EDLT: Enabling Deep Learning for Generic Data Classification

Huimei Han[†,‡], Xingquan Zhu[‡], and Ying Li[†]
[‡] *School of Telecommunication Engineering, Xidian University, Xi'an, Shaanxi 710071, P.R. China*
Dept. of Computer & Elec. Eng. and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA
{hanh, xzhu3}@fau.edu; yli@mail.xidian.edu.cn

***Abstract*—This paper proposes to enable deep learning for generic machine learning tasks. Our goal is to allow deep learning to be applied to data which are already represented in instance-feature tabular format for a better classification accuracy. Because deep learning relies on spatial/temporal correlation to learn new feature representation, our theme is to convert each instance of the original dataset into a synthetic matrix format to take the full advantage of the feature learning power of deep learning methods. To maximize the correlation of the matrix , we use 0/1 optimization to reorder features such that the ones with strong correlations are adjacent to each other. By using a two dimensional feature reordering, we are able to create a synthetic matrix, as an image, to represent each instance. Because the synthetic image preserves the original feature values and data correlation, existing deep learning algorithms, such as convolutional neural networks (CNN), can be applied to learn effective features for classification. Our experiments on 20 generic datasets, using CNN as the deep learning classifier, confirm that enabling deep learning to generic datasets has clear performance gain, compared to generic machine learning methods. In addition, the proposed method consistently outperforms simple baselines of using CNN for generic dataset. As a result, our research allows deep learning to be broadly applied to generic datasets for learning and classification (Algorithm source code is available at http://github.com/hhmzwc/EDLT).**

***Index Terms*—Deep learning; feature learning; convolutional neural networks; classification**

I. Introduction

Deep learning represents a series of layered neural network structures with tuneable weight values learned from training data [1], [2]. Due to its superb accuracy and feature learning capability, deep learning has been successfully used in many real-world applications, especially in image/video/audio recognition, time series, and financial data analysis [3],[4]. For all these domains, the temporal and/or spatial correlation of the data allow deep learning methods to learn effective feature to represent data for better classification. Methods, such as convolution neural networks (CNN) or long-short term memory units (LSTM), all utilize data correlation to learn better feature representation for deep learning [5], [6], [7], [8], [9]. Take CNN as an example, when applying CNN to an image, the convolution procedure is equivalent to a spatial filtering process to learn new features, such as corners or edges, for image recognition [10], [11], [12]. Similarly, one dimensional CNN has also been used to data with temporal correlations, such as stock index [13], [14], with convolution being applied to learn meaningful patterns in the data.

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

In reality, traditional machine learning considers 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, machine learning methods typically do not consider feature/data correlation in the learning process. Nearly all traditional machine learning methods do not explicitly consider feature interactions for learning, and leave feature correlations to be handled by the data processing process to create independent features before applying machine learning methods. For example, feature extraction, such as principle component analysis or manifold learning [15], [16], [17], learns a low dimensional feature representation of the original data better fit to the underlying learning algorithms. Such feature extraction 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 above dilemma raises simple questions on (1) whether deep learning is still effective for generic data, and (2) 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 wound't be reasonable. For example, one can consider each instance as a one dimensional vector and apply 1-D CNN to learn a new feature representation. This simple approach, however, leaves big concerns on what exactly the CNN is learning from the data, and what are the meaning of the feature learned from this process. More fundamentally, enabling deep learning to generic data classification has the following three major challenges:

- ← **Higher Order Feature Correlation and Ordering:** Given a generic data set represented in instance-feature format, features in the data have mutual or higher order correlation. 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 features is critical for deep learning to learn effective features.

978-1-5386-9159-5/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDM.2018.00030

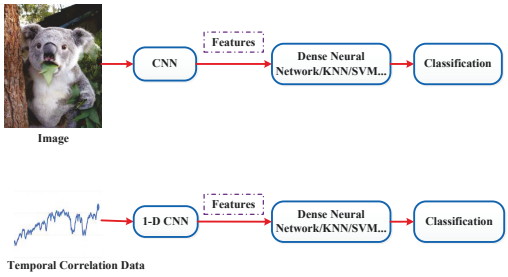

Fig. 1. Typical routines of applying deep learning to image or data with temporal correlations for classification.

Fig. 2. Typical routine of machine learning for generic data classification, where input data are represented in instance-feature tabular format.

