parameter `digit` to be a valid integer in range [0, 9] and the `size` to be less than the number of that digit avaiable in the dataset.

```
void print_image(const array<uint8_t, PIXELS>& img_ar) const;
```

**Description** - Visualize an image by print each pixel's integer value. It serves as a helper function to check if your following image manipulation implementaion works.

Example output of a digit 9:

## Parameters

- ## Notes

- 7/19

```
void print_image(const img_dt& img) const;
```

**Description** - Same to the above print\_image function but with different image's data type.

**Parameters**

- img - the image to visualize.

**Notes**

- Refer to the above print\_image function.

```
img_dt convert_image_dt(const array<uint8_t, PIXELS>& img_ar) const;
```

**Description** - Convert an image in array type to img\_dt type.

**Parameters**

- img\_ar - the image to convert.

**Return** - the converted image.

**Notes**

- You can assume the image to have a fixed size of 28 x 28 for this function.

```
array<uint8_t, PIXELS> convert_image_dt(const img_dt& img) const;
```

**Description** - Convert an image in img\_dt type to array type.

**Parameters**

- img\_ar - the image to convert.

**Return** - the converted image.

**Notes**

- You can assume the image to have a fixed size of 28 x 28 for this function.

```
void shift_image(img_dt& img, const enum shift_type dir, const size_t p, const uint8_t value=
```



**Description** - Shift an image horizontally, vertically or diagonally by the specified number of pixels. The empty region after shifting are filled with the specified value.

If we print the shifted image on the above untouched digit 9 with `dir = LEFT`, `p = 7`, we have the output:

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	55	148	210	253	253	113	87	148	55	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	87	232	252	253	189	210	252	252	253	168	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	4	57	242	252	190	65	5	12	182	252	253	116	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	96	252	252	183	14	0	0	92	252	252	225	21	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	132	253	252	146	14	0	0	0	215	252	252	79	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	126	253	247	176	9	0	0	8	78	245	253	129	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	232	252	176	0	0	0	36	201	252	252	169	11	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	252	252	30	22	119	197	241	253	252	251	77	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	231	252	253	252	252	252	226	227	252	231	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	55	235	253	217	138	42	24	192	252	143	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	62	255	253	109	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	71	253	252	21	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	253	252	21	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	71	253	252	21	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	106	253	252	21	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	45	255	253	21	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	218	252	56	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	96	252	189	42	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	14	184	252	170	11	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	14	147	252	42	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0									

Parameters

- `img` - the image to shift.
- `dir` - the direction to shift. There are eight directions in total.
- `p` - the number of pixels to shift.

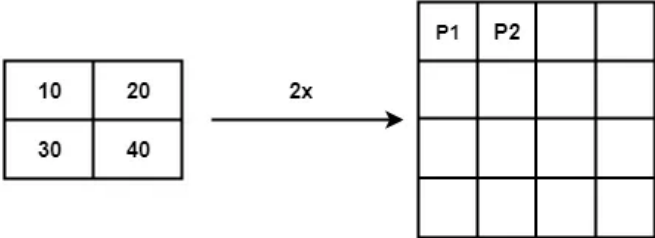
- value - the value to fill the empty region.

Notes

- p is non-negative. If it exceed the image width and height, the entile image will be filled with the specified value.

```
void resize_image(img_dt& img, const size_t new_rows, const size_t new_cols);
```

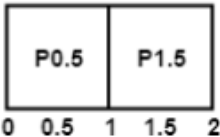
**Description** - Resizing an image using the nearest neighbour method. Let's see how this works. Suppose we have a 2x2 image and we want to resize it to 4x4 as shown below.



Let's pick up the first pixel (denoted by 'P1') in the resized image. To assign it a value, we must find its nearest pixel in the input 2x2 image. Let's first see some facts and assumptions used in this.

Assumption: a pixel is always represented by its center value. Each pixel in our input 2x2 image is of unit length and width.

