Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

TINTO: Converting Tidy Data into image for classification with 2-Dimensional Convolutional Neural Networks

Manuel Castillo-Cara [a,*], Reewos Talla-Chumpitaz [b], Raúl García-Castro [a], Luis Orozco-Barbosa [c]

[a] Universidad Politécnica de Madrid, Madrid, Spain
[b] Universidad Nacional de Ingenieria, Lima, Peru
[c] Albacete Research Institute of Informatics, Universidad de Castilla-La Mancha, Albacete, Spain

A R T I C L E I N F O

Article history:
Received 16 January 2023
Received in revised form 28 March 2023
Accepted 18 April 2023

Keywords:
Tabular to image conversion
Image classification
Image blurring technique
Image generation
Convolutional neural networks
Tabular data into image

A B S T R A C T

The growing interest in the use of algorithms-based machine learning for predictive tasks has generated a large and diverse development of algorithms. However, it is widely known that not all of these algorithms are adapted to efficient solutions in certain tidy data format datasets. For this reason, novel techniques are currently being developed to convert tidy data into images with the aim of using Convolutional Neural Networks (CNNs). TINTO offers the opportunity to convert tidy data into images through the representation of characteristic pixels by implementing two dimensional reduction algorithms: Principal Component Analysis (PCA) and t-distributed Stochastic Neighbour Embedding (t-SNE). Our proposal also includes a blurring technique, which adds more ordered information to the image and can improve the classification task in CNNs.

Code metadata

Current code version	v1.2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00032
Permanent link to Reproducible Capsule	https://github.com/oeg-upm/TINTO
Legal Code License	Apache License, 2.0
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	scipy, sklearn, pandas, matplotlib, argparse, numpy, math, pickle
If available Link to developer documentation/manual	https://github.com/oeg-upm/TINTO
Support email for questions	jcastillo@fi.upm.es

1. Motivation and significance

Nowadays, there is great interest in being able to use data for predictive purposes in any area of application. Towards this end, there are numerous Machine Learning (ML) algorithms, Multilayer Perceptron (MLP), and advanced Convolutional Neural Networks (CNNs) [1,2]. In the case of ML and MLP, a series of structured data, such as tabular or tidy data format (here on in referred to as tidy data), are required as input [3]. In the case of CNNs, the input data must be nonstructured, specifically in image format [4].

Due to increasing interest in exploiting the power of CNNs, the use of techniques to convert tidy data into images has been the subject of an open research area that has developed in recent years [5,6]. Recently, several works have been developed based on the transposition of ordered data into images [7,8]. The main mechanisms of these works differ in the conversion framework used in the image construction process [6,9]. The main challenge is to create images from tabular data while preserving the features embedded in the data [8]. This will enable us to fully exploit the use of CNNs in model fitting and generalisation [10,11].

* Corresponding author.
E-mail addresses: manuel.castillo.cara@upm.es (Manuel Castillo-Cara), reewos.talla.c@uni.pe (Reewos Talla-Chumpitaz), r.garcia@upm.es (Raúl García-Castro), luis.orozco@uclm.es (Luis Orozco-Barbosa).

https://doi.org/10.1016/j.softx.2023.101391

Consequently, the DeepInsight framework [9], the basis framework of TINTO software (here in on referred as TINTO), performs the transformation to images with dimensionality reduction algorithms, so that it can obtain Cartesian coordinates for each feature in the dataset. From the generated coordinates, it generates an image pattern where a pixel represents the feature and the value of each pixel would be given by the value of the feature. Finally, the generated images can be processed by a CNN to obtain the final classification result [10].

Therefore, taking into account the DeepInsight framework, our proposal starts by transforming tidy data, in a classification problem in ML, into images to allow the modelling of a CNN [12,13]. The proposal uses the methods and mechanisms of the articles [9,10] as the starting point for developing the overall processing schema. In the first step of our proposed schema, two dimensional reduction techniques [14] are used to transform tidy data into images: t-distributed Stochastic Neighbour Embedding (t-SNE) [5,15] and Principal Component Analysis (PCA) [16]. Moreover, TINTO adds the classical painting technique known as blurring to represent more ordered contextual information in the resulting image and, thus improve the process of feature extraction and generalisation of CNNs [10].