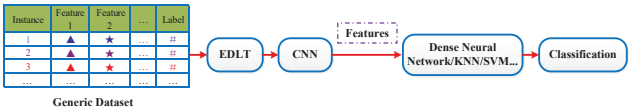

Fig. 3. The routine of the proposed EDLT which enables deep learning for generic data, which are represented in instance-feature tabular format.

- **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.
- **Theoretical Modeling:** We need to find theoretical basis to ensure that each instance in the original dataset is represented by a new representation with strong correlations for deep learning.

The above observations motivate our research to enable deep learning for generic classification tasks as shown in Fig. 3, where learning is not limited to audio/visual data, like most deep learning methods have been commonly applied to.

In order to handle higher order feature correlation, we propose to utilize Pearson correlation to obtain feature-feature and feature-label correlations to reorder features and create local spatial data correlation. To tackle the second challenge (deep learning compatible instance representation), we create a synthetic matrix to represent each instance where 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 maintain maxim local correlation, but also have maximum global correlation.

The remainder of the paper is organized as follows. Section II introduces the problem definition and system overview. The details of the EDLT algorithm are elaborated in Section III. The experiments and conclusion are given in Section IV and Section V, respectively.

II. Problem Definition and System Overview

A. Problem Defination

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, respectively. The **aim** of EDLT is to find a new representation for $\boldsymbol{x_t}$, denoted by $\mathcal{F}(\boldsymbol{x_i})$, such that deep learning methods can be directly applied to $\mathcal{M}(\boldsymbol{x_t})$ to learn features for better classification accuracy, compared to classifiers trained from the original feature domains. In this paper, we use CNN as the deep learning methods, and will compare CNN with generic machine learning classifiers, including k-NN, support vector machines (SVM), decision trees (DT), and multi-layer neural networks (NN).

Because deep learning uses temporal and/or spatial correlation of the data to learn effective feature for classification, our motivation is to create "artificial correlation" and represent each instance $\boldsymbol{x_t}$ as a synthetic matrix where features with strong correlations are adjacent to each other. By doing so, the new synthetic matrix serves as a "synthetic image" for a deep learning module, such as CNN model, to further learn effective features for classification.

B. Overall Framework of EDLT

Fig. 4 lists the overall framework of the proposed EDLT 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 correlation matrix $\boldsymbol{\mathcal{M}}$. In addition, we also create a feature-label correlation vector $\boldsymbol{\mathcal{L}}$ 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 instances, we use $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{L}}$ to construct an $m \leq m$ feature reordering matrix $\boldsymbol{\mathcal{O}}$ $\emptyset$ $\mathcal{V}^{m \bullet m}$, where $\mathcal{V}$ denotes the set of all real numbers.
- **Generating new presentation of instances:** After obtaining feature reordering matrix $\boldsymbol{\mathcal{O}}$, we reorder original feature values of instance $\boldsymbol{x_t}$ and convert instance $\boldsymbol{x_t}$ as a synthetic matrix format $\boldsymbol{\mathcal{F}(x_t)}$.

After converting each instance of the original dataset into a synthetic matrix, where all values are normalized to [0,1], EDLT applies deep learning methods to the matrix representation of each instance $\boldsymbol{x_t}$ to learn new features for classification.

III. EDLT: Enabling Deep Learning for Generic Classification Tasks

The main procedures of the proposed EDLT algorithm are briefly introduced in Algorithm 1. In this section, we

