Dimension Reduction by Ring Projection Pipeline Module

Bui Nhat Quang

***Abstract*—The size-orientation-invariance aspect contributes an essential part in pattern recognition. It has various applications in computer vision, character recognition. Ring-Projected algorithm is introduced to tackle the rotation variance problem. This algorithm is implemented as pipeline module of LearningAPI - Machine Learning libraries running on top of .NET Core. By projecting binarized contents of the image to circles, the output function is rotation-invariant. Furthermore, Unit test is conducted to verify this characteristic of the algorithm. This will simplify the recognition process which it can treat all the different-orientation characters as resemble characters. Also, since the data size is reduced, the process becomes faster. This algorithm can be expanded to object recognition rather than only character recognition.**

Keywords—pattern recognition, Ring-Projected Algorithm, rotation-invariant, LearningAPI, .NET Core.

I. INTRODUCTION

Ring Projection is a dimensional reduction method used in image recognition. The method converts two dimensional image into one dimensional function. This method is implemented as part of LearningAPI Machine Learning Foundation running on top of .NET Core [1]. Beside implementing `RingProjectionPipelineModule`, there are others pipeline modules that are also implemented for reporting the result as CSV file and to carry out the algorithm evaluation with certain test images and MNIST dataset of handwritten digits. The rest of the report is organized as follows. Section II describes the algorithm in details and presents the implementation alongside with the unit test. The results are presented and discussed in Section III. Finally, the conclusion gives a summary and future development of Ring-Projected algorithm as part of LearningAPI.

II. METHODS

A. Ring-Projected Algorithm

It is required to centralize the 2D-pattern to the image by its center of mass, normalize the size, and convert to binary before applying the Ring-Projected algorithm. Suppose a two dimensional pattern such as an alphanumeric symbol has been represented in a binary image where the origin of the Cartesian coordinate is placed at the center of mass of the pattern. The image can be represented as follow:

$$p(x,y) = \begin{cases} 1 \text{ if } (x,y) \in D \\ 0 \quad \text{otherwise} \end{cases}$$

where domain D corresponds to the black region of the letter. This function can be viewed as a two dimensional density function of mass distribution on the plane. Since the pattern is already binarized, the corresponding density function is uniform distribution. Using polar coordinate,

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases} \text{ where } r \geq 0 \text{ and } 0 \leq \theta \leq 2\pi$$

the density function can be given as

$$p(x,y) = p(r\cos\theta, r\sin\theta)$$

For an arbitrary radius r we calculate the integral

$$f(r) = \int_0^{2\pi} p(r\cos\theta, r\sin\theta)d\theta$$

The resulting $f(r)$ f(r) is one dimensional and only depends on the radius r [2]. An example of the function is illustrated in Fig. 1.

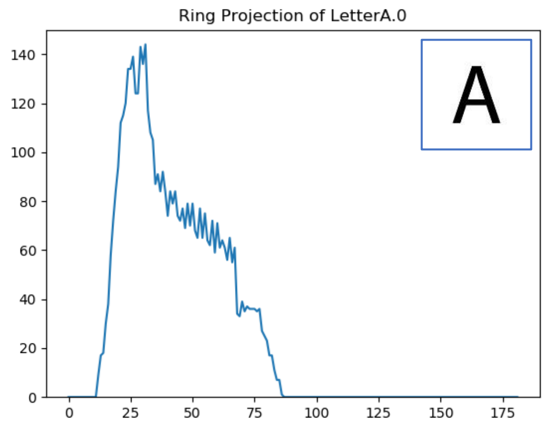
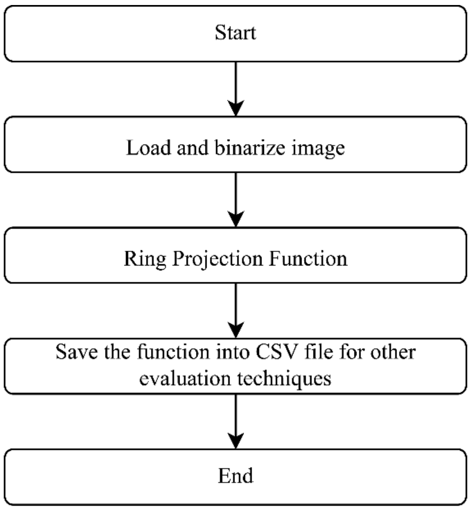


Fig. 1. Ring-Projected function of letter A

B. Architecture

The program is divided into two main projects: `RingProjectionAlgorithm` and `UnitTest-RingProjectionAlgorithm`. The implementation of the algorithm is created inside the first project and the second one is for testing the algorithm by feeding data into the pipeline in which the algorithm is injected and present the result. The project is created based on .NET Core 2.0 using C# programming language.