TINTO has been developed based on the framework introduced in DeepInsight [9] by the authors of this article in Python. TINTO also includes the widely used blurring technique in plastic arts, e.g. professional painting (not to be confused with the computer vision blurring technique). The main contributions of TINTO are the following:

- Use of two dimensionality reduction algorithms to convert tidy data into images, spatial distributed data readings, namely, the t-SNE and PCA algorithms.
- Use of the blurring painting technique to introduce contextual information to the image, which can improve the classification process.
- Use CNNs models for binary or multiclass classification problems in classical ML as the TINTO allows the conversion of Tidy Data into images.

2. Software description

This section describes the software architecture and specifications and the image rendering techniques for TINTO.

2.1. Software architecture

One of the first transformations to be performed is to convert tidy data into image data format, i.e., the corresponding sample (row) in the sampling phase (see Table 1). The reader can learn more about the mathematical and framework foundations of the transformation process by consulting previous scientific articles [9,10]. The following considerations should be taken into account when creating the images:

- The input dataset must be in CSV, taking into account the tidy data format [3].
- The target (variable to be predicted), Y, should be set as the last column of the dataset. Therefore, the first columns will be the features, X.
- All data must be in numerical data type. TINTO does not accept data in string or any other non-numeric data type.
- TINTO will create as many folders with their corresponding images in each folder as there are targets. For example, in this case, the dataset has 15 different targets, so TINTO will create 15 different folders, one for each class.

Table 1
Structure of the dataset in tidy data format. The first 5 columns would be the features (Fe) and the last column would be the target. The first row is the heading of each column.

Fe01	Fe02	Fe03	Fe04	Fe05	Target
−65	−61	−74	−73	−67	1
−60	−57	−83	−62	−69	2
−66	−70	−78	−63	−73	3
…	…	…	…	…	…
−58	−66	−71	−73	−69	14
−60	−62	−73	−69	−57	15

Accordingly, Fig. 1 depicts the tidy data to image transformation process. As can be seen, tidy data are converted into image data through a two dimensional space in X and Y. This translation to two dimensional space enables us to build an image of characteristic pixels for each sample of the dataset, i.e., each data sample is converted to a two dimensional space.

Therefore, the pipeline for converting each sample in tidy data format into a two dimensional space consists of four main processing tasks (see Fig. 1):

1. Data dimensionality reduction. The initial matrix is transposed. Note that each feature is represented with a different colour for presentation purposes, although the resulting figure will consist of two channels, i.e., black and white. This task makes use of a dimensionality reduction algorithm. In this case, TINTO explored two such algorithms: PCA and t-SNE.
2. Centre of mass and delimitation. Having obtained the coordinates, the centre of gravity of the points is determined, and the area is subsequently delimited.
3. Scaling and pixel positions. The matrix is transposed, scaled and the values are rounded to integer values.
4. Characteristic pixel positions. The values obtained would be the positions of the characteristic pixels for the creation of the image pattern.

We should note that the settings of the parameters of the dimensionality reduction algorithms and the blurring technique play a major role in the image generation process. Readers interested on making use of TINTO should further consult Refs. [9,10]. which includes an analysis of the impact of the most relevant parameters of the reduction algorithms and the blurring technique. Parameters, such as the initial random seed of the t-SNE algorithm may provide undesirable artefacts, such as the overlap of one or more characteristic pixels. Ref. [10], provides the required steps to obtain the positions of the characteristic pixels from the initial tidy data.

2.2. Image rendering techniques

This section discusses the image generation process based on the two implemented cases, i.e., characteristic pixels with and without the blurring technique.

Case 1: Characteristic pixels without blurring

In this case, TINTO fills in the data A_{scale} at their corresponding characteristic pixel positions in the matrix M. The matrix M is then converted into an image. This is done for each A_{scale} sample. All other values of the matrix M are zeros, so they are not significant in this case.

For the creation of images from the tidy data, the characteristic pixel position and the scale generated by the normalisation are used. Fig. 3(a) shows an example of the results without the blurring technique.

