Information Sciences 477 (2019) 448–465

Contents lists available at ScienceDirect

Information Sciences

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

Convolutional neural network learning for generic data classification

Huimei Han[a,b,*], Ying Li[a,*], Xingquan Zhu[b]

[a] School of Telecommunications Engineering, Xidian University, Xi'an, Shannxi 710071, China
[b] Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA

ARTICLE INFO

Article history:
Received 11 June 2018
Revised 17 October 2018
Accepted 28 October 2018
Available online 30 October 2018

Keywords:
Deep learning
Feature learning
Convolutonal neural networks
Classification

ABSTRACT

Convolutional Neural Network (CNN) uses convolutional layers to explore spatial/temporal adjacency to construct new feature representations. So, CNN is commonly used for data with strong temporal/spatial correlations, but cannot be directly applied to generic learning tasks. In this paper, we propose to enable CNN for learning from generic data to improve classification accuracy. To take the full advantage of CNN's feature learning power, we propose to convert each instance of the original dataset into a synthetic matrix/image format. To maximize the correlation in the constructed matrix/image, we use 0/1 optimization to reorder features and ensure that the ones with strong correlations are adjacent to each other. By using a feature reordering matrix, we are able to create a synthetic image to represent each instance. Because the constructed synthetic image preserves the original feature values and correlation, CNN can be applied to learn effective features for classification. Experiments and comparisons, on 22 benchmark datasets, demonstrate clear performance gain of applying CNN to generic datasets, compared to conventional machine learning methods. Furthermore, our method consistently outperforms approaches which directly apply CNN to generic datasets in naive ways. This research allows deep learning to be broadly applied to generic datasets.

1. Introduction

Deep learning represents a series of multi-layered neural network structures with tuneable weight values learned from underlying training data [29,39]. Due to its superb accuracy and feature learning capability, deep learning has been successfully used in many real-world applications, especially in domains involving image/video/audio recognition or time series and financial data analysis [16]. For all these domains, the temporal and/or spatial correlation of the data allow deep learning methods to learn effective features to represent data for better classification. Deep learning models, such as convolution neural networks (CNN) and long-short term memory units (LSTM), commonly utilize data correlation to learn better feature representation [14,21,31,43,44]. Take CNN as an example, when applying CNN to an image, the convolution procedure is equivalent to a spatial filtering process learning meaningful features, such as corners or edges, for image recognition [9,25]. Similarly, one dimensional CNN has also been used to data with temporal correlations, such as stock index [8,12], with convolution being applied to learn meaningful patterns in the data.

* Corresponding author.
E-mail addresses: hanh@fau.edu (H. Han), yli@mail.xidian.edu.cn (Y. Li), xzhu3@fau.edu (X. Zhu).

https://doi.org/10.1016/j.ins.2018.10.053
0020-0255/

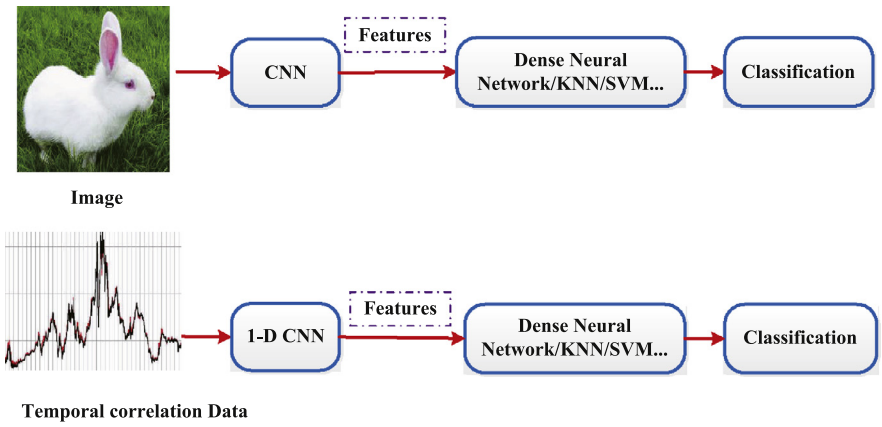

Fig. 1. The typical routines of applying convolutional neural networks to image or data with temporal correlations for classification.

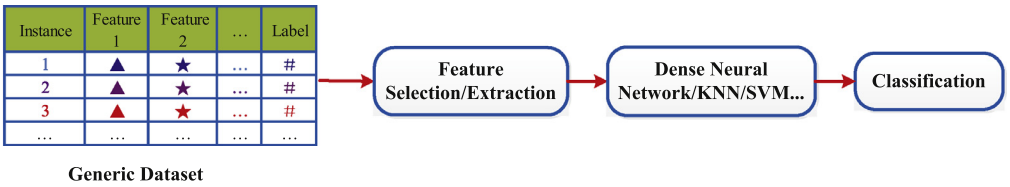

Fig. 2. The typical routine of machine learning for generic data classification, where input data are assumed to be identical and independently distributed (i.i.d), and are represented in instance-feature tabular format.

Indeed, deep learning differs from conventional machine learning in the sense that it can learn good feature representation from the data. Existing deep learning methods largely benefit from this feature learning power to find meaningful features capturing temporal/spatial correlations, as shown in Fig. 1.

In reality, traditional machine learning and data classification consider a much different setting where instances are assumed to be independent and identically (i.i.d.) distributed and features used to represent data are assumed to have weak or no correlation. Because of this assumption, conventional machine learning methods do not consider feature/data correlation in the learning process. Nearly all traditional machine learning methods, including multi-layer neural networks and randomized learning methods, such as stochastic configuration networks (SCNs) [47], do not explicitly consider feature interactions for learning, mainly because they refer feature correlations to be handled by a data processing process which creates independent features before applying machine learning methods. For example, feature extraction, such as principle component analysis or manifold learning [32,37], are common approaches to learn a low dimensional feature representation of the original data. The new orthogonal feature space often produces a better fit for conventional machine learning algorithms to learn accurate classifiers, compared to the ones learned from the original feature space. However, such feature extraction process does not consider temporal or spatial correlation of the data, but reply on arithmetic decomposition of features for learning, as shown in Fig. 2.

The effectiveness of deep learning methods and the popularity of generic learning tasks raise simple questions on whether deep learning is still effective for generic data, and how to apply deep learning to generic data for effective learning and classification. For generic machine learning tasks, data provided for learning are in an instance-feature tabular format, as shown in Fig. 2. Applying deep learning, such as CNN to such data is feasible, but would't be reasonable. For example, one can consider each instance as a one dimensional vector, and carry out 1-D CNN to the instance to learn a new feature representation. This simple approach, however, leaves many concerns on what is rationality of applying CNN to the generic data, what exactly the CNN is learning from the data, what is local field for convolutional feature learning process, and what are the meaning of features learned from such 1-D CNN. More fundamentally, enabling deep learning to generic data classification has the following three major challenges:

- **Higher order feature correlation and ordering:** Given a generic dataset represented in instance-feature format, features in the data have mutual or higher order correlations. Some features are strongly correlated whereas others are independent of each other. Finding correlations between features and use such correlations to create a new representation of the instance is a critical step allowing deep learning to leverage correlation to learn effective features.
- **Deep learning compatible instance representation:** Assume features are suitably ordered, we need a new instance representation to ensure that feature values of the original data are accurately preserved, and the feature correlations are also maximally presented for deep learning modules to learn effective features for classification.

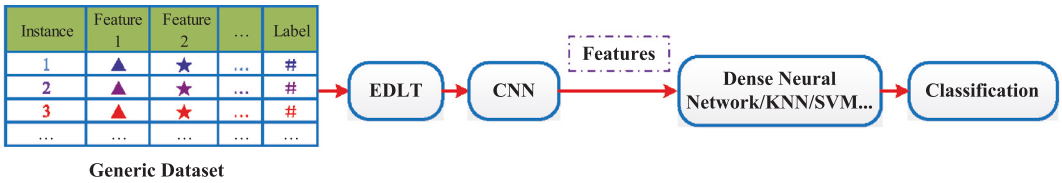

Fig. 3. The routine of the proposed EDLT framework which enables convolutional neural networks for generic data classification. EDLT takes data represented in instance-feature tabular format as input, and transforms each instance into a new format suitable for CNN to learn new features for learning and classification.

- **Theoretical modeling:** To ensure maximized global correlation for CNN to learn good feature representations, we need to find a theoretical basis for feature reordering. So each instance in the original dataset is represented by the new representation with carefully aligned local and global feature correlations for deep learning.

Motivated by the above challenges, in this paper, we study enabling deep learning for generic classification tasks. A typical routine of the proposed EDLT framework is shown in Fig. 3, where the learning is not limited to audio/visual data, like most deep learning methods have been commonly applied to. The contributions of this paper are summarized as follows.

- In order to tackle the first challenge (higher order feature correlation and ordering), we propose to use feature correlation to determine a new ordering of original features for deep learning. More specifically, we utilize Pearson correlation to obtain a feature-feature correlation matrix and a feature-label correlation vector. Based on the correlation matrix and correlation vector, we can reorder features to create local spatial feature correlation.
- To tackle the second challenge (deep learning compatible instance representation), we create a synthetic matrix, where features with the strongest correlation are adjacent to others, to represent each instance for deep learning. Meanwhile, in order to preserve feature values of the original data, the synthetic matrix contains all of the original features and their values (but in different orders). As a result, the deep learning methods can be applied to the synthetic matrix to learn meaningful features for classification.
- For the third challenge (the theoretical modeling), we propose to use 0/1 optimization to ensure that instance representation is created to not only maximize local correlation, but also have a maximal global correlation.

The remainder of the paper is organized as follows. Section 2 reviews existing work on data representation. Section 3 introduces the problem definition and overview of the proposed framework. The details of the EDLT algorithm are elaborated in Section 4. Section 5 analyzes the computational complexity of the EDLT algorithm. The experiments and conclusion are reported in Section 6 and Section 7, respectively.

2. Related work

One major step of machine learning tasks is to find good features to represent the underlying object for learning accurate models. A large number of feature processing methods have been proposed to deal with the data representation problem. We categorize these approaches into the following two categories: Data representation based on feature selection and extraction, and data representation based on deep learning.

2.1. Data representation based on feature selection and extraction

Feature selection and extraction aim to transform data from the original feature space into a low dimensional subspace. This process may require domain expertise information, such as class labels. Feature selection and extraction results are commonly used to support conventional machine learning methods for a better classification accuracy, compared to classifiers learned from the original feature space.

2.1.1. Feature selection

Feature selection methods select a subset of features from the original features to represent the data. Because the selected features are a subset of the original features, the new data representation does not contain new features. Feature selection can be categorized into three categories: filter methods, wrapper methods, and embedded methods. The comparison of different selection algorithms are given in [26]. In summary, filter methods select highly ranked features to represent the original data, and wrapped methods warp predictors to a search algorithm to find features with the highest accuracy [23]. Embedded methods [5,17,27] incorporate the feature selection into the training process to decrease the computation time consumed in wrapper methods.

