**MNIST DIGIT SEQUENCE GENERATOR**

# **Contents:**

1. Introduction
2. Method
3. Implementation details
4. Experiment and results
5. Conclusion

# **Introduction:**

This report documents a model that generates the image of a sequence of digits. This digit sequence, range of spacing between the digits and also total image width are specified by the user. This kind of sequence images can be used for data augmentation purposes

# **Method used:**

Idea is to horizontally stack individual digit images of corresponding user specified digit in sequence along with a spacing which is coming from a uniform distribution in user specified range. The user specified image width restricts the available width for random spacing. If available space is less than probable random spacing, then random spacing is not possible. In these cases, we go for uniform spacing.

Let “digit\_width” represent the MNIST digit image width. Let “l” represent the sequence length and “image\_width” be the total width. Let “min\_space” and “max\_space” represent the user specified spacing range.

Available\_space = image\_width – L\*digit\_width

Then there could be 3 possible cases:

1. Available\_space<(L-1)\*min\_space:
2. (L-1)\*min\_space<=Available\_space<(L-1)\*max\_space
3. Available\_space>(L-1)\*max\_space

**CASE 1**: Available space for gap is not sufficient to even accommodate minimum desired space. Hence exception is encountered here.

**CASE 2**: Available\_space is in between the min and max range. This cannot guarantee that random spacing can be accommodated as we could end up with all max\_spaces. This leads to exception. Hence this case is handled by providing same spacing between all digits and this “uniform\_space” is given by:

Uniform\_space = floor(Available\_space/(L-1))

Floor function is used to convert to integer as the unit is number of pixels. There is a chance of total width not matching the user given value due to the conversion to integer. This is taken care by appending a gap corresponding to leftover pixels on left and right side of the sequence image.

**CASE 3**: Available space is sufficient to have a random spacing. The gap between digits follow a uniform distribution with range specified by user. As the canvas is fixed, the remaining pixels are accommodated as gap on left and right side of the sequence image to get desired width.

Hence the generated image is finally saved as “.png” file in the same folder.

# **Implementation:**

The code is written in object oriented approach in python in order to provide flexibility for further changes. A python API is provided along with a CLI script which takes command line input. Details of script usage are specified in the README files.

Generate() function takes the input as digit\_sequence, spacing\_range and image\_width and outputs the sequence image. It makes use of 2 other funtions: get\_image\_for\_digit() which randomly choses and returns an MNIST digit image for the given digit and spacing\_check() which gets the spacing mode as per above described logic and updates the gap\_array. The flow for these three functions are provided below:

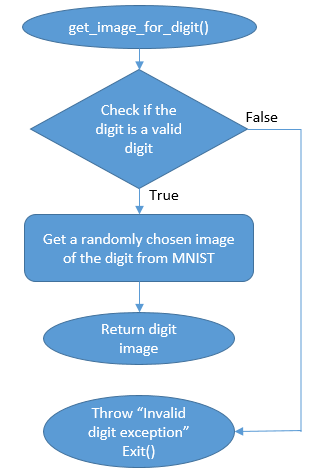


Figure1: Flow diagram of get\_image\_for\_digit()

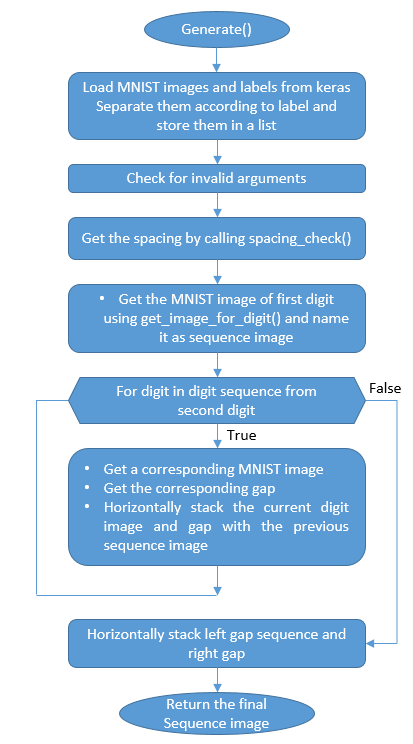


Figure2: Flow diagram of generate()

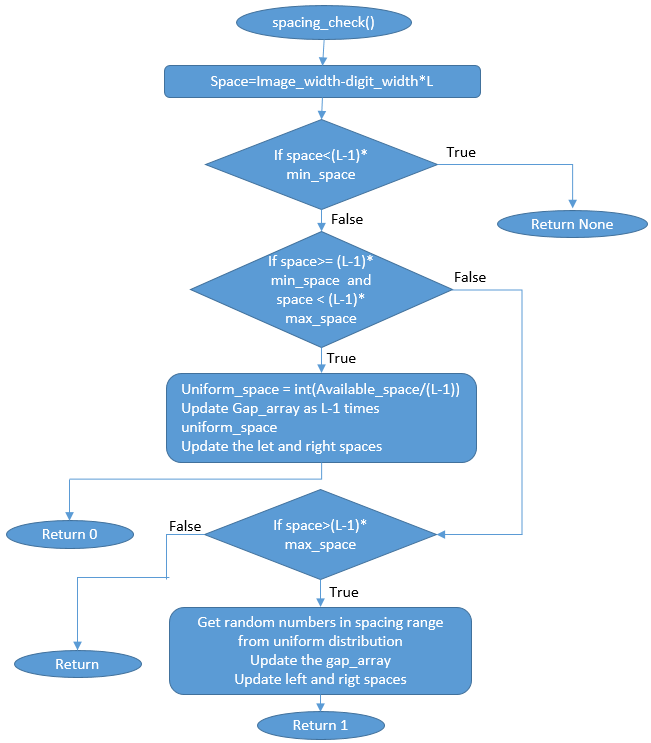


Figure3: Flow diagram of spacing\_check()

**TESTS:**

This code is tested for the following 10 different scenarios:

1. Loading of MNIST images

-Check images height, width and number of classes

2. Sequence image generation

-Check image height

-Check image width (in specified range)

-Check datatype of pixels

-Check pixel max and min value

3. Single digit sequence test

-Check height and width of image

-Check datatype, min and max value of pixel

4. Invalid sequence test

-Check if invalid sequence input generates "None" as output

5. Invalid spacing range test

-Check if invalid spacing range generates "None" as output

6. Empty sequence test

-Check if empty sequence as input generates "None" as output

7. Saving the generated image

-Check if image is saved properly

8. Checking random spacing mode

-Check if random spacing mode is selected appropriately

-Check if generated image is of desired dimension

9. Checking uniform spacing mode

-Check if uniform spacing mode is selected appropriately

-Check if generated image is of desired dimension

10. Checking invalid spacing range and total width combination

-Check if invalid spacing range and total width combination is appropriately handled

# **Experiment and results:**

Experiments were conducted to generate digit sequences of varied combination of sequence, spacing range and total width and test cases were executed. The code is found to pass the tests specified in previous section.

The following images are a few examples for uniform and random spacing:



Figure4: Example for random spacing



Figure5: Example for uniform spacing

# **Conclusion:**

The sequence image generator using MNIST images is implemented with 2 spacing modes.

The code is written in object-oriented approach along with API – CLI scripts to present it in productized format. Exceptions handling is used to take care of error situations during runtime. Logging is provided which logs the important events like exceptions and some decisions in the code.

**Further additions:** Configparser could be implemented to handle the configuration of the code.

MNIST data is imported from KERAS library. A script could be added to obtain MNIST data directly from the official website.