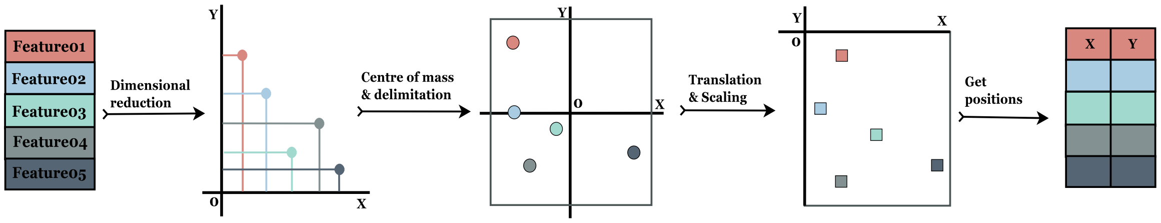


Fig. 1. Process for obtaining the feature coordinates from the transpose matrix. It can be seen that the tidy data are converted into a two dimension matrix through the different phases with their respective techniques, i.e., delimitation, translation, and so on.

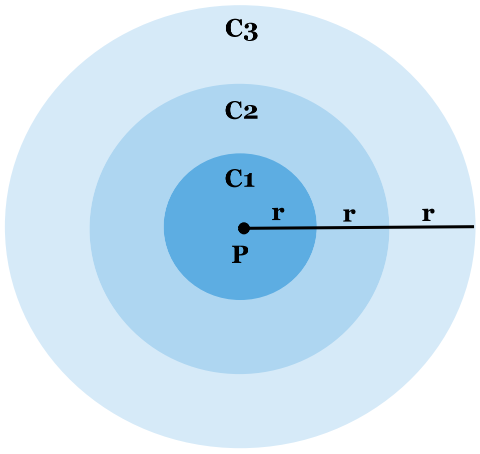


Fig. 2. Representation of the blurring technique. The colour represents the intensity for each stroke or circumference. It moves away from the characteristic pixel until it disappears when the blurring technique is performed. C1 is the characteristic pixel with the highest intensity, C2 are the neighbouring pixels with lower intensity than C1, and C3 are the neighbouring pixels with lower intensity than C3.

Case 2: Characteristic pixels with blurring

This procedure was created empirically, based on the blurring technique used in plastic arts. This technique is often used to soften and extend strokes so that the transition from intense to faint is uniform. Then, by making an analogy with the characteristic pixels as spots on a canvas, these spots can be blurred so that they cover more area without losing the intensity of the original characteristic pixel [10].

Accordingly, the use of the blurring technique of the characteristic pixels is proposed to enlarge their area. To do this, the thickness, number, and intensity of the strokes must be defined. Each stroke would be a circumference around the pixel and the previous stroke with a thickness called *distance*, represented as r; and the number of strokes or circles would be denoted as *steps*, represented as C_x, see Fig. 2. Moreover, the intensity of this stroke or circumference would be determined by the encompassed area of the circumference, with the centre (the characteristic pixel) being the initial intensity, represented as P. Additionally, the intensity fades as you move away from the characteristic pixel.

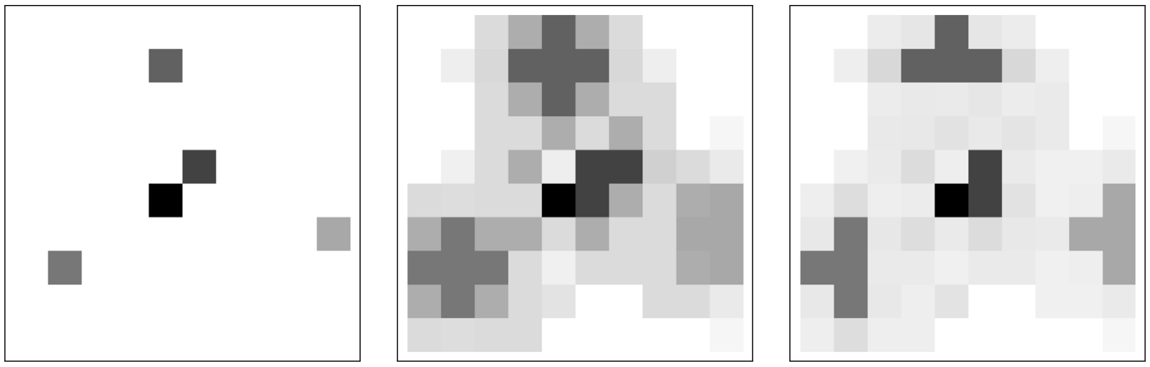
Finally, by using this technique, it is possible to determine the step size in which the blurring of two characteristic pixels overlaps. In this case, TINTO experimented with two possible solutions for the representation of the scene with overlapping pixels:

- **Average value**: An alternative to the above is to average the intensity of both, the one already in the matrix and the new one (see Fig. 3(b)).
- **Maximum value**: The highest scoring value pixel of the overlapping pixels is chosen, i.e., if the value of the pixel in the matrix is lower than the value of the incoming pixel, it is replaced; otherwise, it is kept (see Fig. 3(c)).

2.3. Software specifications

In this context, having seen the steps in which TINTO performs a conversion from tidy data into images, this section will describe the most relevant Python code. The functions shown below demonstrate the process of creating the characteristic pixels and, above all, the blurring procedure using filters and submatrices. Therefore, we show the main three functions implemented: `createFilter`, `blurringFilter` and `imageSampleFilter`. Note that all Python code can be found in the repository [17].

In `createFilter` function (see Alg. 1) the filter is a matrix of size $2 \times distance \cdot total\_steps + 1$. This matrix covers the entire circular space of the characteristic pixel determined by the distance per total step. This "filter" would be multiplied by



(a) Without blurring. (b) Blurring with maximum value. (c) Blurring with average value.

Fig. 3. Image samples generated by TINTO using PCA.

a scalar that is the intensity value. This resulting matrix would then be placed as a submatrix within the final matrix, where the centre of the submatrix would be the position of the characteristic pixel.

```python
def createFilter(distance=2, steps=3, amplification=np.pi):
    size_filter = int(2 * distance * steps + 1)
    center_x = distance * steps
    center_y = distance * steps
    print(distance,steps)
    filter  = np.zeros([size_filter,size_filter])

    for step in reversed(range(steps)):
        r_actual = int(distance*(step+1))
        intensity=min(amplification*1/(np.pi*r_actual**2),1)
        lim_inf_i = max(center_x - r_actual - 1, 0)
        lim_sup_i = min(center_x + r_actual + 1, size_filter)
        lim_inf_j = max(center_y - r_actual - 1, 0)
        lim_sup_j = min(center_y + r_actual + 1, size_filter)
        for i in range(lim_inf_i, lim_sup_i):
            for j in range(lim_inf_j, lim_sup_j):
                if((center_x-i)**2 + (center_y-j)**2 <= r_actual**2):
                    filter[i,j]=intensity
    filter[center_x,center_y] = 1
    return filter
```

Listing 1: Blurring filter specification

On the other hand, this function adds more ordered contextual information to the image through the classical painting technique called blurring (see Alg. 2). It develops the following main steps: (i) take the coordinate matrix of the characteristic pixels; and (ii) create the blurring according to the number of steps taken in a loop.

```python
def blurringFilter(matrix, filter, values, coordinates, option):
    iter_values = iter(values)
    size_matrix = matrix.shape[0]
    size_filter = filter.shape[0]
    matrix_extended = np.zeros([size_filter+size_matrix, size_filter+size_matrix])
    matrix_add = np.zeros([size_filter+size_matrix, size_filter+size_matrix])
    center_filter = int((size_filter - 1)/2)
    for i,j in coordinates:
        i = int(i)
        j = int(j)
        value = next(iter_values)
        submatrix = filter * value
        lim_inf_i = i
        lim_sup_i = i+2*center_filter+1
        lim_inf_j = j
        lim_sup_j = j+2*center_filter+1
        if(option=='mean'):
            matrix_extended[lim_inf_i:lim_sup_i,lim_inf_j:lim_sup_j] += submatrix
            matrix_add[lim_inf_i:lim_sup_i,lim_inf_j:lim_sup_j] += (submatrix > 0)*1
        elif(option=='maximum'):
            matrix_extended[lim_inf_i:lim_sup_i,lim_inf_j:lim_sup_j] = np.maximum(matrix_extended[lim_inf_i:lim_sup_i,lim_inf_j:lim_sup_j], submatrix)
    if(option=='mean'):
        matrix_add[matrix_add == 0] = 1
        matrix_extended = matrix_extended / matrix_add
    matrix_final = matrix_extended[center_filter:-center_filter-1,center_filter:-center_filter-1]
    return matrix_final
```

Listing 2: Representation of the Blurring filter in the coordinate matrix.