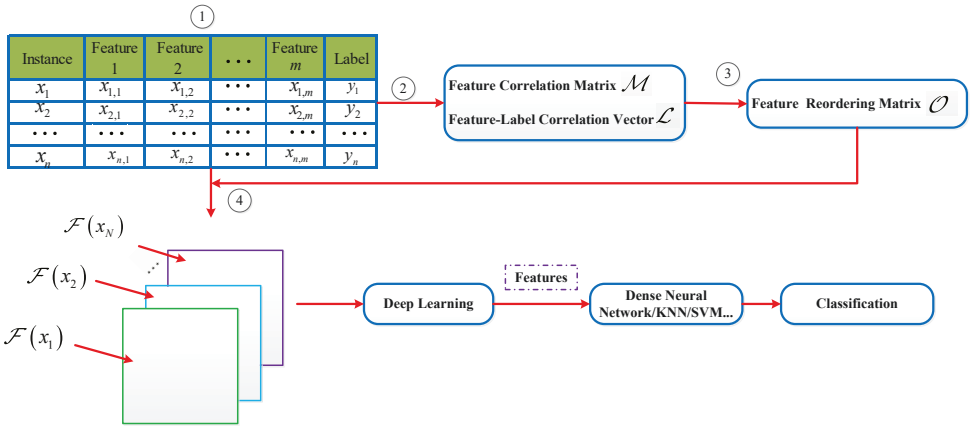

Fig. 4. A conceptual view of EDLT for classification. Given a generic dataset represented in instance-feature tabular format in ①, EDLT first calculates feature-feature correlation matrix and feature-label correlation vector ②. After that, EDLT constructs a feature reordering matrix ③, and converts each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

Algorithm 1 EDLT: Enabling Deep Learning for Generic Classification Tasks

Input:
$\mathcal{D}$: A generic data set;
$k \bullet k$: The size of convolution filter;
Output:
The synthetic matrix format of the original dataset $\mathcal{F}(\mathcal{D})$;
1: $\mathcal{M} \in$ Feature-feature correlation matrix using Eq. (1);
2: $\mathcal{L} \in$ Feature-label correlation vector;
3: Constructing feature reordering matrix $\mathcal{O}$:
1) $\mathcal{O} = \mathcal{F}$;
2) $\mathcal{V} \in$ Sorting features in a descending order according to their correlation to class label $\mathcal{L}$;
3) $\mathcal{V}_{1,l} = \cup_l, l = 1, 2, \ldots, m$: Determine the first row in $\mathcal{O}$;
4) **for each** $i \ / \ [1, 2, \times\times\times, (m \quad k+1)]$ **do**
a) **for each** $j \ / \ [1, 2, \times\times\times, (m \quad k+1)]$ **do**
i) $\boldsymbol{I}_i^j \in$ Find the set of known feature indexes in Δ_i^j;
ii) $\boldsymbol{\alpha}_i^j(\boldsymbol{\alpha}_i^0 = \mathcal{F}) \in$ Find the set of coordinates of unknown elements in Δ_i^j;
iii) $\mathcal{M}_i^j \in$ Set the sth row and column in $\mathcal{M}$ to 0 $\}s \ / \ [\mathcal{O}_{\boldsymbol{\alpha}_i^1}, \ldots, \mathcal{O}_{\boldsymbol{\alpha}_i^{j-1}}]$ & $s \ / \ \boldsymbol{I}_i^j\dagger$;
iv) $\boldsymbol{u} = [u_1, \times\times\times, u_m] \in$ Apply $\mathcal{M}_i^j$ and $\boldsymbol{I}_i^j$ to obtain the elements in Δ_i^j using Eq. (8).
v) $\boldsymbol{U} = \mathcal{F}$;
vi) **for each** $u_r \ / \ \boldsymbol{u}$ **do**
$\boldsymbol{U} = \boldsymbol{U} \{ \ r \ (u_r = 1 \& r \ / \ \boldsymbol{I}_i^j)$;
vii) **end**
viii) $\mathcal{O}_{\boldsymbol{\alpha}_i^j} = \boldsymbol{U}$;
b) **end**
5) **end**
4: **for all** $\boldsymbol{x_t} \ / \ \mathcal{D}$ **do**
5: $\mathcal{F}(\boldsymbol{x_t}) \in$ Algorithm 2 $(\boldsymbol{x_t}, \mathcal{O})$
6: **end for**
7: **return** $\mathcal{F}(\mathcal{D})$

first discuss technical details about feature-feature correlation matrix $\mathcal{M}$, feature-label correlation vector $\mathcal{L}$, and feature reordering matrix $\mathcal{O}$ construction. An example will be used to explain the conversion of each instance in the original dataset into synthetic matrix format, by utilizing the proposed EDLT method.