Start
Load and binarize image
Ring Projection Function
Save the function into CSV file for other evaluation techniques
End
Fig. 2. Architecture of the project

The overview of the project is shown in Fig. 2. First, a centered image will be loaded into the program, binarized and transformed to a suitable form for the algorithm to work with. After the projection, the 1D data will be stored as a CSV file for further data manipulation. An example of setting up and using the algorithm is illustrated in Fig. 3.

```
LearningApi api = new LearningApi();
api.UseActionModule<object, double[][]>((input, ctx) =>
{
    Binarizer biImg = new Binarizer();
    double[][] data = biImg.GetBinaryArray(
        Path.Combine(trainingImagesPath, testImages[i]), 50);
    return data;
});
api.UseRingProjectionPipelineComponent();
api.AddModule(new RingProjectionFunctionToCSVPipelineModule(
     "LetterA", i, ";", trainingImagesPath));
api.Run();
```

Fig. 3. Code snippet of setting up and using Ring-Projected algorithm

C. *Implemetation*

1) *Ring Projection Pipeline Component*

The algorithm is required to be implemented as a pipeline module of LearningAPI. LearningAPI can run in the pipeline of modules compatible to each other as the output TOUT of the previous module is the input TIN of the next module [1]. These individual modules should implement `IPipelineModule` interface (Fig. 4) to be able to run as LearningAPI modules.

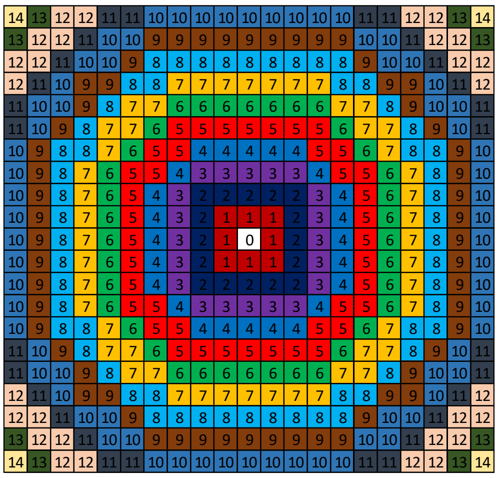
```
public interface IPipelineModule<TIN, TOUT> : IPipelineModule
{
    TOUT Run(TIN data, IContext ctx);
}
```

Fig. 4. Code Snippet of interface `IPipeLineModule`

Base on the concept, the implementation of the algorithm is quite similar. It should take into account all the pixels that are on the same ring from the center of the image. However, instead of calculating the trigonometric function with infinite points between 0 and 2π, we can calculate the distance d for every single point (x, y) from the center $O(x_0, y_0)$

$$d = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

and compare the result with the radius of the ring. From that, the value of each point from the Ring-Projected Function $f(r)$ will be calculated. Besides, using this method makes sure that every pixel of the image is considered. Fig. 5 shows an example of Ring-Projected Algorithm iteration over a 2D array with size 21x21 where the number represent the radius value. Every pixel with the same radius that is marked with same color belong to one circle. In order to get the circumcircle with maximum radius, the distance from one of the corner pixels of the image to its center is computed beforehand.