Finally, this function creates the images using the two previous functions to represent the characteristic pixel and blurring technique (see Alg. 3).

```python
def imageSampleFilter(X, Y, coord, matrix, folder, amplification, distance=2, steps=3, option='maximum', train_m=False):
    if distance * steps * amplification != 0:
        filter = createFilter(distance,steps,amplification)
    for i in range(X.shape[0]):
        matrix_a = np.zeros(matrix.shape)
        if distance * steps * amplification != 0:
            matrix_a = blurringFilter(matrix_a, filter, X[i], coord, option)
        else:
            iter_values_X = iter(X[i])
            for eje_x,eje_y in coord:
                matrix_a[int(eje_x),int(eje_y)]=next(iter_values_X)
    ...
    return matrix
```

Listing 3: Main function to create the images

3. Illustrative examples

3.1. Command execution

Before visualising the images generated according to a dataset as shown in Table 1, the basic guidelines are presented to run TINTO in the command terminal. Note that the dataset guidelines are specified in Section 2.1.

All TINTO options are printed with the help parameter:

```
$ python tinto.py -h
```

where,

- `python`: Python run command.
- `tinto.py`: TINTO Python script.
- `-h`: Show the TINTO parameters and exit.

The following command executes TINTO with the default parameter values, i.e., the generation of images with the characteristic pixels (no blurring), PCA and the image size to 20 × 20 pixels:

```
$ python tinto.py "dataset.csv" "folder"
```

where,

- `''dataset.csv''`: Set the path of the dataset (CSV).
- `''folder''`: Sets the path where images are saved.

Finally, the next command indicates some parameters that TINTO accepts:

```
$ python tinto.py "dataset.csv" "folder" -B -alg t-SNE \
 -oB maximum -px 30 -sB 5
```

where,

- `-B`: Create images with the blurring technique.
- `-alg t-SNE`: Select the *t*-SNE algorithm.
- `-oB maximum`: Create images with maximum overlapping pixel value.
- `-px 30`: Set the image size to 30 × 30 pixels.
- `-sB 5`: Expand the blurring to 5 pixels.

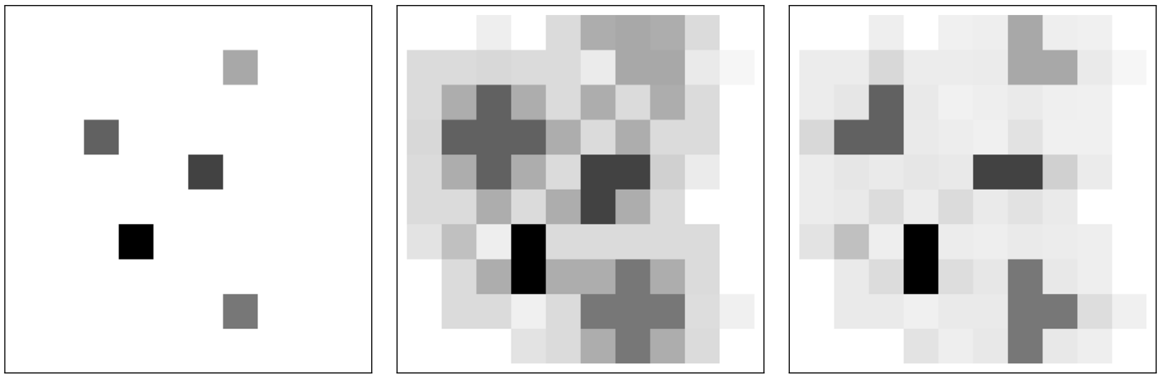

(a) Without blurring.
(b) Blurring with maximum value.
(c) Blurring with average value.

Fig. 4. Image samples generated by TINTO using t-SNE.

3.2. Image generation

Further to the creation of images based on the t-SNE and PCA dimensionality reduction algorithms, TINTO includes the mechanisms to further process and obtain three different image modalities: images of characteristic pixels (without blurring) and with maximum or average blurring values. Each image is stored in a different folder, i.e., each target sample of the dataset results in an image stored in a separate folder. We have included in [9,10,17] a description of the dimensionality reduction algorithms and blurring technique and their main input parameters.