2.1.2. Feature extraction

Feature extraction methods are common approaches used to learn a low dimensional new feature representation of the original data, such that the new feature space produces a better fit to conventional machine learning methods. Depending

on the availability of label information, feature extraction methods can be roughly categorized into three categorizes: supervised methods, unsupervised methods, and semi-supervised methods. Fisher Discriminant Analysis (FDA), a supervised feature extraction method, extracts discriminant features to represent original data by utilizing the label information [40]. PCA [37], a well-known unsupervised method, transforms features into new orthogonal feature space by utilizing covariance structure of data without label information. Independent Component Analysis (ICA), another well-known unsupervised method, extracts features by minimizing the statistical dependence of the components of the desired representation [28]. Semi-supervised feature extraction methods employ both labeled and unlabeled data to extract features, where the supervised information includes labels or constraints [18]. Semi-supervised universum, a semi-supervised feature extraction method, utilizes the labeled data, unlabeled data and the universum data to address semi-supervised classification problem with high accuracy [48]. For applications such as transfer learning cross-domain learning, feature extractions are also used to link data from different domains for learning [49].

2.2. Deep learning for feature representation learning

Deep learning, such as CNN and LSTM, is a series of multi-layered neural network structures relying on spatial/temporal correlation to learn new feature representation without domain expertise.

CNNs are designed to learn effective features from the original data represented in multiple arrays form, such as 1-D sequences or 2-D images. CNN has been successfully used in image/video/audio/speech processing by utilizing the characteristics of local connections, shared weights, pooling and multi-layers [30]. Recurrent neural networks (RNNs), another commonly used deep learning model, is often utilized to deal with sequential data by learning long-term dependencies, such as speech and language. A previous research [4] has shown that RNNs cannot store the information for a long period of time, accordingly, LSTM is proposed to address this problem by utilizing the memory cell [20], which is proved to be more effective than RNNs [15].

2.3. Differentiation from the proposed work

In summary, the above two types of data representation methods are used for different data format. More specifically, data representation based on feature selection and extraction are usually utilized for generic data represented in instance-feature format, where instances are assumed to be independent and identically (i.i.d.) distributed and features used to represent data are assumed to have weak or no correlation. Nearly all traditional machine learning methods employ the feature selection and extraction to represent the data. Data representation based on deep learning are generally used for data with temporal and/or spatial correlation.

Generally, data representation based on deep learning is more effective than feature selection and extraction methods. On one hand, feature selection and feature extraction are insufficient for dealing with the case that all features are of same significance. On the other hand, existing research [3] shows that data representation based on deep learning achieves better performance than data representation based on feature selection and extraction. Data representation based on deep learning can learn meaningful features to represent initial data to achieve superb accuracy performance. However, this kind of data representation does not work for generic data where features may have weak or no correlation.

In order to address the above issues, our work proposes a new representation of generic data, such that CNN can also be applied to generic datasets to achieve better accuracy for classification. To the best of our knowledge, there is no existing work available for enabling deep learning for generic data classification. In this paper, we limit our experiments to CNN only, but the principle of the underlying data transformation and the overall EDLT framework are valid for other types of deep learning methods.

3. Problem definition and overview framework

3.1. Problem definition

Given a generic dataset $\mathcal{D}$, which contains n instances and m features represented in tabular format, we represent the tth instance as $\boldsymbol{x_t} = \{x_{t,1}, ..., x_{t,m}; y_t\}$, where $x_{t,i}$ and y_t denote the ith feature and label of the instance $\boldsymbol{x_t}$, respectively. The **aim** of EDLT is to find a new representation of instance $\boldsymbol{x_t}$, denoted by $\mathcal{F}(\boldsymbol{x_t})$, such that deep learning methods can be directly applied to $\mathcal{F}(\boldsymbol{x_t})$ to learn features and train a better classifier, compared to the ones trained from the original feature space.

When carrying out conventional machine learning from generic data for classification, two types of learning approaches are commonly used: deterministic-heuristic based learning approaches, and randomization based learning approaches [35,38,47]. The former uses deterministic heuristics to search for best parameters describing the underlying training data. Methods such as k-NN, support vector machines (SVM), decision trees (DT), and multi-layer feed-forward neural networks (NN), all fall into this category. Alternative, randomization based learning approaches, such as stochastic configuration networks (SCNs) [47], use randomized weight values to create a random projection of the original data, followed by a mathematical optimization approaches to separate randomly projected data. While SCNs and multi-layer neural networks share resemblance in networked neurons and tunable weight values as learning model structures, the randomized learning

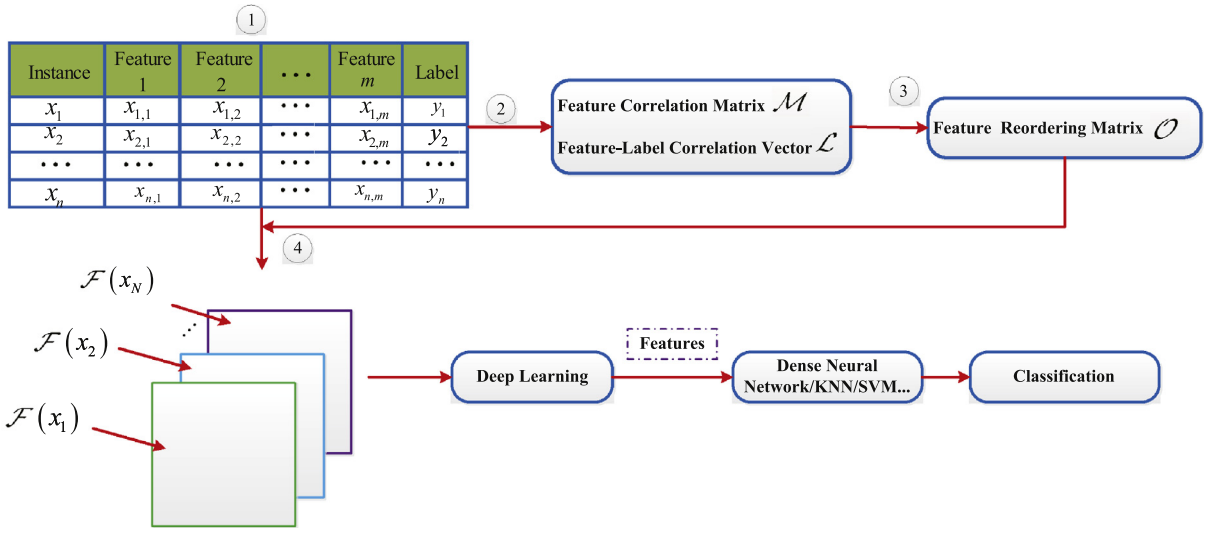

Fig. 4. A conceptual view of the proposed EDLT framework for enabling CNN for generic data classification. Given a generic dataset represented in instance-feature tabular format in ①, EDLT first calculates a feature-feature correlation matrix and a feature-label correlation vector ②. After that, EDLT constructs a feature reordering matrix ③ and converts each instance into a synthetic matrix ④ which is fed into deep learning module, such as CNN, for learning features for classification.

mechanism of SCNs have the characteristics of simplicity in implementation, fast learning and sound performance, which makes them attractive to the tasks than the deterministic-heuristic based learning approaches [38].

In this paper, we use CNN as the deep learning methods, and will compare CNN with both deterministic-heuristic based learning and randomization based learning methods for generic data classification. Our goal is to validate whether CNN based deep learning can deliver a more accurate classifier for generic datasets, compared to deterministic-heuristic based learning and randomization based learning algorithms.

3.2. Overall framework of EDLT

To enable deep learning for generic dataset classification, we propose an EDLT method to convert each instance $\boldsymbol{x_t}$ in the original tabular instance-feature format to a synthetic matrix where features with strong correlations are adjacent to each other to create "artificial correlation" in the data. The rationale behind our motivation is that deep learning mainly leverages temporal and/or spatial correlation of the data to learn effective features for classification, if we can create a synthetic matrix as a "synthetic image" where rows and columns have strong correlations, deep learning module, such as CNN model, will utilize such correlations to learn effective features for classification.

Fig. 4 shows the overall framework of the proposed EDLT method which includes three major steps:

- **Building feature-feature correlation matrix and feature-label correlation vector:** In order to explore feature correlations, we use Pearson correlation to build a pairwise feature-feature correlation matrix $\mathcal{M}$. In addition, we also create a feature-label correlation vector $\mathcal{L}$, using Pearson correlation, to evaluate the relevance of each feature to the class label.
- **Constructing a feature reordering matrix:** In order to obtain deep learning compatible instance representation and preserve original feature values of each instance, we construct a feature reordering matrix $\mathcal{O} \in \mathcal{R}^{m \times m}$ by utilizing $\mathcal{M}$ and $\mathcal{L}$, where $\mathcal{R}$ denotes spaces of real-valued numbers.
- **Generating new presentation of instances:** By using feature reordering matrix $\mathcal{O}$, EDLT reorders original feature values of instance $\boldsymbol{x_t}$ and converts instance $\boldsymbol{x_t}$ into a synthetic matrix format $\mathcal{F}(\boldsymbol{x_t})$.

After converting each instance of the original dataset into a synthetic matrix, where feature values in adjacent rows and adjacent columns share spatial correlation resembling to the spatial adjacent areas in an image, EDLT applies deep learning methods to the matrix representation of each instance $\boldsymbol{x_t}$ to learn new features for classification.

The main procedures of the proposed EDLT algorithm are detailed in Algorithm 1 which includes the three major steps from building feature correlations matrix/vector, constructing feature reordering matrix, and converting instance into the new matrix representation for deep learning.

4. EDLT: Enabling deep learning for generic classification tasks

In this section, we first discuss technical details about feature-feature correlation matrix $\mathcal{M}$, feature-label correlation vector $\mathcal{L}$, and feature reordering matrix $\mathcal{O}$ construction. At the end of the section, we will use an example to demonstrate the conversion of an instance in the original dataset into synthetic matrix format, by utilizing the proposed EDLT method.

Algorithm 1 EDLT: Enabling deep learning for generic classification tasks.