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 tth instance in the original dataset;
Output:
The synthetic image format $\mathcal{F}(\boldsymbol{x_t})$;
1: $i = 1, j = 1$;
2: **while** $i \not\in m$ **do**
3: **while** $j \not\in m$ **do**
4: $\mathcal{O}(x_t)_{i,j} = x_{t,\mathcal{O}_{i,j}}$
5: $i = i + 1$
6: **end while**
7: $j = j + 1$
8: **end while**

A. Feature-feature Correlation Matrix $\mathcal{M}$ and Feature-label Correlation Vector $\mathcal{L}$

To handle higher order feature correlation, EDLT first constructs a feature-feature correlation matrix $\mathcal{M} \ \emptyset \ \mathcal{V}^{m \bullet m}$ and a feature-label vector $\mathcal{L} \ \emptyset \ \mathcal{V}^{1 \bullet m}$. To capture pair-wise correlation between features, we utilize Pearson correlation coefficient, a common metric to measure correlation between random variables [18], to compute the matrix $\mathcal{M}$ and vector $\mathcal{L}$ as follows.

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

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 $\mathcal{L}$.

The absolute values in $\mathcal{M}$ and $\mathcal{L}$ vary 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 absolve values of $\mathcal{M}$ and $\mathcal{L}$ throughout the paper.

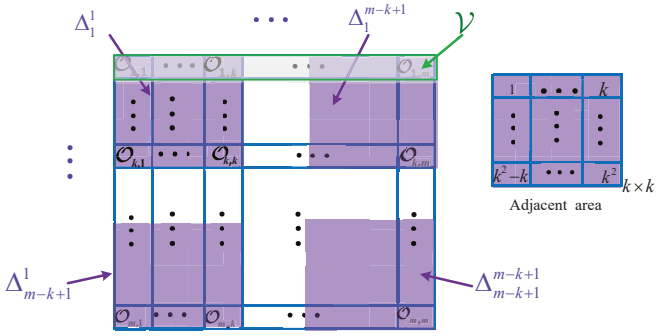

Fig. 5. A conceptual view of feature reordering matrix $\mathcal{O} \in \mathcal{R}^{m\times m}$. Features are reordered such that their correlation in each $k \times k$ adjacent area is maximized, and global correlation in $\mathcal{O}$ is also maximized. The size of the adjacent area is determined by the convolution filter size.

B. Feature Reordering Matrix $\mathcal{O}$ Construction

Given features correlation matrix $\mathcal{M}$ and feature-label correlation vector $\mathcal{L}$, feature reordering matrix $\mathcal{O}$ is created to bring correlated features close to each other for deep learning methods to utilize correlation for learning.

Fig. 5 shows the structure of the feature reordering matrix $\mathcal{O}$, where each element $\mathcal{R}_{i,j}$ in the matrix denotes the index of the original feature used to construct the synthetic matrix (using Algorithm 2). For each small adjacent area in $\mathcal{O}$, we expect features in the area to have a strong correlation. This correlation will eventually help build a synthetic matrix $\mathcal{F}(\boldsymbol{x_t})$ for each instance, where feature values in any small area of $\mathcal{F}(\boldsymbol{x_t})$ will have a strong correlation.

Formally, we define the area in synthetic image $\mathcal{F}(\boldsymbol{x_t})$ that will be multiplied by the entries of the convolution filter as a adjacent area. Based on Algorithm 2, the adjacent area in $\mathcal{F}(\boldsymbol{x_t})$ is that in $\mathcal{O}$, which is shown in Fig. 5 where Δ_i^j denote the adjacent area when operating the ith multiplication with convolution filter along the horizontal axis and the jth multiplication with convolution filter along vertical axis. Considering a $k \leq k$ convolution filter and setting the convolution stride to 1, the number of adjacent areas is $(m \quad k+1)^2$. Our aim is to maximize the sum of the features correlation in adjacent area, which can be formulated as

$$\arg\max_{\mathcal{O}} \sum_{i=1}^{m\ k+1} \sum_{j=1}^{m\ k+1} C(\Delta_i^j), \tag{3}$$

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

$$C(\Delta_i^j) = \sum_{s=j}^{j+k\ 1} \sum_{t=i}^{i+k\ 1} \sum_{r=j}^{j+k\ 1} \sum_{q=i}^{i+k\ 1} \mathcal{O}_{\mathcal{V}_{s,t},\mathcal{V}_{r,q}}. \tag{4}$$

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

$$\begin{aligned} &C(\Delta_i^j) = (\boldsymbol{u^{ij}})^{\mathrm{T}} \mathcal{M} \boldsymbol{u^{ij}}. \\ &\text{s.t.} \quad u_r^{ij} = d_r(r \ \emptyset \ \boldsymbol{I_i^j}), u_r^{ij} \ \emptyset \ \}0, 1, ..., k^2|\,. \end{aligned} \tag{5}$$

where $(\cdot)^{\mathrm{T}}$ stands for the transposition operation, $\boldsymbol{u^{ij}} = [\ u_1^{ij}, \times\times, u_m^{ij}]$ 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 r in set Δ_i^j.