Consequently, Figs. 3 and 4 show the generation of three sample images for the different cases under study, PCA and t-SNE, respectively. These sample images are clear examples of the resulting images that can be inputted into CNN.

3.3. Results

TINTO can improve the generalisation and learning process in ML classification problems. Accordingly, in DeepInsight [9], which uses characteristic pixels (without blurring), the framework is compared with the Random Forest algorithm on different datasets. DeepInsight achieves more than 90% accuracy in most of them, improving results over Random Forest, e.g., for the RNA-seq it improves by 3%, for vowels it improves by 7%, etc.

The technique has been successfully tested in an indoor localisation use-case [10]. The localisation accuracy results based on the use of images of characteristic pixels (without blurring) have shown a substantial improvement over the results reported by the classical ML algorithms [18]. Furthermore, the use of the blurring technique has further improved the results by another 12% regardless of the blurring technique used, i.e., maximum or average.

4. Impact

As discussed in the previous sections, TINTO has the main purpose of converting tidy data into images. Accordingly, TINTO creates the possibility of being able to develop CNN-based models to solve classification problems in classical ML in which either there were solutions with very little convergence of the models or, as was the case in many cases, they could not be solved due to poor generalisation and model fitting. In fact, the following major impacts are expected:

- TINTO is an open source software based on [9,10] that converts tidy data into images and can be easily extended or modified.
- TINTO has an object-oriented structure consisting of multiple Python modules that allow for easier debugging and reuse of codes through inheritance. This saves the time needed for coding and manipulating data, and thus finding direct solutions by extending tidy data format solutions to CNNs.
- TINTO allows us to convert tidy data into images mainly for classification problems in classical ML. Hence, TINTO allows us to extend the search for solutions of complex problems to CNNs when classical ML algorithms or MLP did not generalise or did not approach an acceptable solution.
- TINTO uses well-documented dimensionality reduction algorithms such as PCA and t-SNE in the image construction process. Note that these are not the only algorithms, and more of these algorithms or other techniques can be added and studied to represent characteristic pixels in images. With TINTO, this could be done very quickly and easily by adding a few additional lines of code.
- TINTO also presents the classical painting technique called blurring, which allows contextual information to be added to the image in an orderly manner. This allows us to improve, in many cases, the classification process and, therefore, the generalisation and fitting in CNNs [10].
- TINTO performs the training and validation process for image conversion in a fairly acceptable time. For example, for the Iris dataset, it takes about 10 minutes, and for the data used as an example in [10] it takes 15 minutes. Note that datasets with large amounts of samples can take hours; this is due to the training process of the dimensionality reduction algorithms, i.e., in this case, t-SNE and PCA.

5. Conclusions and open challenges

This article introduces TINTO: software to convert tidy data into images. This conversion, from structured data into unstructured data, makes possible the use of CNNs to solve complex problems and opens their use to solve complex problems where classical ML algorithms or customised MLP solvers may be unsuitable.

To this end, two dimensionality reduction algorithms as t-SNE and PCA were evaluated. In this context, two cases of image generation were tested, namely: (i) use the characteristic pixels; and (ii) apply the blurring classical painting technique.

Finally, one of the main points to develop is to be able to set up more dimensionality reduction algorithms and to study the impact of each of them on the generation of images. Furthermore, one of the main challenges that should be studied is the impact on the generation of patterns based on the types of data and the distribution of these data in a dataset.

CRediT authorship contribution statement

Manuel Castillo-Cara: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Reewos Talla-Chumpitaz:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization. **Raúl García-Castro:** Formal analysis, Investigation, Methodology, Funding acquisition, Supervision, Writing – original draft, Writing – review & editing. **Luis Orozco-Barbosa:** Formal analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All resources, as well as the source code of TINTO can be found on Github [17]

Acknowledgements

The research leading to these results has been funded by the Spanish Ministry of Science and Universities (MCIU) and the European Commission (EC), Grant No. RTI2018-098156-B-C52; by the CYTED, Grant No. 520rt0011; and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with the Universidad Politécnica de Madrid in the Excellence Programme for University Teaching Staff, in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). Manuel Castillo-Cara is supported by a postdoctoral research grant (Recualificación - María Zambrano) of the European Union-NextGenerationEU programme and the Spanish Ministry of Universities, ref. No. UP2021-035.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2023.101391.