Input:

$\mathcal{D}$: A generic dataset;

$k \times k$: The size of convolution filter;

Output:

The synthetic matrix format of the original dataset $\mathcal{F}(\mathcal{D})$;

1: $\mathcal{M} \leftarrow$ Feature-feature correlation matrix using (1);

2: $\mathcal{L} \leftarrow$ Feature-label correlation vector;

3: Constructing feature reordering matrix $\mathcal{O}$:

1. $\mathcal{O} = \emptyset$;
2. $\mathcal{V} \leftarrow$ Sorting features in a descending order according to their correlation to class label $\mathcal{L}$;
3. $\mathcal{O}_{1,l} = \mathcal{V}_l, l = 1, 2, \ldots, m$: Determine the first row in $\mathcal{O}$;
4. **for each** $i \in [1, 2, \cdots, (m-k+1)]$ **do**
   (a) **for each** $j \in [1, 2, \cdots, (m-k+1)]$ **do**
      i. $\boldsymbol{I_i^j} \leftarrow$ Find the set of known feature indexes in Δ_i^j;
      ii. $\boldsymbol{\alpha_i^j}(\boldsymbol{\alpha_i^0} = \emptyset) \leftarrow$ Find the set of coordinates of unknown elements in Δ_i^j;
      iii. $\boldsymbol{\mathcal{M}_i^j} \leftarrow$ Set the s^{th} row and column in $\mathcal{M}$ to 0 $\{s \in [\mathcal{O}_{\boldsymbol{\alpha_i^1}}, \ldots, \mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$ & $s \notin \boldsymbol{I_i^j}\}$;
      iv. $\boldsymbol{u} = [u_1, \ldots, u_m] \leftarrow$ Apply $\boldsymbol{\mathcal{M}_i^j}$ and $\boldsymbol{I_i^j}$ to obtain the elements in Δ_i^j using (11).
      v. $\boldsymbol{U} = \emptyset$;
      vi. **for each** $u_r \in \boldsymbol{u}$ **do**
         $\boldsymbol{U} = \boldsymbol{U} \cup r$ $(u_r = 1 \& r \notin \boldsymbol{I_i^j})$;
      vii. **end**
      viii. $\mathcal{O}_{\boldsymbol{\alpha_i^j}} = \boldsymbol{U}$;
   (b) **end**
5. **end**

4: **for all** $\boldsymbol{x_t} \in \mathcal{D}$ **do**

5: $\quad \mathcal{F}(\boldsymbol{x_t}) \leftarrow$ Algorithm 2 $(\boldsymbol{x_t}, \mathcal{O})$

6: **end for**

7: **return** $\mathcal{F}(\mathcal{D})$

4.1. Feature-feature correlation matrix $\mathcal{M}$ and feature-label correlation vector $\mathcal{L}$

To handle high-order feature correlation, EDLT first constructs a feature-feature correlation matrix $\boldsymbol{\mathcal{M}} \in \mathcal{R}^{m \times m}$ and a feature-label vector $\boldsymbol{\mathcal{L}} \in \mathcal{R}^{1 \times m}$. The feature correlation matrix and feature-label correlation vector will serve as a basis for EDLT to build a feature reordering matrix and ensure that for any adjacent area of the new constructed matrix the correlation of the reordered feature values is maximized.

To capture pair-wise correlation between features, we utilize Pearson correlation coefficient, a common metric used to measure the correlation between two random variables [19], to compute the matrix $\boldsymbol{\mathcal{M}}$ and vector $\boldsymbol{\mathcal{L}}$ as follows.

$$\mathcal{M}_{i,j} = \frac{\sum_{c=1}^{N}(x_{c,i} - \bar{f}_i)(x_{c,j} - \bar{f}_j)}{\sqrt{\sum_{c=1}^{N}(x_{c,i} - \bar{f}_i)^2}\sqrt{\sum_{c=1}^{N}(x_{c,j} - \bar{f}_j)^2}}, \tag{1}$$

where $\bar{f}_i$ and $\bar{f}_j$ are the sample means of feature i and j respectively, which can be written as

$$\bar{f}_i = \sum_{c=1}^{N} x_{c,i}, \quad \bar{f}_j = \sum_{c=1}^{N} x_{c,j}. \tag{2}$$

Similarity, we can use Pearson correlation or other measures, such as Chi-Square or Information Gain, to calculate correlation between each feature and the class label as a correlation vector $\boldsymbol{\mathcal{L}}$.

$$\mathcal{L}_{1,i} = \frac{\sum_{c=1}^{N}(x_{c,i} - \bar{f}_i)(y_c - \bar{y})}{\sqrt{\sum_{c=1}^{N}(x_{c,i} - \bar{f}_i)^2}\sqrt{\sum_{c=1}^{N}(y_c - \bar{y})^2}}, \tag{3}$$

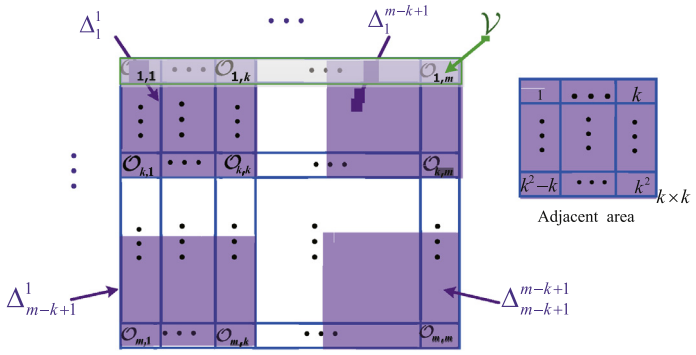

Fig. 5. A conceptual view of the feature reordering matrix $\mathcal{O}$ for EDLT. The left panel shows the $m \times m$ feature reordering matrix $\mathcal{O}$ (where m denotes the number of features of the original dataset), and the right panel shows an adjacent area covering a $k \times k$ area of $\mathcal{O}$. Each row of the matrix $\mathcal{O}$ contains $m - k + 1$ such $k \times k$ adjacent areas. EDLT aims to not only maximize the feature correlations in each adjacent area (*i.e.* features used in the adjacent areas have the maximum correlation), but also maximizes the sum of correlations of all feature pairs used in $\mathcal{O}$.

where $\bar{y}$ is the sample mean of label, which can be computed as

$$\bar{y} = \sum_{c=1}^{N} y_c. \tag{4}$$

The absolute values in $\mathcal{M}$ and $\mathcal{L}$ range from 0 to 1. The larger the value, the stronger the correlation is. Because we only focus on the magnitude of the correlation, we use absolute values of $\mathcal{M}$ and $\mathcal{L}$ throughout the paper.

4.2. Feature reordering matrix $\mathcal{O}$ construction

Given the feature-feature correlation matrix $\mathcal{M}$ and the feature-label correlation vector $\mathcal{L}$, we construct a feature reordering matrix $\mathcal{O}$ to create areas with strongly correlated adjacent features for deep learning methods.

Each element in the feature reordering matrix $\mathcal{O}$ denotes index of feature used to convert each instance from its initial tabular feature format into a synthetic matrix, according to Algorithm 2. Because deep learning methods, such as CNN, aim

Algorithm 2 Generating synthetic matrix $\mathcal{F}(\boldsymbol{x_t})$ for instance $\boldsymbol{x_t}$.

Input:

$\mathcal{O}$: The feature reordering matrix;

$\boldsymbol{x_t}$: The t^{th} instance in the original dataset;

Output:

The synthetic image format $\mathcal{F}(\boldsymbol{x_t})$;

```
i = 1, j = 1;
while i ≠ m do
    while j ≠ m do
        F(x_t)_{i,j} = x_{t,O_{i,j}}
        i = i + 1
    end while
    j = j + 1
end while
```

to explore spatial/tempral correlation within small areas, we can create reordering matrix $\mathcal{O}$ such that each adjacent area of $\mathcal{O}$ will have maximized feature value correlations. Therefore, we define the area in synthetic matrix $\mathcal{F}(\boldsymbol{x_t})$ that will be multiplied by the entries of the convolution filter as an adjacent area. Based on Algorithm 2, the adjacent area in $\mathcal{F}(\boldsymbol{x_t})$ corresponds to the same area in $\mathcal{O}$. In other words, each adjacent area $\mathcal{F}(\boldsymbol{x_t})$ is determined by the feature reordering matrix $\mathcal{O}$.

Considering a $k \times k$ convolution filter and setting the convolution stride to 1, as shown in Fig. 5, each row of the matrix $\mathcal{O}$ contains $m - k + 1$ adjacent areas, and the total number of adjacent areas in matrix $\mathcal{O}$ is $(m - k + 1)^2$. We use Δ_i^j to denote the jth adjacent area in the ith row of $\mathcal{O}$, which is shown in Fig. 5. Our aim is to maximize the sum of the features correlation in each adjacent area, which can be formulated as

$$\arg\max_{\mathcal{O}} \sum_{i=1}^{m-k+1} \sum_{j=1}^{m-k+1} C(\Delta_i^j), \tag{5}$$

where $C(\Delta_i^j)$ is the sum of features correlation in adjacent area Δ_i^j, can be written as

$$C(\Delta_i^j) = \sum_{s=j}^{j+k-1} \sum_{t=i}^{i+k-1} \sum_{r=j}^{j+k-1} \sum_{q=i}^{i+k-1} \mathcal{M}_{\mathcal{O}_{s,t},\mathcal{O}_{r,q}}. \tag{6}$$

(6) can be represented by the matrix format as follows

$$\begin{aligned} &C(\Delta_i^j) = (\boldsymbol{u^{ij}})^{\mathrm{T}} \boldsymbol{\mathcal{M}} \boldsymbol{u^{ij}}. \\ &\text{s.t.} \quad u_r^{ij} = d_r, r \in \boldsymbol{I_i^j}, u_r^{ij} \in \{0, 1, \ldots, k^2\}. \end{aligned} \tag{7}$$

where $(\cdot)^{\mathrm{T}}$ stands for the transposition, $\boldsymbol{u^{ij}} = [u_1^{ij}, \ldots, u_m^{ij}]$ is an m-dimensional column vector, $\boldsymbol{I_i^j}$ denotes the set of known feature indexes in the adjacent area Δ_i^j when starting to compute the elements in Δ_i^j, and d_r is the number of times a feature index r appearing in the set Δ_i^j. Because each adjacent area of a $k \times k$ convolution filter covers k^2 number of features, a feature can appear zero time to up to k^2 times in the adjacent area. As a result, the range of the d_r value varies from 0 to k^2.

Substituting (7) into (5), we have

$$\begin{aligned} &\underset{[\boldsymbol{u^{11}}, \boldsymbol{u^{12}}, \ldots, \boldsymbol{u^{(m-k+1)(m-k+1)}}]}{\arg\max} \sum_{i=1}^{m-k+1} \sum_{j=1}^{m-k+1} (\boldsymbol{u^{ij}})^{\mathrm{T}} \boldsymbol{\mathcal{M}} \boldsymbol{u^{ij}} \\ &\text{s.t.} \quad u_r^{ij} = d_r, r \in \boldsymbol{I_i^j}, u_r^{ij} \in \{0, 1, \ldots, k^2\}. \end{aligned} \tag{8}$$

Finding feature reordering matrix $\mathcal{O}$ satisfying (8) is equal to finding feature indexes in each Δ_i^j such that

$$\begin{aligned} &\arg\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}} \boldsymbol{\mathcal{M}} \boldsymbol{u} \\ &\text{s.t.} \quad u_r = d_r, r \in \boldsymbol{I_i^j}, u_r \in \{0, 1, \ldots, k^2\}. \end{aligned} \tag{9}$$

For simplify, we omit the superscript of $\boldsymbol{u}$ in (9). The optimal solutions of feature-selection problem in (9) can only be solved through brute-force search, which is computationally expensive and difficult to implement, especially for datasets with high dimensional features. Accordingly, we propose to reduce the computational complexity of solving (9) by determining the first row in feature reordering matrix $\mathcal{O}$ and adding the constraints on $\sum u_r$ and $\mathcal{M}$. The detail procedures are described as follows.