Substituting Eq. (5) into Eq. (3), we have

$$\begin{aligned} &\arg\max_{[\boldsymbol{u^{11}}, \boldsymbol{u^{12}}, \cdots, \boldsymbol{u^{(m-k+1)(m-k+1)}}]} \sum_{i=1}^{m\ k+1} \sum_{j=1}^{m\ k+1} (\boldsymbol{u^{ij}})^{\mathrm{T}} \mathcal{M} \boldsymbol{u^{ij}} \\ &\text{s.t.} \quad u_r^{ij} = d_r(r \ \emptyset \ \boldsymbol{I_i^j}), u_r^{ij} \ \emptyset \ \}0, 1, ..., k^2|\,. \end{aligned} \tag{6}$$

Finding feature reordering matrix $\mathcal{O}$ satisfying Eq. (6) is equivalent to finding feature indexes in each Δ_i^j such that

$$\begin{aligned} &\arg\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}} \mathcal{M} \boldsymbol{u} \\ &\text{s.t.} \quad u_r = d_r(r \ \emptyset \ \boldsymbol{I_i^j}), u_r \ \emptyset \ \}0, 1, ..., k^2|\,. \end{aligned} \tag{7}$$

For simplify, we omit the superscript of $\boldsymbol{u}$ in Eq. (7). Apparently, the solutions of Eq. (7) can only be obtained through brute-force search, which is very expensive and hard to implement.

To reduce the computation complexity to solve Eq. (7), we use feature-label correlation vector $\mathcal{L}$ to determine the first row in $\mathcal{O}$ 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 $\mathcal{V} = [\{_1, \{_2, \times\times, \{_m]$ as shown in Fig. 5, *i.e.*, $\mathcal{R}_{1,j} = \{_j (1 \in j \in m)$. By doing so, only a part of elements in $\Delta_1^j (1 \in j \in (m \quad k+1))$ is required to be solved, such that the computation complexity is reduced. Then, 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, which is consistent with the convolution process. Furthermore, to ensure the fairness between features, we make each row of the matrix $\mathcal{O}$ contain indexes of all features by adding the constrain on $\sum u_r$ and $\mathcal{M}$. As a result, the original feature-selection problem in Eq. (7) is transformed into a 0/1 integer programming problem as follows.

$$\begin{aligned} &\arg\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}} \mathcal{M}_{\boldsymbol{i}}^{\boldsymbol{j}} \boldsymbol{u} \\ &\text{s.t.} \sum_{r, u_r/\boldsymbol{u}} u_r = \boldsymbol{I_i^j} + \boldsymbol{\alpha_i^j}, \quad u_r \ \emptyset \ \}0, 1|\,; \\ &\mathcal{R}_{1,l} = \{_l, l = 1, 2, ..., m; \\ &u_r = 1 \ (r \ \emptyset \ \boldsymbol{I_i^j}), \end{aligned} \tag{8}$$

where $\boldsymbol{\alpha_i^j}$ is the set of coordinates of unknown elements in Δ_i^j, $\times$ denotes the cardinality of a set, and $\mathcal{M}_{\boldsymbol{i}}^{\boldsymbol{j}}$ is the modified feature correlation matrix which is derived by setting the sth row and column in $\mathcal{M}$ to $\mathbf{0}$ ($s \ \emptyset \ [\mathcal{O}_{\boldsymbol{\alpha_i^1}}, \times\times, \mathcal{O}_{\boldsymbol{\alpha_i^{j-1}}}]$ & $s \not\emptyset \ \boldsymbol{I_i^j}$).