References

[1] Sharma A, Kumar D. Classification with 2-D Convolutional Neural Networks for breast cancer diagnosis. Sci Rep 2022;12(1):21857.

[2] Rangel JC, Martínez-Gómez J, Romero-González C, García-Varea I, Cazorla M. Semi-supervised 3D object recognition through CNN labeling. Appl Soft Comput 2018;65:603–13. http://dx.doi.org/10.1016/j.asoc.2018.02.005.

[3] Wickham H. Tidy data. J Stat Softw 2014;59(10):1–23. http://dx.doi.org/10.18637/jss.v059.i10.

[4] Vasquez-Espinoza L, Castillo-Cara M, Orozco-Barbosa L. On the relevance of the metadata used in the semantic segmentation of indoor image spaces. Expert Syst Appl 2021;184:115486. http://dx.doi.org/10.1016/j.eswa.2021.115486.

[5] Van der Maaten L, Hinton G. Visualizing data using t-SNE. J Mach Learn Res 2008;9(11).

[6] Zhu Y, Brettin T, Xia F, Partin A, Shukla M, Yoo H, et al. Converting tabular data into images for deep learning with Convolutional Neural Networks. Sci Rep 2021;11(1):1–11.

[7] Sun B, Yang L, Zhang W, Lin M, Dong P, Young C, et al. Supertml: Two-dimensional word embedding for the precognition on structured tabular data. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2019.

[8] Lee E, Nam M, Lee H. Tab2vox: CNN-based multivariate multilevel demand forecasting framework by tabular-to-voxel image conversion. Sustainability 2022;14(18):11745.

[9] Sharma A, Vans E, Shigemizu D, Boroevich KA, Tsunoda T. DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture. Sci Rep 2019;9(1):1–7.

[10] Talla-Chumpitaz R, Castillo-Cara M, Orozco-Barbosa L, García-Castro R. A novel deep learning approach using blurring image techniques for bluetooth-based indoor localisation. Inf Fusion 2023;91:173–86. http://dx.doi.org/10.1016/j.inffus.2022.10.011.

[11] Iqbal MI, Mukta MSH, Hasan AR, Islam S. A dynamic weighted tabular method for Convolutional Neural Networks. IEEE Access 2022.

[12] Maaten Lvd, Hinton G. Visualizing data using t-SNE. J Mach Learn Res 2008;9(Nov):2579–605.

[13] Borisov V, Leemann T, Seßler K, Haug J, Pawelczyk M, Kasneci G. Deep neural networks and tabular data: A survey. 2021, arXiv preprint arXiv:2110.01889.

[14] Menvouta EJ, Serneels S, Verdonck T. Direpack: A Python 3 package for state-of-the-art statistical dimensionality reduction methods. SoftwareX 2023;21:101282. http://dx.doi.org/10.1016/j.softx.2022.101282.

[15] Wattenberg M, Viégas F, Johnson I. How to use t-SNE effectively. Distill 2016. http://dx.doi.org/10.23915/distill.00002.

[16] Kim D, Joo D, Kim J. TiVGAN: Text to image to video generation with step-by-step evolutionary generator. IEEE Access 2020;8:153113–22. http://dx.doi.org/10.1109/ACCESS.2020.3017881.

[17] Castillo-Cara M, García-Castro R, Orozco-Barbosa L. TINTO: Converting Tidy Data into Image for Classification with 2-Dimensional Convolutional Neural Networks. 2023, Ontology Engineering Group, GitHub, URL https://github.com/oeg-upm/TINTO. [Online: Accessed 10 January 2023].

[18] Lovón-Melgarejo J, Castillo-Cara M, Huarcaya-Canal O, Orozco-Barbosa L, García-Varea I. Comparative study of supervised learning and meta-heuristic algorithms for the development of bluetooth-based indoor localization mechanisms. IEEE Access 2019;7:26123–35. http://dx.doi.org/10.1109/ACCESS.2019.2899736.