For sake of simplicity, we start indexing from 0.5 which means that our first pixel is at 0.5 next at 1.5 and so on as shown below.

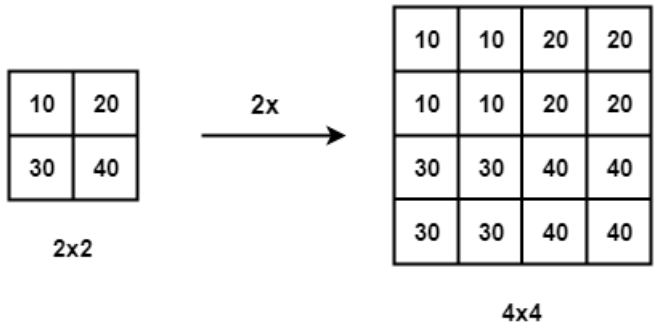


So for the above example, the location of each pixel in inut image is {'10': (0.5, 0.5), '20': (1.5, 0.5), '30': (0.5, 1.5), '40': (1.5, 1.5)}.

After finding the location of each pixel in the input image, follow these 2 steps:

1. First, find the position of each pixel (of the resized image) in the input image. This is done by projecting the 4x4 image on the 2x2 image. SO, we can easily find out the coordinates of each resized image pixel, e.g., location of 'P1' in the input image is (0.25, 0.25), for 'P2' (0.75, 0.25) and so on.
2. Now, compare the above-calculated coordiantes of each unkown pixel with the input image pixels to find out the nearest pixel, e.g., 'P1' (0.25, 0.25) is nearest to 10 (0.5, 0.5). So we assign 'P1' value of 10. Similarly, for other pixels, we can find their nearest pixel.

The final result we get is shown in figure below:



Resizing to smaller images follow similar protocol.

If we print the resized image on the above untouched digit 9 with new\_rows = 32 and new\_cols = 32, we have the output:

[illegible]

If we print the resized image on the above untouched digit 9 with `new_rows = 18` and `new_cols = 18`, we have the output:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	148	210	253	113	148	0	0	0	0	0
0	0	0	0	0	0	0	87	252	253	210	252	253	0	0	0	0	0
0	0	0	0	0	0	252	252	14	0	92	252	225	0	0	0	0	0
0	0	0	0	0	132	252	146	0	0	215	252	79	0	0	0	0	0
0	0	0	0	232	252	0	0	36	201	252	169	0	0	0	0	0	0
0	0	0	0	252	252	22	119	241	253	251	77	0	0	0	0	0	0
0	0	0	0	55	235	217	138	24	192	143	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	62	255	109	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	253	21	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	106	253	21	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	45	255	21	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	96	189	42	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	14	252	170	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Parameters

- `img` - the image to resize.
- `new_rows` - the rows or height of the resized image.
- `new_cols` - the columns or width of the resized image.

Notes

- You can assume that the `new_rows` and `new_cols` are positive integers.
- ~~Please choose the top, left, or top left one when there is more than one pixel having the shortest distance.~~
- Dealing with such boundary cases are not our primary intent. For ease of the implemation, please choose the bottom, right, or bottom right one when there is more than one pixel having the shortest distance. For example, if there are exactly two pixels having the same shortest distance, choose the bottom (right) one if they are vetically (horizontally) adjacent nerghbors; if there are four pixels having the same shortest distance, choose the bottom right one.

```
void crop_image(img_dt& img, const size_t y0, const size_t x0, const size_t new_rows, const size_t new_cols)
```

**Description** - Crop the image to a patch with specified dimensions. The coordinates specify the top-left pixel of the cropped patch.