TABLE I
A TOY MACHINE LEARNING DATASET

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

In Eq. (8), the constraint on $\sum u_r = \boldsymbol{I_i^j} + \boldsymbol{\alpha_i^j}$ and $\boldsymbol{\mathcal{M}_i^j}$ jointly ensure that the elements in each row of $\boldsymbol{\mathcal{O}}$ contains the indexes of all features to ensure the fairness between all features, *i.e.*, the feature indexes in $\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^j}}$ are not only different with each other but also different from the feature index in $[\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^1}}, \times\times\times, \boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^{j-1}}}]$, $j = 1, ..., (m \quad k+1)$. Specifically, we divide the elements in $[\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^1}}, \times\times\times, \boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^{j-1}}}]$ into two subsets $\boldsymbol{E_1}$ and $\boldsymbol{E_2}$. $\boldsymbol{E_1}$ contains the same elements in $[\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^1}}, \times\times\times, \boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^{j-1}}}]$ and $\boldsymbol{I_i^j}$, $\boldsymbol{E_2}$ contains the remaining elements in $[\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^1}}, \times\times\times, \boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^{j-1}}}]$. $\boldsymbol{\mathcal{M}_i^j}$ is obtained by setting the sth row and column in $\boldsymbol{\mathcal{M}}$ to 0 ($s \not\emptyset$ $[\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^1}}, \times\times\times, \boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^{j-1}}}]$ & $s \not\emptyset$ $\boldsymbol{I_i^j}$, i.e., s $\emptyset$ E_2) to make the elements in $\boldsymbol{I_i^j}$ are belong to the solutions of Eq. (8), which can only ensure that the feature indexes in $\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^j}}$ are different from the feature indexes in $\boldsymbol{E_2}$. Another constrain, *i.e.*, $\sum u_r = \boldsymbol{I_i^j} + \boldsymbol{\alpha_i^j}$, ensures that the feature indexes in $\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_i^j}}$ are not only different with each other but also different with the feature indexes in $\boldsymbol{E_1}$.

Eq. (8) is a standard 0/1 optimization problem, which can solved by using SDP [19] to find approximate solutions. In our experiments, we use branch-and-bound algorithm (BMIBNB) [20], which is based on a simple spatial branch-and-bound strategy, to find solutions for Eq. (8). The solution $u_r = 1$ ($r \not\emptyset$ $\boldsymbol{I_i^j}$) denotes that index r is one of the $\boldsymbol{\alpha_i^j}$ unknown elements in Δ_i^j, which is shown in Algorithm 1.

The time complexity to solve Eq. (8) is $O(m^2 2^m)$. In comparison, solving original optimization problem defined in Eq. (7), using brute-force search, requires $O(m^2 2^m)$ time complexity. Therefore, EDLT (*i.e.* Eq. (8)) can dramatically reduce the time complexity.

C. ***Example:*** *Synthetic Matrix Generation*

We use an example to explain the conversion of a generic dataset into synthetic matrix format for deep learning.

1) The generic dataset: Table I lists a dataset with 6 instances, 5 features, and 2 class labels.

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

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

TABLE II
THE CONVERTED SYNTHETIC MATRIX FOR INSTANCE $\boldsymbol{x_1}$: $\boldsymbol{\mathcal{F}(x_1)}$

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

Assume that the size of the CNN convolution filter is $2 \leq 2$, there are 16 adjacent areas of $2 \leq 2$ in $\boldsymbol{\mathcal{O}}$. Then, EDLT derives each adjacent area in $\boldsymbol{\mathcal{O}}$ from top to bottom, left to right. In other words, EDLT derives feature indexes in Δ_1^1,Δ_1^2,Δ_1^3, Δ_1^4,Δ_2^1,$\times\times\times$ 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 $s \not\emptyset$ ($\boldsymbol{\alpha_i^0} = \mathcal{F}$&$s \not\emptyset$ ($\boldsymbol{I_1^1} = [4, 3]$), 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 (8), EDLT derives the solution $\boldsymbol{u} = [4, 3, 2, 5]$. Because $\boldsymbol{I_1^1}$ is $[4, 3]$, the feature reordering $\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_1^1}} = [2, 5]$, *i.e.*, $\mathcal{R}_{2,1} = 2$ and $\mathcal{R}_{2,2} = 5$.