14	13	12	12	11	11	10	10	10	10	10	10	10	10	10	11	11	12	12	13	14
13	12	12	11	10	10	9	9	9	9	9	9	9	9	9	10	10	11	12	12	13
12	12	11	10	10	9	8	8	8	8	8	8	8	8	8	9	10	10	11	12	12
12	11	10	9	9	8	8	7	7	7	7	7	7	7	8	8	9	9	10	11	12
11	10	10	9	8	7	7	6	6	6	6	6	6	6	7	7	8	9	10	10	11
11	10	9	8	7	7	6	5	5	5	5	5	5	5	6	7	7	8	9	10	11
10	9	8	8	7	6	5	5	4	4	4	4	4	5	5	6	7	8	8	9	10
10	9	8	7	6	5	5	4	3	3	3	3	3	4	5	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	2	2	2	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	1	1	1	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	1	1	1	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	2	2	2	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	5	4	3	3	3	3	3	4	5	5	6	7	8	9	10
10	9	8	8	7	6	5	5	4	4	4	4	4	5	5	6	7	8	8	9	10
11	10	9	8	7	7	6	5	5	5	5	5	5	5	6	7	7	8	9	10	11
11	10	10	9	8	7	7	6	6	6	6	6	6	6	7	7	8	9	10	10	11
12	11	10	9	9	8	8	7	7	7	7	7	7	7	8	8	9	9	10	11	12
12	12	11	10	10	9	8	8	8	8	8	8	8	8	8	9	10	10	11	12	12
13	12	12	11	10	10	9	9	9	9	9	9	9	9	9	10	10	11	12	12	13
14	13	12	12	11	11	10	10	10	10	10	10	10	10	10	11	11	12	12	13	14

Fig. 5. Loop iteration of Ring-Projected Algorithm

The algorithm receives `double[][]` array, which holds the coordinates of recognizing pattern, as input and produce `double[]` array as output. To run Ring Projection pipeline component as part of LearningAPI, we can use method `AddModule()` and pass in an instance of the class which inherit `IPipelineModule` or make extension method which also do the same thing. Extension method is preferred due to simplicity and code suggestion of Visual Studio.

2) *Export to CSV File*

The result from the Ring-Projected algorithm is an 1D function. In order to present the result in different format, it is required to be save as a csv file. A pipeline module is created to handle this exportation. The module generates file on hard drive, therefore, it does not return value or it returns null. Only a universal type which accept null value without changing the meaning of the output of the module is object. A function returns a null object means it returns nothing. The output CSV file consists of two columns: Radius r and value $f(r)$. The first line is column names. The data is shown from the second line. For different country, a delimiter must be specified as a parameter in the pipeline module.

3) *ImageBinarizer*

This algorithm required binarized input photo. There exists ImageBinarizer project in LearningAPI repository. However, this tool only can give the output as string to display to console window or as a text file. It lacks of converting image into `double[][]` which is necessary for the compatibility of Ring Projection Pipeline Component. Besides, ImageBinarizer does not include the capability of setting threshold for binarization and giving invert binary image with the background is ‘0’ and the pattern region is ‘1’. Therefore, it is needed to write extension methods for ImageBinarizer. There are two extension methods that give the output in `double[][]` while receiving different input data type as show in Fig. 6.

```
public static double[][] GetBinaryArray(
    this Binarizer bi, string image,
    out Bitmap img, int threshold)...

public static double[][] ConvertToBinary(
    this Binarizer bi, double[][] mnistImage, int threshold)...
```

Fig. 6. Code Snippet from `BinarizerExtension`

The implementation of extension method rather simple. For `GetBinaryArray()`, which converts from an image file acquired by the its path, if all color parameters of a pixel in the source image are less than threshold, the corresponding pixel of the result is set to 1, otherwise, it is 0. For `ConvertToBinary()` method, which receives `double[][]` directly in inverted grayscale, the conversion is contrary. The significant pixel, which is greater than the given threshold results in value of 1 for the corresponding pixel of the output array whereas the value less than threshold will be set to 0.

D. UnitTest

1) Iteration path

The algorithm should go through `double[][]` array calculate the sum of all points which lie on the same circle. Therefore, it is needed to check whether the algorithm meets the requirement. Also, a stopwatch is set up to compare the elapsed time of this iteration and normal iteration through rows and columns.

2) Rotation-invariance of Ring-Projected Algorithm

A set of test images is given in TestImages folder. The set contains images of letter A, C, G and O in various orientations. All images are assumed to be centered by their own center of mass. The test is setup as follow: a pipeline is generated for each test image, each pipeline consists of binarizer, projection algorithm and save to CSV modules. In order to give data into the pipeline, an action module is used to implement ImageBinarizer as it is not part of LearningAPI NuGet packet (See Fig. 7).

```
api = new LearningApi();
api.UseActionModule<object, double[][]>((input, ctx) =>
{
    Binarizer biImg = new Binarizer();
    double[][] data = biImg.GetBinaryArray(
        Path.Combine(trainingImagesPath, testImages[i]), 50);
    return data;
});
```

Fig. 7. Code Snippet of ImageBinarizer in action module

An action module defines certain functions to be executed when the pipeline is running [1]. Normally, an action module defines custom code to give initial input to the pipeline. This action module does not receive any input. However, it is not allowed to specify the input type as void since void means the previous module does not return any value which is contradict to the definition of Generics in C#. Hence the input type is set to object and C# will assume the input is null by default. This module handles the loading and binarizing process and produce `double[][]` array which is compatible with ring projection module. The third module of the pipeline saves the output function as a CSV file on the hard disk. The file is located in the same folder as the test images. After running through all test images, cross correlation coefficients are calculated among these results to check whether the rotation-invariant property is hold true.