To reduce the computational complexity and maximize the local and global feature correlations, we propose to determine the first row in $\mathcal{O}$ by utilizing the feature-label correlation vector $\mathcal{L}$, such that the $\mathcal{O}$ is a label-targeting feature reordering matrix. Specifically, we first order the values in $\mathcal{L}$ in descending order to obtain a vector $\mathcal{L}'$. It is easy to obtain the feature indexes order $\mathcal{V}$ corresponding to $\mathcal{L}'$. The feature ordering in the first row of $\mathcal{O}$ is $\mathcal{V} = [\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m]$ as shown in Fig. 5, *i.e.*, $\mathcal{O}_{1,j} = \mathcal{V}_j (1 \le j \le m)$. By doing so, only a part of elements in $\Delta_1^j (1 \le j \le (m-k+1))$ is required to be solved, such that the computation complexity is reduced.

Meanwhile, in order to ensure that all feature values are kept in the new instance representation, so the information in the original instance is maximally preserved, we make each row of the matrix $\mathcal{O}$ contain indexes of all features by adding the constraint on $\sum u_r$ and $\mathcal{M}$, which is detailed at the end of this section. As a result, the original feature-selection problem in (9) is transformed into a 0/1 integer programming problem as follows.

$$\begin{aligned} &\arg\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}} \boldsymbol{\mathcal{M}}_{\boldsymbol{i}}^{\boldsymbol{j}} \boldsymbol{u} \\ &\text{s.t.} \sum_{r, u_r \in \boldsymbol{u}} u_r = |\boldsymbol{I_i^j}| + |\boldsymbol{\alpha_i^j}|, \quad u_r \in \{0, 1\}; \\ &\mathcal{O}_{1,l} = \mathcal{V}_l, l = 1, 2, \ldots, m; \\ &u_r = 1 \ (r \in \boldsymbol{I_i^j}), \end{aligned} \tag{10}$$

where $\boldsymbol{\alpha_i^j}$ is the set of coordinates of unknown elements in Δ_i^j, $|\cdot|$ denotes the cardinality of a set, and $\boldsymbol{\mathcal{M}_i^j}$ is the modified feature correlation matrix which is derived by setting the s^{th} row and column in $\mathcal{M}$ to $\boldsymbol{0}$ ($s \in [\mathcal{O}_{\boldsymbol{\alpha_i^j}1}, \ldots, \mathcal{O}_{\boldsymbol{\alpha_i^j}-1}]$ & $s \notin \boldsymbol{I_i^j}$).

(10) is a standard 0/1 optimization problem. SDP [13], gives an approximate solution to solve this kind of maximization problem. Following SDP problem formulation defined in (10), one can employ publicly available Matlab open source packages to solve (10). In our experiments, we use branch-and-bound algorithm (B&B) [41], which is based on a simple spatial branch-and-bound strategy, to find solutions for (10). The solution $u_r = 1$ ($r \notin \boldsymbol{I_i^j}$) denotes that index r is one of the $|\boldsymbol{\alpha_i^j}|$ unknown elements in Δ_i^j, which is shown in Algorithm 1. By utilizing (10), we derive each adjacent area in $\mathcal{O}$ from top to bottom, left to right, *i.e.*, we first compute the feature indexes in Δ_1^1, then Δ_1^2 and so on.

Table 1
A toy dataset with six instances, five features, and a binary class label.

Instances	f_1	f_2	f_3	f_4	f_5	Label
x_1	0.2	0.3	0.6	0.4	0.1	1
x_2	0.4	0.72	0.3	0.2	0.7	1
x_3	0.13	0.55	0.3	0.1	0.33	0
x_4	0.22	0.42	0.14	0.44	0.11	1
x_5	0.34	0.51	0.48	0.35	0.52	0
x_6	0.28	0.37	0.59	0.27	0.47	0

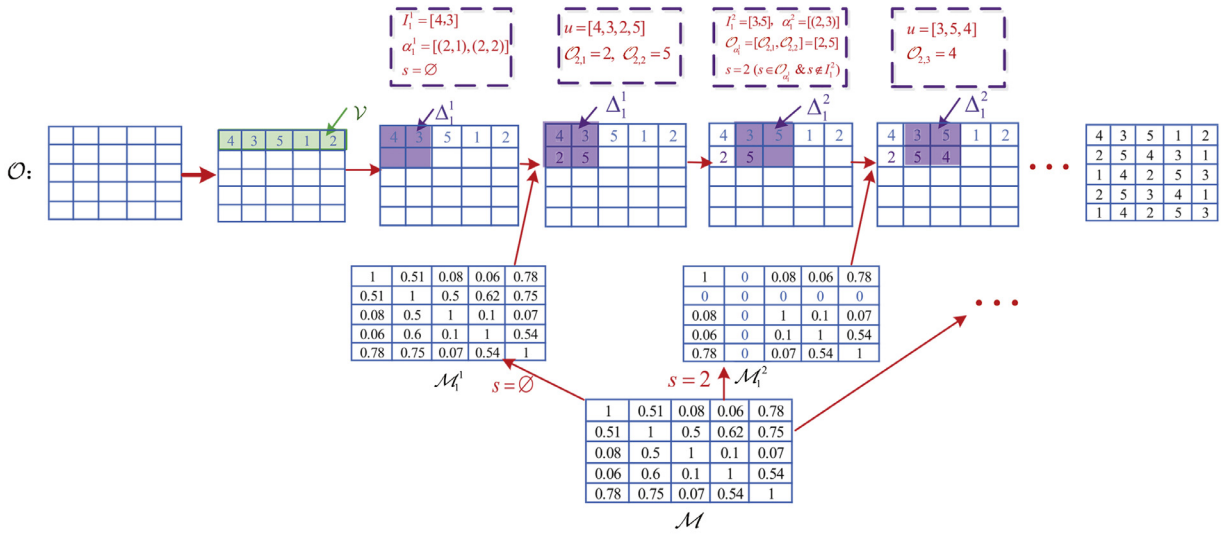

Fig. 6. Example: The process of constructing feature reordering matrix $\mathcal{O}$. The lower panel shows the feature-feature correlation matrix $\mathcal{M}$. The second panel, from left to right, shows the feature reordering matrix $\mathcal{O}$, a 5×5 matrix, which is initially set as an empty matrix. To reduce the computational complexity, the first row of $\mathcal{O}$ is first set by using feature feature-label correlation vector. The remaining adjacent areas, each showing as a 2×2 shaded area, are constructed sequentially to ensure that feature correlation within each adjacent area is maximized.

Now we explain why the constraints on $\sum u_r=|\boldsymbol{I_i^j}|+|\boldsymbol{\alpha_i^j}|$ and $\mathcal{M}_{\boldsymbol{i}}^{\boldsymbol{j}}$ can jointly ensure that each row of $\mathcal{O}$ contains the indexes of all features, *i.e.*, the feature indexes in $\mathcal{O}_{\boldsymbol{\alpha_i^j}}$ are not only different from each other but also different from the feature indexes in $[\mathcal{O}_{\boldsymbol{\alpha_i^1}},\ldots,\mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$. We divide the elements in $[\mathcal{O}_{\boldsymbol{\alpha_i^1}},\ldots,\mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$ into two mutual exclusive subsets, denoted by $\boldsymbol{E_1}$ and $\boldsymbol{E_2}$. Specifically, $\boldsymbol{E_1}$ contains the element d ($d\in[\mathcal{O}_{\boldsymbol{\alpha_i^1}},\ldots,\mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$ & $d\in\boldsymbol{I_i^j}$), and $\boldsymbol{E_2}$ contains the element s ($s\in[\mathcal{O}_{\boldsymbol{\alpha_i^1}},\ldots,\mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$ & $s\notin\boldsymbol{I_i^j}$). One constraint $\mathcal{M}_{\boldsymbol{i}}^{\boldsymbol{j}}$, which is obtained by setting the s^{th} row and column in $\mathcal{M}$ to 0 ($s\in\boldsymbol{E_2}$), ensures that the feature indexes in $\mathcal{O}_{\boldsymbol{\alpha_i^j}}$ are different from the feature indexes in $\boldsymbol{E_2}$. The other constraint, *i.e.*, $\sum u_r=|\boldsymbol{I_i^j}|+|\boldsymbol{\alpha_i^j}|$, ensures that the feature indexes in $\mathcal{O}_{\boldsymbol{\alpha_i^j}}$ are not only different from each other but also different from the feature indexes in $\boldsymbol{I_i^j}$. Due to the fact that $\boldsymbol{E_1}\subseteq\boldsymbol{I_i^j}$, $\sum u_r=|\boldsymbol{I_i^j}|+|\boldsymbol{\alpha_i^j}|$ exactly ensures that the feature indexes in $\mathcal{O}_{\boldsymbol{\alpha_i^j}}$ are different from the feature indexes in $\boldsymbol{E_1}$. As a result, the constraints on $\sum u_r=|\boldsymbol{I_i^j}|+|\boldsymbol{\alpha_i^j}|$ and $\mathcal{M}_{\boldsymbol{i}}^{\boldsymbol{j}}$ can jointly ensure that the feature indexes in $\mathcal{O}_{\boldsymbol{\alpha_i^j}}$ are not only different from each other but also different from the feature indexes in $[\mathcal{O}_{\boldsymbol{\alpha_i^1}},\ldots,\mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$.

4.3. ***Example:*** *synthetic matrix generation*

In this subsection, we use an example to demonstrate the transformation of a generic dataset into synthetic matrix format for deep learning. Table 1 lists a toy dataset with 6 instances, 5 features, and a binary class label.

4.3.1. Feature-feature correlation matrix $\mathcal{M}$ and feature-label correlation vector $\mathcal{L}$

EDLT first creates feature-feature correlation matrix $\mathcal{M}$, as shown in Fig. 6, and feature-label correlation vector $\mathcal{L}=[0.1298, 0.0122, 0.3267, 0.4542, 0.3144]$.

Table 2
The converted synthetic matrix for instance $\boldsymbol{x_1}$: $\mathcal{F}(\boldsymbol{x_1})$.

0.4	0.6	0.1	0.2	0.3
0.3	0.11	0.4	0.6	0.2
0.2	0.4	0.3	0.1	0.6
0.3	0.1	0.6	0.4	0.2
0.2	0.4	0.3	0.1	0.6

4.3.2. The feature reordering matrix construction and synthetic matrix generation

EDLT first orders the values in $\mathcal{L}$ in a descending order to obtain a vector $\mathcal{L}' = [0.4542, 0.3267, 0.3144, 0.1298, 0.0122]$. Then, it is easy to obtain the feature indexes order vector $\mathcal{V} = [4, 3, 5, 1, 2]$ corresponding to $\mathcal{L}'$. The feature ordering in the first row of $\mathcal{O}$ is $\mathcal{V} = [4, 3, 5, 1, 2]$, as shown in Fig. 6.

Assume that the size of the convolution filter is 2×2, there are 16 adjacent areas of 2×2 in $\mathcal{O}$. Then, EDLT derives each adjacent area in $\mathcal{O}$ from top to bottom, left to right. In other words, EDLT derives feature indexes in $\Delta_1^1, \Delta_1^2, \Delta_1^3, \Delta_1^4, \Delta_2^1$, ..., Δ_4^4 in turn. Fig. 6 shows the details of obtaining feature indexes in Δ_1^1 and Δ_1^2, and the elements in other adjacent areas can be derived in the same way.