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 $s \not\emptyset$ ($\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_1^1}} = [2, 5]$) & $s \not\emptyset$ ($\boldsymbol{I_1^1} = [3, 5]$), $\boldsymbol{\mathcal{M}_1^2}$ is obtained by setting the 2th row and column of $\boldsymbol{\mathcal{M}}$ to $\mathbf{0}$ as in Fig. 6. Substituting $\boldsymbol{I_1^1}$, $\boldsymbol{\mathcal{M}_1^2}$, and $\boldsymbol{\alpha_1^2}$ into (8), we derive the solution $\boldsymbol{u} = [3, 5, 4]$. Because $\boldsymbol{I_1^1}$ is $[3, 5]$, the feature reordering $\boldsymbol{\mathcal{O}}_{\boldsymbol{\alpha_1^2}} = [4]$, *i.e.*, $\mathcal{R}_{2,3} = 4$.

Using similar logics , EDLT can derive feature indexes in other 7 adjacent areas, and the final feature reordering matrix $\boldsymbol{\mathcal{O}}$ is shown in Fig. 6.

4) Synthetic matrix generation: Table II shows the synthetic matrix of instance $\boldsymbol{x_1}$, *i.e.*, $\boldsymbol{\mathcal{F}(x_1)}$, constructed using Algorithm 2.

IV. EXPERIMENTS

To validate whether EDLT can indeed enable deep learning for generic datasets, we use CNN as the deep learning module and implement a number of baseline using Tensorflow [21] configured with one GPU card. The feature reordering matrix $\boldsymbol{\mathcal{O}}$ 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 20 benchmark datasets from UCI data repository [22]. In order to interpret features learned from EDLT created synthetic matrix, we utilize natural images from CIFAR-10 dataset [23] for a case study. A brief description of the 20 benchmark datasets is summarized in Table III. All reported results are based on 10 times 5-fold cross validation with classification accuracy being used as the performance metrics.

A. Experimental Settings

In our experiments, we utilize CNN as the deep learning method. For fair comparisons, each CNN model contains two

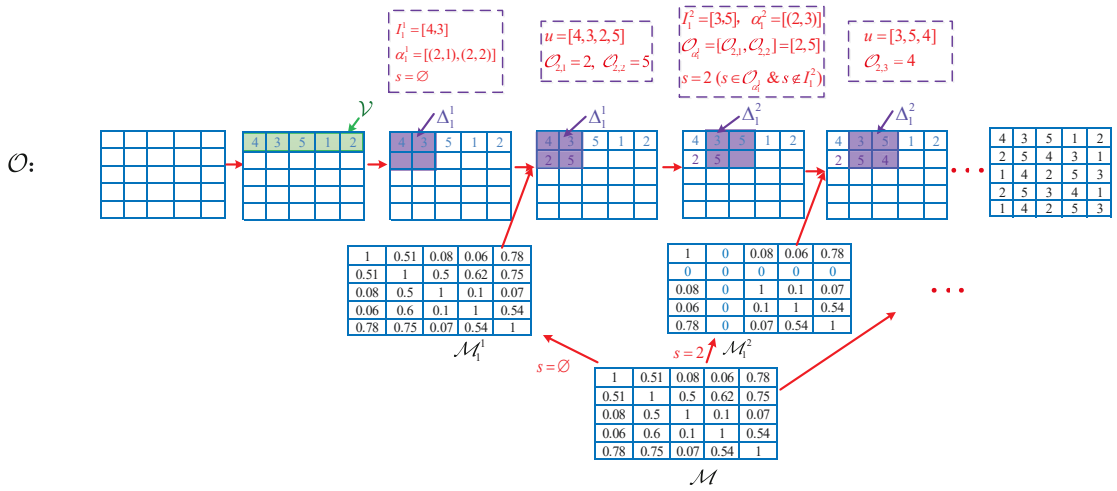

Fig. 6. **Example:** The process of constructing the feature reordering matrix.

TABLE III
A SUMMARY OF THE BENCHMARK DATASETS

ID	Dataset	Instances	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

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 [24], and use the Adam optimizer [25] with a learning rate of 0.001. Features extracted by the CNN model are used to train a single hidden layer dense neural network and a two hidden layer dense neural network to classify test data.

B. Baseline Methods

Because no method currently exists to enable deep learning for generic datasets, we implement two baseline approaches, Random Feature Reordering (RFR) and Label-Feature Correlation reordering (LFC), to compare the efficiency of the feature reordering module in EDLT. Similar to EDLT, both RFR and LFC enable deep learning for generic datasets by converting each instance into a synthetic matrix format. The difference between the two baselines and EDLT is the way of constructing feature reordering matrix $\mathcal{O}$.