3) Ring Projection with MNIST Dataset

For testing the Ring-Projected algorithm, the MNIST training set is used. MNIST dataset has a training set of 60,000 examples of handwritten digits, and test set of 10,000 examples, written by 500 writers. The digits have been size-normalized and centered in a fixed-size image. This dataset has been used for training and testing by many well-known classified methods [3].

In order to read the dataset, a simple pipeline module is written for LearningAPI including class definition of MNIST image. The dataset consists of separated label file and image file.

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label

The labels values are 0 to 9.
```

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Fig. 8. MNIST Files description [3]

According to Fig. 8, the dataset files contains overhead information in 32-bit integer. These value are stored in MSB first (high endian) format used by non-Intel processors. Therefore, for common computer nowadays, these number must be flipped into low endian format.

```
private int ReverseBytes(int number)
{
    byte[] m_temp = BitConverter.GetBytes(number);
    Array.Reverse(m_temp);
    return BitConverter.ToInt32(m_temp);
}
```

Fig. 9. Convert high-endian format to low-endian format

Fig. 9 shows code snippet of the converting process which rearranges the bits of 32-bit integer. For example, number 50 in low-endian format is 00000000 00000000 00000000 01001100 whereas in high-endian format, it is represented by 01001100 00000000 00000000 00000000 corresponding to number 1,275,068,416. Basically, the byte positions are in reversed order as the first byte is swapped with the forth byte and second byte is swapped with the third byte. After getting all information of MNIST dataset, the pixels and labels are read accordingly as MNIST images. An MNIST image consists of image size, pixels, and actual label. The module returns the whole set of MNIST images and stores them in an array.

To process the set, two different pipelines are created for each MNIST image.

The first one visualizes the MNIST image as an actual image file. For exporting MNIST image to image file, two pipeline components are needed. The first one is to convert `double[][]` into Bitmap type and the second one transforms Bitmap into real picture file on the hard drive. It is obvious that we can get image file straight from `double[][]`, however, it is better to split the process into two modules for future code reuse.

The second pipeline is responsible to run Ring Projection on the MNIST image and save the result as a CSV file same as seen in II.D.2).

The whole dataset is sorted by label into 10 different digits and stored in corresponding folders. After that, the CSV files are loaded back to the test. Then the correlation among images with the same label are calculated and the result is given in statistical format (e.g. including average, minimum, maximum and standard deviation) and another CSV file is produced. The file contains statistical information of the cross-correlation coefficient of each digit.

III. Results

According to the unit test, the results is illustrated as follows.

A. Iteration path

The iteration path can be illustrated in Fig. 5 which shows that for a small radius, in digital world, a circle is the same as a square. When the radius increases, the number of points/pixels to describe the circle increases, result in a smooth circle. The CSV file generated from the unit test is processed by Excel by simply highlighting the cells which have the same number with the same color. In order for the file to be compatible with Excel, the line "`sep=;`" is included at the top of the file. The line defines the separator which separates data values in the CSV file.

By testing the ring iteration on 21x21 array, the result in Fig. 10 shows a significant higher time consumption against normal iteration.