For Δ_1^1, the known feature indexes in Δ_1^1 is $\boldsymbol{I_1^1} = [4, 3]$, and the set of coordinates of unknown elements in Δ_1^1 is $\boldsymbol{\alpha_1^1} = [(2, 1), (2, 2)]$. Because there is no s satisfying $s \in (\boldsymbol{\alpha_i^0} = \emptyset) \& s \notin (\boldsymbol{I_1^1} = [4, 3])$, we have $\boldsymbol{\mathcal{M}_1^1} = \boldsymbol{\mathcal{M}}$. Substituting $\boldsymbol{I_1^1}$, $\boldsymbol{\mathcal{M}_1^1}$, and $\boldsymbol{\alpha_1^1}$ into (10), EDLT derives the solution $\boldsymbol{u} = [4, 3, 2, 5]$. Because $\boldsymbol{I_1^1}$ is [4,3], the feature reordering $\mathcal{O}_{\boldsymbol{\alpha_1^1}} = [2, 5]$, *i.e.*, $\mathcal{O}_{2,1} = 2$ and $\mathcal{O}_{2,2} = 5$.

For Δ_1^2, the known feature indexes in Δ_1^2 is $\boldsymbol{I_1^2} = [3, 5]$, and the set of coordinates of unknown elements in Δ_1^2 is $\boldsymbol{\alpha_1^2} = [(2, 3)]$. Because $s = 2$ satisfies $s \in (\mathcal{O}_{\boldsymbol{\alpha_1^1}} = [2, 5])$ & $s \notin (\boldsymbol{I_1^1} = [3, 5])$, $\boldsymbol{\mathcal{M}_1^2}$ is obtained by setting the 2th row and column of $\boldsymbol{\mathcal{M}}$ to $\boldsymbol{0}$ as in Fig. 6. Substituting $\boldsymbol{I_1^1}$, $\boldsymbol{\mathcal{M}_1^2}$, and $\boldsymbol{\alpha_1^2}$ into (10), we derive the solution $\boldsymbol{u} = [3, 5, 4]$. Because $\boldsymbol{I_1^1}$ is [3,5], the feature reordering $\mathcal{O}_{\boldsymbol{\alpha_1^2}} = [4]$, *i.e.*, $\mathcal{O}_{2,3} = 4$.

Using similar logics, EDLT can derive feature indexes in other 14 adjacent areas, and the final feature reordering matrix $\mathcal{O}$ is shown in Fig. 6.

Finally, based on Algorithm 2, each instance in the original dataset is converted into a synthetic matrix format. Table 2 shows the converted synthetic matrix representation of instance $\boldsymbol{x_1}$, *i.e.*, $\mathcal{F}(\boldsymbol{x_1})$.

5. Computational complexity analysis

The proposed EDLT algorithm converts each instance from its feature values into a synthetic matrix/image format based on the feature reordering matrix $\mathcal{O}$, such that the CNN can be utilized to learn features from generic data for conventional classification tasks.

The computational complexity of EDLT mainly relies on the computational costs in solving (10) to obtain the unknown feature indexes in each adjacent area Δ_i^j of $\mathcal{O}$, as shown in Fig. 5. Indeed, (10) is a 0/1 integer programming problem, and we utilize the branch and bound (B&B) algorithm to solve this optimization problem. Let $g(\Delta_i^j)$ denote the computational complexity of solving (10) to obtain the unknown feature indexes in adjacent area Δ_i^j. Then, the computational complexity of the EDLT method is

$$\delta = \sum_{i=1}^{m-k+1} \sum_{j=1}^{m-k+1} g(\Delta_i^j) \tag{11}$$

We now discuss the complexity of $g(\Delta_i^j)$. Specifically, for $g(\Delta_i^1)$, when starting to derive the unknown feature indexes in Δ_i^1, the feature indexes in the $k-1$ rows of Δ_i^1 are available. The constraints on (10) ensures that the feature indexes in each row of the feature reordering matrix $\mathcal{O}$ are different from each other, and that the unknown feature indexes to be solved by (10) are different from known feature indexes in Δ_i^1. Therefore, the maximal size of solution space for solving (10) is $m - k$. As a result, we have $g(\Delta_i^1) = O(2^{m-k})$ [41].

Similarly, for $g(\Delta_i^j)(\forall j \geq 2)$, because the feature indexes in each row of the feature reordering matrix $\mathcal{O}$ are different from each other and the unknown feature indexes to be solved by (10) are different from the known feature indexes in Δ_i^j, the maximal size of solution space for solving (10) is $m - k - j + 2$. Therefore, we have $g(\Delta_i^j) = O(2^{m-k-j+2})$. As a result, $g(\Delta_i^j)$ can be calculated by

$$g(\Delta_i^j) = \begin{cases} O(2^{m-k}), & j = 1, \\ O(2^{m-k-j+2}), & j \geq 2. \end{cases} \tag{12}$$

Substituting (12) into (11), we have

$$\delta = O((m-k)2^{m-k}). \tag{13}$$

Table 3
A brief description of the benchmark datasets.

ID	Dataset	Instance	Features	Classes
1	wall-following	5456	24	4
2	vehicle	946	18	4
3	breast tissue	106	9	6
4	vowel	990	9	6
5	ecoli	336	7	8
6	wine quality-red	1599	11	11
7	breast cancer wisconsin (Diagnostic)	569	30	2
8	wine	178	13	3
9	banknote authentication	1372	4	2
10	vertebral column	310	6	2
11	yeast	1484	8	10
12	seeds	210	7	3
13	climate model simulation crashes	540	18	2
14	glass identification	214	9	6
15	leaf	340	14	30
16	plrx	182	12	2
17	pima	768	9	2
18	iris	150	4	3
19	wireless indoor localization	2000	7	4
20	sonar	208	60	2
21	hill-vallay	1212	100	2
22	gas sensor array drift	13910	128	6

In comparisons, when solving the objective function in (9) using brute-force search, we have $g(\Delta_i^j) = 2^m$, which results in the total computational complexity $O(m^2 2^m)$. Therefore, the proposed EDLT algorithm (*i.e.* (10)) can dramatically reduce the computational complexity.

6. Experiments

To validate the performance of EDLT in enabling deep learning for generic classification tasks, we use CNN as the deep learning module and implement a number of baseline using Tensorflow [1] configured with one GPU card for accelerated training. The feature reordering matrix $\mathcal{O}$ in EDLT is calculated using MATLAB R2016b, running on a 64-bit Windows 10 workstation with a 3.5-GHz Intel Core CPU and 128G memory.

We compare the algorithm performance on 22 benchmark datasets from UCI machine learning data repository [34]. In order to interpret features learned from EDLT created synthetic matrix, we utilize natural images from CIFAR-10 dataset [24] for a case study. A brief description of the 22 benchmark datasets is summarized in Table 3. All reported results are based on 10 times 5-fold cross validation with classification accuracy being used as the performance metrics.

6.1. Experimental settings

In our experiments, we utilize CNN as the deep learning method. For fair comparisons, each CNN model contains two convolutional layers with same filter size and each convolutional layer is followed by a max pooling layer. We utilize the leaky relu as the activation function [33], and use the Adam optimizer [22] with a learning rate of 0.001. Features extracted by the CNN model are used to train a single hidden layer dense neural network to classify test data.

6.2. Baseline methods

Because no existing method exists to make deep learning applicable for generic datasets, we implement two baseline approaches, random reordering feature (RR) and label-feature correlation reordering feature (LFC), to compare the efficiency of the feature reordering module in EDLT. Similar to EDLT, both RR and LFC enable deep learning for generic datasets by converting each instance from its feature vector into a synthetic matrix format. The difference between the two baselines and EDLT is the way of constructing feature reordering matrix $\mathcal{O}$.

RR constructs the feature reordering matrix $\mathcal{O}$ by random ordering. Specifically, each row of the feature reordering matrix $\mathcal{O}$ contains all feature, placed in a random order. Because PR uses random order to create a synthetic matrix for each instance, there is no local correlation compared to EDLT. If EDLT outperforms PR, it will indicate that reordering features and their values to create local correlation, like EDLT does, is preferable for deep learning.

LFC constructs the feature reordering matrix $\mathcal{O}$ based on the feature-feature correlation matrix and feature-label correlation vector without considering global correlation maximization. Specifically, each row of the feature reordering matrix $\mathcal{O}$ contains all feature indexes. The first column of $\mathcal{O}$ are feature indexes order vector $\mathcal{V}$. Then, the other feature indexes in the ith row are obtained by ordering the correlation with the first feature in descending order, *i.e.*, the second element in

Table 4
Algorithm performance comparisons (the number of filter of each convolution layer is 32 and 4, respectively. The number of hidden layer of the dense NN is 1).

Dataset	RR		LFC		EDLT	
	2×2	3×3	2×2	3×3	2×2	3×3
wall-following	85.16	87.72	84.538	87.11	88.6	88.25
vehicle	63.15	73.68	65.96	71.048	73.68	78.94
breast tissue	72.72	72.72	74.995	75.45	77.27	77.27
vowel	86.36	93.93	86.559	93.881	86.44	92.67
ecoli	88.23	85.29	85.2903	86.172	85.29	85.29
wine quality-red	61.7	61.29	61.56	61.5	61.25	61.56
breast cancer wisconsin (Diagnostic)	95.6	98.24	96.38	98.15732	97.3684	98.24
wine	97.2	100	96.85	100	100	100
banknote authentication	100	98.9	97.2	99.52	100	100
vertebral column	77.41	75.8	78.84	78.7	78.22	79.03
yeast	59.745	59.59	60.861	61.6	59.59	62.43
seeds	90.47	92.85	92.15	94.27	92.85	92.85
climate model simulation crashes	94.4	95.37	95.17	94.44	95.37	97.22
glass identification	58.1	58.1	56.31	61.85	53.48	58.13
leaf	73.52	74.9	75.729	75.79	77.94	76.47
plrx	56.75	56.75	55.55	54.31	56.75	59.45
pima	80.51	77.27	79.4	79.66	79.87	79.2
iris	100	100	98.3	98.3	100	100
wireless indoor localization	96.75	98.5	96.5	98.4	96	98.75
sonar	90.47	90.47	89.41	90.47	90.47	83.33
hill-vally	61.31	58.29	62.13	62.5	69.26	71.87
gas sensor array drift	99.08	99.06	98.83	99.3	99.35	99.2
Average	81.24	82.65	81.32	82.42	82.64	83.55

the ith row is the index of the feature with the largest correlation with the first feature in the ith row, and so on. Obviously LFC only considers a local correlation between the first selected feature and the remaining features, while EDLT considers both global and local correlation maximization.

For comparison purposes, we also compare the algorithm performance using popular machine learning methods, including k-nearest neighbors algorithm (k-NN), support vector machine (SVM), Decision tree learning (DT), dense neural network (NN)), by utilizing the sklearn module in tensorflow [36]. Furthermore, we also compare the performance with the stochastic configuration networks (SCNs), which is a randomized learning method.

***k*-NN** is a non-parametric method to find the instance's n nearest neighbors in the training set [2]. The k-NN classifies the instance as the class that the most n nearest neighbors belong to.