If we print the cropped image on the above untouched digit 9 with  $y_0 = 1$ ,  $x_0 = 1$ ,  $\text{new\_rows} = 24$  and  $\text{new\_cols} = 24$ , we have the output:

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	55	148	210	253	253	113	87
148	55	0	0	0	0	0												
	0	0	0	0	0	0	0	0	0	0	0	87	232	252	253	189	210	252
253	168	0	0	0	0	0												
	0	0	0	0	0	0	0	0	4	57	242	252	190	65	5	12	182	252
253	116	0	0	0	0	0												
	0	0	0	0	0	0	0	0	96	252	252	183	14	0	0	92	252	252
225	21	0	0	0	0	0												
	0	0	0	0	0	0	0	132	253	252	146	14	0	0	0	215	252	252
79	0	0	0	0	0	0												
	0	0	0	0	0	0	126	253	247	176	9	0	0	8	78	245	253	129
0	0	0	0	0	0	0												
	0	0	0	0	0	16	232	252	176	0	0	0	36	201	252	252	169	11

```
void padding_image(img_dt& img, const size_t top=0, const size_t down=0, const size_t left=0,
```

**Description** - Pad specified value to the borders of the image.  
If we print the padded image on the above cropped digit 9 with top=1, down=2, left=2, right=1, value=1, we have the output:



	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	55	148	210	253	253
113	87	148	55		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	87	232	252	253	189	210
252	252	253	168		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	4	57	242	252	190	65	5	12
182	252	253	116		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	96	252	252	183	14	0	0	92
252	252	225	21		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	132	253	252	146	14	0	0	0	0	215
252	252	79	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	126	253	247	176	9	0	0	8	78	245	
253	129		0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	16	232	252	176		0	0	36	201	252	252
169	11		0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	22	252	252	30	22	119	197	241	253	252	251
77	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	16	231	252	253	252	252	252	226	227	252	231
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	55	235	253	217	138	42	24	192	252	143	
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	62	255	253	109
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	71	253	252	21
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	253	252	21
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	71	253	252	21
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	106	253	252	21
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	45	255	253	21
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	218	252	56
0	0	0	0		0	0	0	0	1										
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96	252	189
42	0	0	0		0	0	0	0	1										
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1										
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1										

Parameters

- `img` - the image to pad.
- `top` - how many rows to pad on the top.
- `down` - how many rows to pad on the bottom.
- `left` - how many rows to pad on the left.
- `right` - how many rows to pad on the right.

Resources & Sample I/O

- Skeleton code: [PA10\\_skeleton.zip](#)  
You are suggested to compile the program using our provided `Makefile`.
- Sample output file: you should get the output as in the [demo output 1](#) (results writed to file) or the output as in the [demo output 2](#) (results printed on the terminal) after finishing all related tasks and run the `main.cpp` we provided without any unintended modification. The exact accuracy might **fluctuate a bit** due to the inconsistency of random number on different platform.

End of Resources & Sample I/O

## Submission & Grading

**Deadline: 19 November 2022 Saturday HKT 23:59.**

Submit a zip file to [ZINC](#) (The ZINC submisssion portal will be available soon). Compress the source files `dataset.h`, `dataset.cpp` and `train_test.cpp` as `PA10.zip` directly, **not a folder containing them**.

**Late Policy:**

Please refer to the [3-day late budget policy](#).

You are strictly forbidden to modify the signature of given classes and functions. However, you may overload the functions. When we grade your program, if we cannot find the exact function whose signature is what we described in [Tasks section](#), you will have **no score** for **all related tasks**.

Before deadline, the test cases on ZINC will only contain a few cases. If you passed those tests, it only means your program can successfully run on ZINC and can pass those test cases. The result is by no mean complete and the score are irrelavent to your actual score. In actual grading stage after deadline, we will use a **totally different set of test cases**, which is expected to be more complete and more strict, to test the correctness your program.

Please ensure that you submit to ZINC well before the deadline as **all late submissions will be automatically rejected**.