```
Normal iteration: 15 ns
Ring iteration: 2696 ns
```

Fig. 10. Compare normal iteration and ring iteration

B. Rotation-invariance of Ring-Projected Algorithm

The CSV file generated from `RingProjection2D()` is visualize using Python script `plotRingProjection-Function.py`. The result is shown in Fig. 11. All of the orientations result in a similar shape of Ring-Projected function. The relation is described as a box plot in Fig. 12 which is produced by another Python script `boxplot.py`. The result shows that the algorithm produces the same function regarding the different rotation.

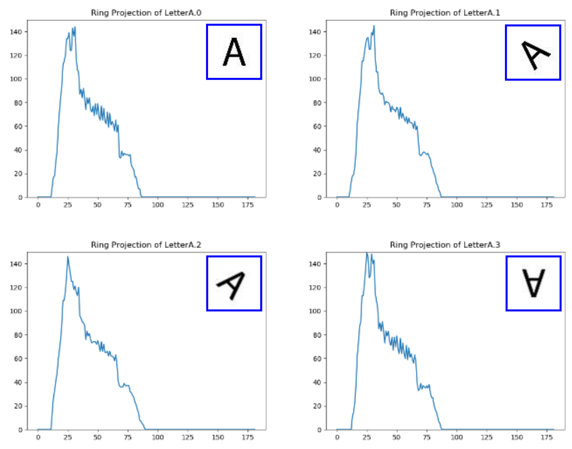


Fig. 11. Ring Projection of letter with different orientations

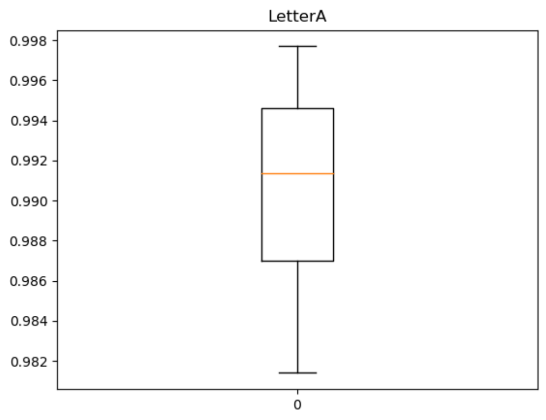


Fig. 12.Box plot of correlation coefficients corresponding to Ring-Projected functions of letter A

C. Ring Projection with MNIST Dataset

The statistical CSV file is transform into graphical view (See Fig. 13) using a Python script `plotStat.py`. The figure describes the mean value as a dot and the standard deviation as black line around the dot. The gray line shows the minimum and maximum value of cross-correlation coefficients. It is understandable that for handwritten digits from many writers, there are some which are totally different to the other. This results in almost zero correlation or even negative correlation. However, despite the variety, the average values are close to 1 and standard deviations are approximately 0.1 which proves that MNIST dataset is reliable for training purpose.

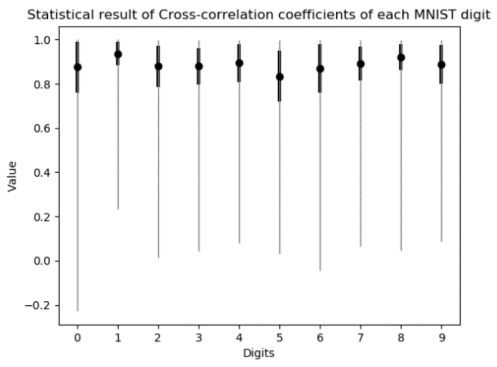


Fig. 13. Visualization of the relation among MNIST images with the same label

IV. Discussion and Conclusion

The goal of the project is the implementation of Ring-Projected algorithm for Dimensionality Reduction as a module for LearningAPI. The module gets a `double[][]` array, which holds the coordinates of recognizing pattern, transforms 2-dimensional function into one dimensional function. Return type of the module is `double[]` array. When testing the algorithm, it is required to extend ImageBinarizer that provides the desired input for the algorithm and create more pipeline modules that handle loading and giving the output. These additional pipeline modules are created not only for the necessity of the project but also for future purpose. Also, some Python scripts are used to visualize the output data by creating graphs. The algorithm is proved to be rotation-invariant and can works with either alphabet or digit. In addition, using the algorithm on the MNIST dataset of handwritten digits provides a positive result.

The disadvantage of this algorithm is it take longer time to loop through a `double[][]` array than looping as usual way through rows and columns. Therefore, when applying to a large-size image, it would take a lot of time even when running with a powerful computer. However, when using it for character recognition, that would not be a challenge since character size is not too large.

For future development, the algorithm can be modified to apply on a 3D object and theoretically still keeps its rotation-invariant characteristic.

V. References

[1] Open Source, "UniversityOfAppliedSciencesFrankfurt/LearningApi: Machine Learning foundation on top of .NET Core," 2019. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi. [Accessed 11 02 2019].

[2] G.-C. Pan, "A Tutorial of Wavelet for Pattern," Taipei.

[3] Y. LeCun, C. Cortes and C. J. Burges, "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed 2019].