SVM is a supervised learning model to construct a hyperplane with largest margin to separate instances into different classes. Using kennel functions [11], instances are mapped into a high-dimensional space and are classified into different classes, depending on the side of the hyperplane the instances falling into.

DT is to predict the label of the instance by constructing a decision tree where leaves represent class labels and branches represent conjunctions of features [6].

NN represents a dense layer neural network, which is used to approximate some complex function [10]. The information flows from the input layer through all hidden layers to the output layer to predict the label of test data. In our experiments, we use a dense network, so all hidden nodes are connected to the hidden nodes of the next layer or all output layer nodes. We use a dense network with 1 hidden layer or 2 hidden layers in our experiments.

SCNs are randomised learning algorithms for single layer feed-forward neural networks. SCNs randomly assigns input weight values and biases with a supervisory mechanism, and evaluate the output weights in either constructive or selective manner [45–47]. We utilize a SC-III [47] algorithm to assign the random parameters in our experiments.

In the following sections, we first study feature reordering method performance in different parameter settings, including different convolution filter sizes, the number of convolution filters, and the different number of hidden layers in the dense neural network. After that, we report the accuracy performance on all benchmark datasets. Finally, we report a case study to show effectiveness of EDLT in creating new instance representation for CNN to learn features for generic data.

6.3. Feature reordering method comparisons

For all experiments, unless specified otherwise, the parameter settings are as follows. The size of the convolution filter is set to 2×2 and 3×3, and the number of convolution filters is set to (32,4) and (32,16). For dense neural networks (NN), we consider two structures, *i.e.*, a NN with one hidden layer and a NN with two hidden layers. We set the number of nodes in the hidden layer of the two NN structures to 100 and (100,50), respectively.

Tables 4–7 report detailed comparisons of different feature reordering methods, where the results are based on different number of convolution filters and different number of hidden layers in the NN. Specifically, in Tables 4 and 5, we report the

Table 5
Algorithm performance comparisons (the number of filter of each convolution layer is 32 and 16, respectively. The number of hidden layer of the dense NN is 1).

Dataset	RR		LFC		EDLT	
	2×2	3×3	2×2	3×3	2×2	3×3
wall-following	90.03	90.42	90.07	90.256	92.65	91.75
vehicle	73.68	73.68	73.88	73.18	73.68	84.21
breast tissue	77.27	77.27	75.75	77.27	77.27	77.27
vowel	95.45	96.46	95.57	96	95.3	97.06
ecoli	81.93	82.35	85.1	83.33	85.29	85.29
wine quality-red	61.02	60.31	60	60.53	61.83	60.98
breast cancer wisconsin (Diagnostic)	97.36	97.36	97.43	98.76	97.3684	98.24
wine	100	100	99.12	100	100	100
banknote authentication	100	100	100	100	100	100
vertebral column	79.03	75.8	78.72	78.56	80.51	81.84
yeast	60.64	60.6	60.68	59.97	61.27	60.04
seeds	90.47	92.85	91.65	91.69	90.47	90.47
climate model simulation crashes	95.76	96.29	93.3	95.54	94.44	97.22
glass identification	58.1	62.32	60.75	59.95	59.12	63.95
leaf	76.89	73.52	75.97	72.79	76.47	74.7
plrx	54.82	56.75	53.27	59.06	52.35	60.53
pima	79.87	79.22	78.78	78.56	79.79	78.83
iris	100	100	100	100	100	100
wireless indoor localization	96.75	98.5	96.81	98.16	97.25	98.5
sonar	88.09	88.09	90.17	90.16	95.23	90.47
hill-vally	56.52	59.625	61.54	62.254	66.3	76.18
gas sensor array drift	99.04	99.2	98.95	99.3	99.35	99.45
Average	82.43	82.83	82.58	82.87	83.45	84.86

Table 6
Algorithm performance comparisons (the number of filter of each convolution layer is 32 and 4, respectively. The number of hidden layer of the dense NN is 2).

Dataset	RR		LFC		EDLT	
	2×2	3×3	2×2	3×3	2×2	3×3
wall-following	87.74	88.1	87.27	89.56	90.65	89.03
vehicle	71.04	77.1	73.68	78.94	73.68	89.47
breast tissue	75.32	75.32	72.72	77.27	77.27	77.27
vowel	88.54	94.75	92.42	95.95	85.1	92.79
ecoli	82.71	83.98	85.29	82.35	79.41	83.82
wine quality-red	61.09	61.96	58.35	60.1	60.56	61.48
breast cancer wisconsin (Diagnostic)	97.1	98.06	96.49	97.36	97.36	98.24
wine	97.6	100	97.2	100	100	100
banknote authentication	100	100	100	100	100	100
vertebral column	78.79	79.83	77.41	80.64	75.8	82.25
yeast	60.88	60.6	61.44	61.72	60.6	61.06
seeds	90.17	91.95	88.09	90.47	90.47	90.47
climate model simulation crashes	95.24	95.82	97.2	95.37	94.44	97.22
glass identification	61.45	61.33	65.11	65.11	66.5	60.46
leaf	70.79	71.68	64.7	70.58	73.52	74.1
plrx	53.27	56.75	56.75	59.45	51.35	62.16
pima	79.12	78.78	79.87	78.24	81.81	79.04
iris	100	100	100	100	100	100
wireless indoor localization	97.03	98.16	98	98.5	96.125	98.5
sonar	89.87	89.11	92.85	85.71	92.85	90.47
hill-vally	57.95	57.02	60.37	62.87	67.48	69.33
gas sensor array drift	99.13	98.91	98.85	99.25	99.18	99.39
Average	81.58	82.69	82.00	83.15	82.46	84.38

accuracy performance of different feature reordering methods on 22 benchmark datasets using one hidden layer dense NN for final classification. The results of two hidden layer dense NN are reported in Tables 6 and 7.

The results from Tables 4–7 show that EDLT has the best performance gain across different parameter settings, confirming that reordering features and their values to create local and global correlations, like EDLT does, will result in good performance for deep learning to be used for generic data. While LFC only considers local correlation, RR does not consider any correlation in the synthetic matrix. The local correlation is the key for the CNN to learn meaningful features for classification.

Table 7
Algorithm performance comparisons (the number of filter of each convolution layer is 32 and 16, respectively. The number of hidden layer of the dense NN is 2).

Dataset	RR		LFC		EDLT	
	2×2	3×3	2×2	3×3	2×2	3×3
wall-following	90.67	90.398	90.67	90.361	92.1	91.13
vehicle	70.17	71.92	74.73	78.94	73.68	84.21
breast tissue	76.74	75.32	72.72	72.72	77.27	77.27
vowel	94.79	95.99	94.35	96.96	92.72	96.96
ecoli	83.65	81.72	77.52	83.82	85.29	83.08
wine quality-red	61.71	61.6	63.75	63.43	62.5	62
breast cancer wisconsin (Diagnostic)	97.5	98.42	98.24	97.63	97.36	98.24
wine	100	100	100	100	100	100
banknote authentication	100	100	100	100	100	100
vertebral column	80.84	79.55	80.28	80.17	83.87	82.25
yeast	60.48	59.46	60.74	59.93	58.86	59.81
seeds	91.95	92.85	90.47	90.47	90.47	90.47
climate model simulation crashes	95.26	95.83	95.6	94.44	95.6	97.22
glass identification	57.55	58.59	63.66	60.46	59.79	62.78
leaf	74.75	71.68	72.78	72.05	80.88	72.3
plrx	55.98	58.29	53.27	59.45	55.4	56.75
pima	78.67	79.21	80.51	79.54	80.27	78.69
iris	100	100	100	100	100	100
wireless indoor localization	97.6	98	97.43	97.5	96.625	97.5
sonar	90.47	90.94	88.09	88.09	88.09	88.09
hill-vally	59.76	60.159	60.28	60.48	71.87	67.94
gas sensor array drift	99.17	99.19	99.13	99.25	99.33	99.31
Average	82.62	82.68	82.62	82.98	83.72	83.90

In Table 4, compared to RR and LFC, the average performance gains of EDLT are 1.4% and 1.32% for 2×2 filter, and 0.9% and 0.91% for 3×3 filter, respectively. In Table 5, compared to RR and LFC, the average performance gains of EDLT 1.02% and 0.87% for 2×2 filter, and 2.03% and 1.99% for 3×3 filter. In Table 6, the average performance gains of EDLT are 0.88% and 0.46% for 2×2 filter, and 1.69% and 1.23% for 3×3 filter, and in Table 7, the average performance gains of EDLT are 1.1% and 1.1% for 2×2 filter, and 1.22% and 0.92% for 3×3 filter.

Furthermore, the best performance gain of EDLT is higher than those of RR and LFC. To be specific, the best performance gain of EDLT , RR and LFC are 84.86% in Table 5, 82.83% in Table 5, and 83.15% in Table 6, respectively.

The results in Tables 4–7 assert that the feature reordering matrix in EDLT does play an effective role for CNN to leverage local and global correlation to learn effective features.

The experiments show that, for a fixed NN structure, the best CNN learning results for EDLT are obtained by setting the size of the convolution filter to 3×3, the number of convolution filter to (32,16). Therefore, in the following subsections, we utilize this CNN structure settings in the experiments.

6.4. Detailed comparisons of machine learning methods

In Table 8, we report detailed comparisons between EDLT and popular learning methods (*i.e., k*-NN, SVM, DT, NN, and SCNs) on 22 generic benchmark datasets, where NN-*i* and EDLT-*i* mean that the number of hidden layer in the dense NN is *i*.

In our experiments, we set the number of nearest neighbors *n* to 5 for *k*-NN, and use the linear kennel function for SVM. We use CART (Classification And Regression Tree) algorithm to generate a decision tree for DT, and employ NN-*i* as the NN module in EDLT-*i*. We also consider two kinds of NN structures with one and two hidden layers, respectively, and we set the number of nodes in the hidden layer of these two NN module to 100 and (100,50), respectively.

The results from Table 8 show that among all methods, EDLT achieves the best performance gain. Compared to *k*-NN, SVM, DT, NN-1, NN-2, and SCNs, the performance gains of EDLT-1 are 7.9%, 6.99%, 6.12%, 6.18%, 2.34%, and 4.99% respectively.

Comparing randomized learning algorithms (*i.e.* SCNs) to deterministic-heuristic based learning approaches (*i.e., k*-NN, SVM, DT, and NN-1), the results from Table 8 show that SCNs outperform *k*-NN, SVM, DT, and NN-1. The performance gains of the SCNs, compared to each respective method, are 2.91%, 2%, 1.13%, and 1.19%. This demonstrates that randomized learning algorithms are more effective than deterministic-heuristic based learning approaches for the underlying benchmark datasets.

Interestingly, the results show that the performance gain of EDLT-*i* is superior to that of NN-*i*, confirming that features learned by utilizing the CNN are more effective than the original features in the generic dataset for classification. This confirms that converting each instance in the generic dataset into a synthetic matrix/image, like EDLT does, and further applying CNN to the converted data, can indeed lead to a better classification accuracy for generic classification tasks.

Table 8
Comparisons between EDLT and different machine learning methods on benchmark datasets.