## Grading Scheme

In this assignment, we follow the Unit Test manner to grade your assignment on ZINC. Some test cases will test single behavior of a task class. However, some tasks may depend on correct implementation of several other functions, e.g., to test the `void shift_image(img_dt& img, const enum shift_type dir, const size_t p, const uint8_t value=0);` function, we would call the `convert_image_dt(const array<uint8_t, PIXELS>& img_ar);` at first and the `print_image(const img_dt& img);` at last. So, a erroneus implementation of the related helper functions can affect the grading on target functions. The final grade of this assignment will be based on how many test cases you have passed, and no partial points will be given to each test cases. So, please be careful implementing every task class and test your code thoroughly before the ddl.

## Memory Leak

Please bear in mind that we will also check memory leak in test cases. If any of your test cases have memory leak, you will receive a 10% penalty. This penalty will be applied only once in this PA.

End of Submission & Grading

## Change Log

**1. 20:02 03/11/2022**

Update the skeleton code by adding `#include <numeric>` to the `train_test.cpp` file.

**2. 02:13 04/11/2022**

- Update the skeleton code by adding `#include <fstream>` to the `nn.h` file.
- Add notes to the `convert_image_dt` functions, where you can assume the image to has a fixed size of 28 x 28.

- Update the example output of `void testing(NN& nn, dataset& data);`, the demo\_output1 and demo\_output2 by removing the output for the so-called "digit 10".

3. **12:50 04/11/2022**

Update the skeleton code by replacing the statement

```
this->padding_image(img, (ROWS-rand_num)/2, ROWS-(ROWS-rand_num)/2, (COLS-rand_num)/2, COLS-(COLS-rand_num)/2);
```

in the `compound_augment_image` function with

```
this->padding_image(img, (ROWS-rand_num)/2, ROWS-(ROWS-rand_num)/2-rand_num, (COLS-rand_num)/2, COLS-(COLS-rand_num)/2-rand_num);
```

4. ~~14:22 04/11/2022~~

Add notes to the `resize_image` functions:-

~~Please choose the top, left, or top left one when there is more than one pixel having the shortest distance.~~

5. **20:50 04/11/2022**

Add notes to the `resize_image` functions:

Dealing with such boundary cases are not our primary intent. For ease of the implemation, please choose the bottom, right, or bottom right one when there is more than one pixel having the shortest distance.

6. **15:20 07/11/2022**

Add notes to the `testing` functions:

Please assume that each digit has at least one occurrence.

7. **23:22 14/11/2022**

Add further notes to the `resize_image` functions:

For example, if there are exactly two pixels having the same shortest distance, choose the bottom (right) one if they are vetically (horizontally) adjacent nerghbors; if there are four pixels having the same shortest distance, choose the bottom right one.

8. **10:45 18/11/2022**

Add further notes about the accuracy rounding in the `void testing(NN& nn, dataset& data);` function:

For example, 95.405% should become 95.41% and 95.404% should become 95.40%.

9. **12:18 20/11/2022**

Update the part3 accuracies in the demo output 1 and the demo output 2. This won't affect the tasks and requirement of your implementations.

End of Change Log

# Frequently Asked Questions

- Q:** My code doesn't work, here it is, can you help me fix it?  
**A:** As the assignment is a major course assessment, to be fair, you are supposed to work on it by yourself and we should never finish the tasks for you. We are happy to help with explanations and advice, but we are **not allowed** to directly debug for you.
- Q:** Can we include extra libraries? For example, for printf, or for output formatting?  
**A:** Yes, you can include those extra libraries. But remember to **put them under the files that are submitted for grading.** .
- Q:** Which function to fed the image into the network?  
**A:** Use the operator () of the NN class. For example:

```
void testing(NN& nn, dataset& data){  
    vector input = data.get_a_normalized_image(TEST_IMAGE, 0);  
    vector output = nn(input);  
}
```

...

End of FAQ



Maintained by COMP 2012H Teaching Team © 2022 HKUST Computer Science and Engineering