Dataset	*k*-NN	SVM	DT	NN-1	NN-2	SCNs	EDLT-1	EDLT-2
wall-following	85.1	78.8	99.7	67.94	83.42	69.56	91.75	91.13
vehicle	52.63	63.1	57.89	52.63	63.15	73.68	84.21	84.21
breast tissue	45.45	68	72.72	77.27	77.27	61.59	77.27	77.27
vowel	93.43	96.96	75.75	65.65	83.3	74.1	97.06	96.96
ecoli	85.29	80.8	82.35	89.7	88.23	85.87	85.29	83.08
wine quality-red	58.125	62.8	67.5	62.5	64.06	61.4	60.98	62
breast cancer wisconsin (Diagnostic)	95.6	97.36	91.22	97.36	97.36	96.87	98.24	98.24
wine	80.5	100	97.2	100	100	100	100	100
banknote authentication	100	97.8	99.27	94.54	100	100	100	100
vertebral column	77.41	74.19	77.41	74.19	79.03	75.97	81.84	82.25
yeast	54.2	58.24	49.83	58.24	60.94	60.53	60.04	59.81
seeds	85.7	85.7	95.23	95.23	95.23	95.87	90.47	90.47
climate model simulation crashes	96.29	95.37	88.8	94.4	98.1	97.01	97.22	97.22
glass identification	58.13	58.13	51.16	53.48	60.46	59.15	63.95	62.78
leaf	38.23	32.35	41.17	72.05	76.47	66.17	74.7	72.3
plrx	54.05	54.05	54.05	54.05	54.05	54	60.53	56.75
pima	77.9	82.46	78.57	79.87	80.51	80.1	78.83	78.69
iris	100	100	100	90	96.67	92.12	100	100
wireless indoor localization	97.5	96.75	96.25	96.25	97.25	95.27	98.5	97.5
sonar	83.3	83.3	64.28	88.09	85.71	84.82	90.47	88.09
hill-vally	75.2	50	94.3	70.12	75.72	79.204	76.18	67.94
gas sensor array drift	99.18	96.94	97.76	97.5	93.9	98.7	99.45	99.31
Average	76.96	77.87	78.74	78.68	82.52	79.872	84.86	83.9

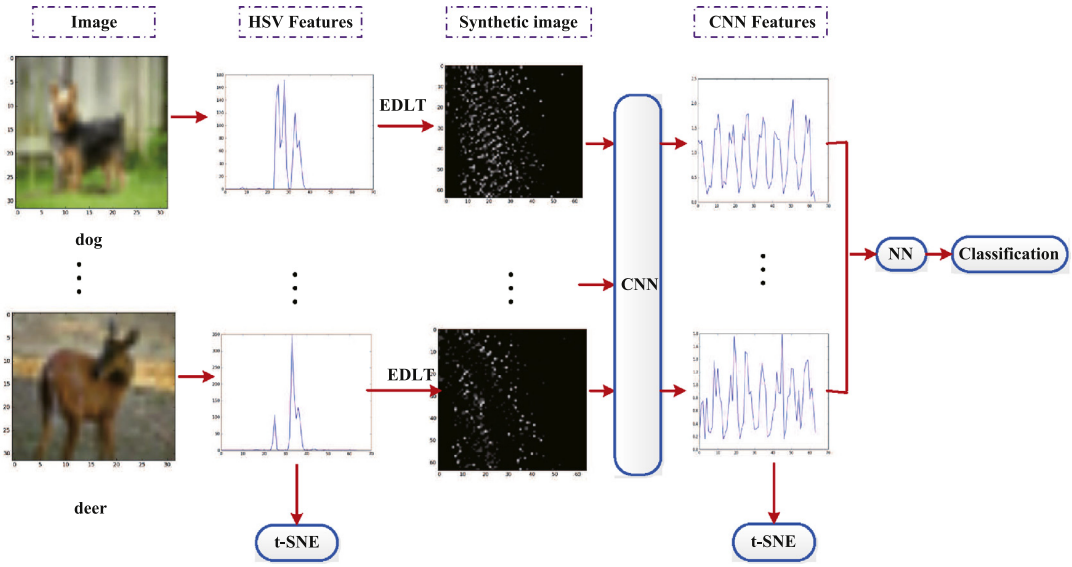

Fig. 7. Comparisons between original HSV features and CNN features learned from EDLT converted matrix/image ("Dog" *vs.* "Deer").

6.5. Case study of EDLT converted matrix/image and features

In this subsection, we carry out a case study to compare features in the original space *vs.* features learned from EDLT converted matrix/image. Our goal is to understand what the deep learning is learning from the synthetic matrix, compared to the original features.

In our experiments, we utilize the natural images collected from CIFAR-10 dataset, a common benchmark task for object recognition, and extract 64-dimensional Hue-Saturation-Value (HSV) features [7] as the original features to represent each image. We use image as the test bed, because we can compare images, against their features, to validate the algorithm performance. It's worth noting that we are not trying to find best features to represent an image, but to understand how EDLT converts the original features to support deep learning. Therefore, we only use HSV features in our case study.

As shown in Figs. 7, 8, and 9, we first extract HSV features from each image. After that, the HSV features are used to generate a synthetic image by using EDLT. Then, the synthetic images are fed to a CNN model to learn new features, which are used to train a dense NN for classification. Finally, we use *t*-SNE tool [42] to compare the difference between the HSV

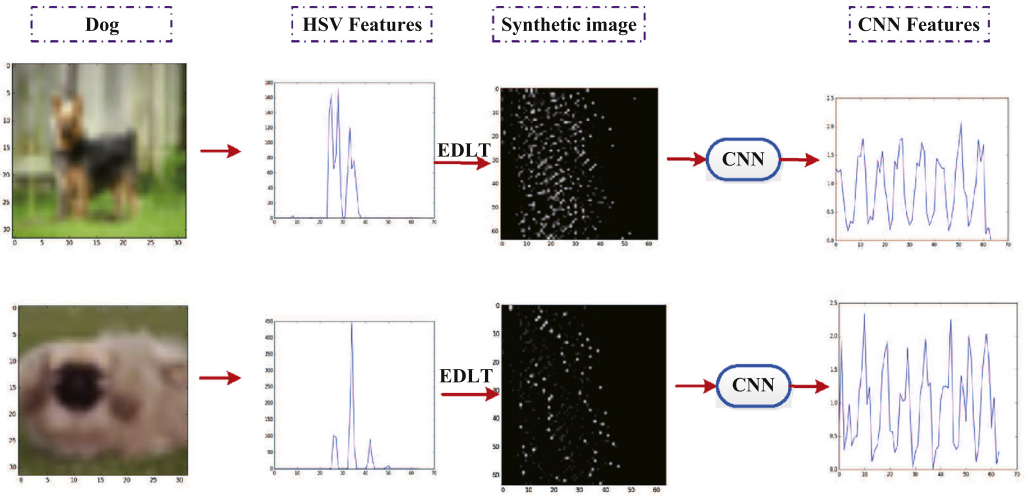

Fig. 8. Comparisons between original HSV features and CNN features learned from EDLT converted matrix/image ("Dog").

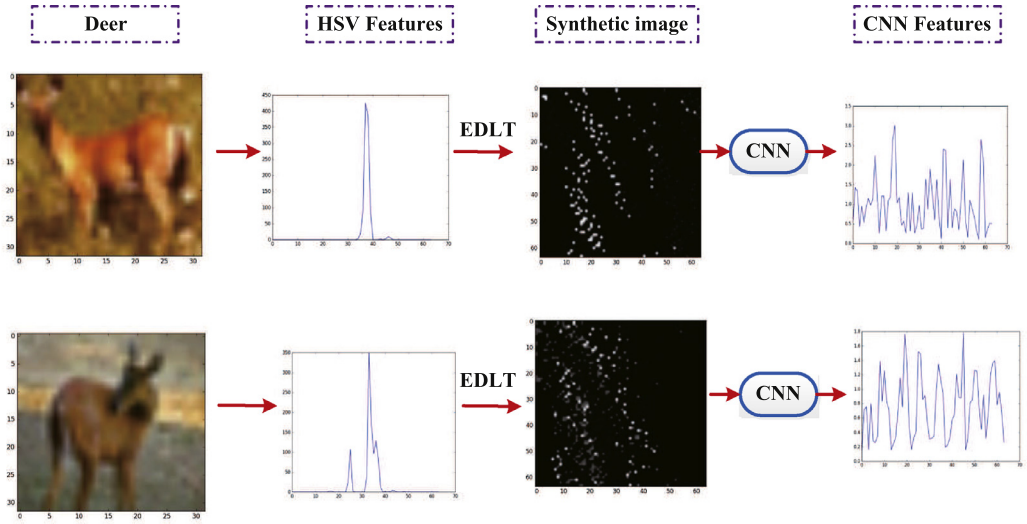

Fig. 9. Comparisons between original HSV features and CNN features learned from EDLT converted matrix/image ("Deer").

features and the features learned by CNN to observe what exactly the CNN is learning from the synthetic image (In the CNN model, we use 3 convolutional layers with filter size 3×3. The number of converlution filter are 1632 and 1).

Fig. 7 shows the feature comparisons between a "Dog" *vs.* a "Deer". It shows that the HSV features in the original feature space are not discriminative to differentiate "Dog" *vs.* "Deer" (noticing high spike overlapping between two HSV maps). By applying EDLT to convert each HSV represented instance, the synthetic image shows a better separation between two categories. This difference is further captured by the CNN to learn distinct features to better classify "Dog" *vs.* "Deer".

In Figs. 10 and 11, we further report the $t-$SNE feature representation results between original HSV features and the EDLT converted features. Fig. 10 shows that the CNN feature has better discrimination than the HSV feature for the two categories. Meanwhile, the results from Fig. 11 show that the three categories almost completely overlap with each other in the HSV feature space, but are roughly distinguishable in the CNN feature space. This asserts that EDLT converted features provide better feature representation for classification.

Table 9 further presents the performance gain for two and three categories by utilizing different machine learning methods. The settings of k-NN, SVM, DT, NN-1, NN-2, and SCNs are the same as that of Table 8. The results show that EDLT achieves the best performance gain. Compared to k-NN, SVM, DT, NN-1, NN-2, and SCNs, the performance gains of EDLT are 2.18%, 0.98%, 5.83%, 1.05%, 1.14%, and 1.6% respectively, for binary class classification, and 3.6%, 2.45%, 8.45%, 2.6%, 1.27% and 2.63% , respectively, for three class classification.

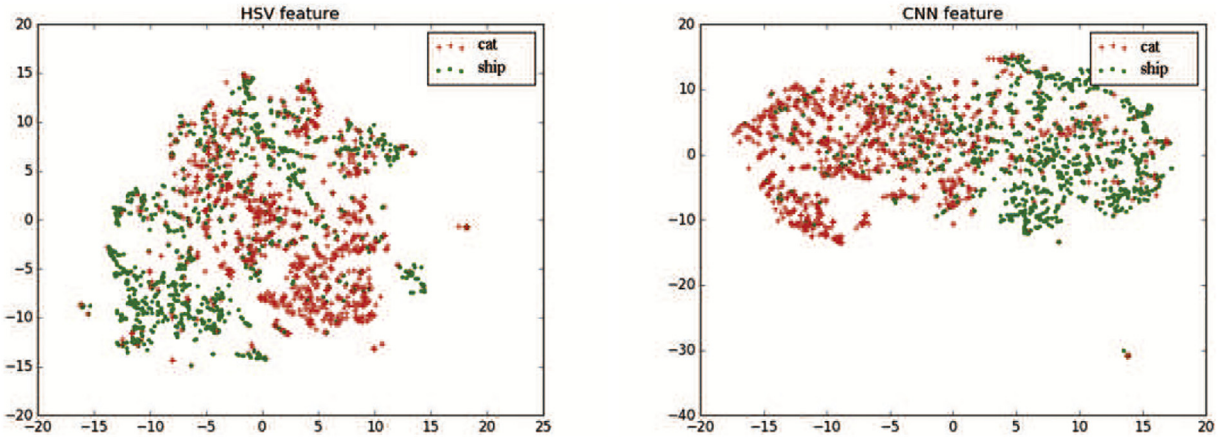

Fig. 10. Comparisons between original features (HSV) *vs.* EDLT converted CNN features("Cat" *vs.* "Ship").

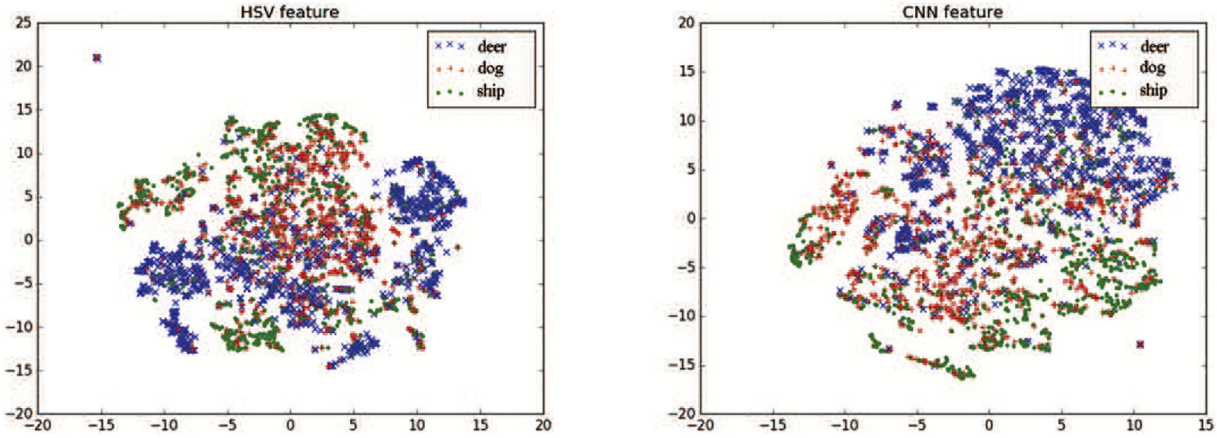

Fig. 11. Comparisons between original features (HSV) *vs.* EDLT converted CNN features("Deer", "dog", and "Ship").

Table 9
Detailed comparisons of machine learning methods on benchmark datasets.

Classification Tasks	*k*-NN	SVM	DT	NN-1	NN-2	SCNs	EDLT
(Bird, Horse)	79.95	81.15	76.3	81.08	80.99	80.53	82.13
(Cat, Deer, Horse)	66.2	67.35	61.35	67.153	68.53	67.17	69.8

7. Conclusion

In this paper, we proposed to enable deep learning for generic data classification where data used for learning are not images/videos, like most deep learning methods require, but are already represented in instance-feature tabular format. We argued that local and global correlations are the key to enable deep learning for generic data, and our goal is to convert each instance into suitable format for deep learning method to be directly applied for learning and classification. To achieve the goal, we proposed to reorder each instance's features and their values as a matrix format with maximum local and global correlations. The proposed method, EDLT, first builds the feature-feature correlation matrix and feature-label correlation vector, and uses 0/1 optimization to obtain a feature reordering matrix such that features with strong correlations are adjacent to each other. After each instance is converted from its instance-feature tabular format into a synthetic matrix format, deep learning methods are applied to learn meaningful features for classification. Experiments and comparisons on 22 benchmark datasets confirm that enabling deep learning to generic datasets has clear performance gain, compared to classifiers learned from original feature space, including decision tress, support vector machines, stochastic configuration networks, *etc.* This research opens opportunities for deep learning to be broadly applied to generic datasets for classification.

Acknowledgment

This research is supported by the US National Science Foundation (NSF) through grant IIS-1763452. Natural Science Foundation of China under Grant 61671345.

References

[1] M. Abadi, A. Agarwal, P. Barham, Tensorflow: large-scale machine learning on heterogeneous systems, arXiv:1603.04467 (2016).
[2] K. Anil, On optimum choice of k in nearest neighbour classification, Computation. Stat. Data An. 50 (11) (2006) 3113–3123.
[3] Y. Bengio, O. Delalleau, N.L. Roux, The curse of highly variable functions for local kernel machines, in: Advances in Neural Information Processing Systems, British Columbia, Canada, MIT Press, 2005, pp. 107–114.
[4] Y. Bengio, P. Simard, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (1994) 157–166.
[5] A. Blum, P. Langley, Selection of relevant features and examples in machine learning, Artif. Int. 97 (1–2) (1997) 245–271.
[6] C.E. Brodley, P.E. Utgoff, Multivariate decision trees, Mach. Learn. 19 (1995) 45–77.
[7] R. Brunelli, O. Mich, Histograms analysis for image retrieval, Pattern Recogn. 34 (8) (2001) 1625–1637.
[8] J.-F. Chen, W.-L. Chen, C.-P. Huang, S.-H. Huang, A.-P. Chen, Financial time-series data analysis using deep convolutional neural networks, in: IEEE International Conference on Cloud Computing and Big Data, Macau, China, 2016, pp. 87–92.
[9] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: IEEE Conference on Computer Vision and Pattern Recognition, Rhode Island, USA, 2012, pp. 3642–3649.
[10] C.M. Bishop, Neural networks for pattern recognition, Oxford University Press, Oxford, UK, 1995.
[11] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297.
[12] X. Ding, Y. Zhang, T. Liu, J. Duan, Deep learning for event-driven stock prediction, in: Morgan Kaufmann International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 2015, pp. 2327–2333.
[13] M. Goemans, D. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, Quaest. Geographicae 42 (6) (1995) 1115–1145.
[14] A. Graves, A.-R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, Canada, 2013, pp. 6645–6649.
[15] A. Graves, A.-R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, Canada, 2013, pp. 6645–6649.
[16] X. Guo, S. Singh, H. Lee, R.L. Lewis, X. Wang, Deep learning for real-time atari game play using offline monte-carlo tree search planning, in: Advance in Neural Information Processing Systems Conference, Montreal, Canada, MIT press, 2014, pp. 3338–3346.
[17] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Mach. Learn. Res. 3 (2003) 1157–1182.
[18] M.F.A. Hady, F. Schwenker, Semi-supervised learning,in handbook on neural information processing, Springer, Berlin, Germany, 2013.
[19] J. Hauke, T. Kossowski, Comparison of values of Pearson's and Spearman's correlation coefficient on the same sets of data, Quaest. Geographicae 31 (2) (2011) 87–93.
[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[21] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, Learning convolutional feature hierarchies for visual recognition, in: Advance in Neural Information Processing Systems Conference, Vancouver, B.C., Canada, MIT press, 2010, pp. 1090–1098.
[22] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: IEEE International Conference on Learning Representations, San Diego, CA, USA, 2015, pp. 1–13.
[23] R. Kohavi, G. John, Wrappers for feature subset selection, Artif. Int. 97 (12) (1997) 273–324.
[24] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, Dept. of Computer Science, Univ. of Toronto, Toronto, 2012 Master's thesis.
[25] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advance in Neural Information Processing Systems Conference, Lake Tahoe, Nevada, USA, MIT press, 2012, pp. 1106–1114.
[26] L. Ladla, T. Deepa, Feature selection methods and algorithms, Int. J. Comput. S. Eng. 3 (5) (2011) 1787–1797.
[27] P. Langley, Selection of relevant features in machine learning, in: AAAI Fall Symposium on Relevance, New Orleans, Louisiana, 1994, pp. 140–144.
[28] D. Langlois, S. Chartier, D. Gosselin, An introduction to independent component analysis: infomax and fastica algorithms, Tutorials in Quant. Meth. Psychol. 6 (1) (2010) 31–38.
[29] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[30] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in Neural Information Processing Systems, Vancouver, Canada, MIT Press, 1990, pp. 396–404.
[31] H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: ACM International Conference On Machine Learning, Montreal, Canada, 2009, pp. 609–616.
[32] D. Lunga, S. Prasad, M.M. Crawford, O. Ersoy, Manifold learning-based feature extraction for classification of hyperspectral data: a review of advances in manifold learning, IEEE Signal Proc. Mag. 31 (1) (2014) 55–66.
[33] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: ACM International Conference on Machine Learning, Atlanta, USA, 2013, pp. 3371–3408.
[34] D. Newman, S. Hettich, C. Blake, C. Merz, Uci repository of machine learning databases, irvine, 1998.
[35] Y.-H. Pao, G.-H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.
[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, Scikit-learn: machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[37] V. Rokhlin, A. Szlam, M. Tygert, A randomized algorithm for principal component analysis, SIAM J. Matrix Anal. A. 31 (3) (2009) 1100–1124.
[38] S. Scardapane, D. Wang, Randomness in neural networks: an overview, WIREs Data Min. Knowl. Discovery (2017), doi:10.1002/widm.1200.
[39] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[40] B. Scholkopft, K.-R. Mullert, Neural Networks for Signal Processing, Springer, 1999.
[41] L. Sven, Integrating sqp and branch-and-bound for mixed integer nonlinear programming, Comput. Optim. Appl. 18 (3) (1998) 1115–1145.
[42] L. van der Maaten, G. Hinton, Visualizing data using t-sne, Mach. Learn. Res. 9 (2008) 2579–2605.
[43] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: ACM International Conference on Machine Learning, Stockholm, Sweden, 2008, pp. 1096–1103.
[44] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, in: ACM International Conference on Machine Learning, Haifa, Israel, 2010, pp. 3371–3408.
[45] D. Wang, C. Cui, Stochastic configuration networks ensemble with heterogeneous features for large-scale data analytics, Inf. Sci. 417(55–71) 2017.
[46] D. Wang, M. Li, Robust stochastic configuration networks with kernel density estimation for uncertain data regression, IEEE Trans. Cybern. 412–413 (2017) 210–222.
[47] D. Wang, M. Li, Stochastic configuration networks: fundamentals and algorithms, IEEE Trans. Cybern. 47 (10) (2017) 3466–3479.
[48] D. Zhang, J. Wang, F. Wang, C. Zhang, Semi-supervised classification with universum, in: SIAM International Conference on Data Mining, San Diego, CA, 2008, pp. 323–333.
[49] X. Zhu, Cross-domain semi-supervised learning using feature formulation, IEEE Trans. Syst. Man Cybern. Part B 41 (6) (2011) 1627–1638.