RFR 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. The feature index order in each row of $\mathcal{O}$ is also obtained by ordering the feature indexes randomly. Because PFR uses random order to create a synthetic matrix for each instance, there is no local correlation compared to EDLT. If EDLT outperforms PFR, 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 features 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 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, where EDLT considers both global and local correlation maximization.

For comparison purposes, we also implement a number of machine learning methods (*i.e.*, 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 [26].

In the following sections, we first compare feature reordering methods *w.r.t.* different parameter settings, including different convolution filter sizes, the number of convolution filter, and number of hidden layers in the dense neural network. Then, we compare EDLT with generic machine learning algorithms on all benchmark datasets. Finally, we study a case to show effectiveness of EDLT in creating new instance representation for CNN to learn features for generic data.

TABLE IV
ALGORITHM PERFORMANCE COMPARISONS *w.r.t.* DIFFERENT CONVOLUTION FILTER SIZES (# OF CONVOLUTION FILTER IS 32 AND 4, AND # OF HIDDEN LAYER OF NN IS 1)

Dataset	RFR		LFC		EDLT	
	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$
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
Average	81.34	83.05	81.4	82.579	82.47	83.36

TABLE V
ALGORITHM PERFORMANCE COMPARISONS *w.r.t.* DIFFERENT CONVOLUTION FILTER SIZES (# OF CONVOLUTION FILTER IS 32 AND 16, AND # OF HIDDEN LAYER OF NN IS 1)

Dataset	RFR		LFC		EDLT	
	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$
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
Average	82.9	83.17	82.8	83.1	83.51	84.57

C. Different 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 $2 \leq 2$ and $3 \leq 3$, and the number of convolution filter is set to (32,4) and (32,16). For dense neural networks (NN), we consider two structures, *i.e.*, an NN with one hidden layer and an 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).

Tables IV-VII 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 IV and V, we report the accuracy performance of different feature reordering methods on 20 benchmark datasets using one hidden layer dense NN for final classification. The results of two hidden layer dense NN are reported in Tables VI and VII.

The results from Tables IV-VII 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, RFR does not consider any correlation in the synthetic matrix. The local correlation is the key for the CNN to learn meaning features.

In Table IV, compared to RFR and LFC, the average performance gains of EDLT are 1.13% and 1.07% for $2 \leq 2$ filter, and 0.31% and 0.78% for $3 \leq 3$ filter, respectively. In Table V, compared to RFR and LFC, the average performance

TABLE VI
ALGORITHM PERFORMANCE COMPARISONS *w.r.t.* DIFFERENT CONVOLUTION FILTER SIZES (# OF CONVOLUTION FILTER IS 32 AND 4, AND # OF HIDDEN LAYER OF NN IS 2)

Dataset	RFR		LFC		EDLT	
	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$
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
Average	81.8875	83.16	82.242	83.36	82.37475	84.39

TABLE VII
ALGORITHM PERFORMANCE COMPARISONS *w.r.t.* DIFFERENT CONVOLUTION FILTER SIZES (# OF CONVOLUTION FILTER IS 32 AND 4, AND # OF HIDDEN LAYER OF NN IS 2)

Dataset	RFR		LFC		EDLT	
	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$	$2 \bullet 2$	$3 \bullet 3$
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
Average	82.939	82.98	82.7405	83.29805	83.53875	83.94

gains of EDLT 0.61% and 0.71 % for $2 \leq 2$ filter, and 1.4% and 1.47% for $3 \leq 3$ filter. In Table VI, the average performance gains of EDLT are 0.49% and 0.13% for $2 \leq 2$ filter, and 1.23% and 1.03% for $3 \leq 3$ filter, and in Table VII, the the average performance gains of EDLT are 0.59% and 0.79 % for $2 \leq 2$ filter, and 0.96% and 0.64% for $3 \leq 3$ filter.

Furthermore, the best performance gain of EDLT is higher than that of RFR and LFC. To be specific, the best performance gain of EDLT , RFR and LFC are 84.57% in Table V, 83.17% in Table V, and 83.36% in Table VI, respectively.

The results in Tables IV-VII 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 $3 \leq 3$, the number of convolution filter to (32,16). Therefore, in the following subsections, we utilize this CNN structure settings in the experiments.

D. Detailed Comparisons for Machine Learning Methods

In Table VIII, we report detailed comparisons between EDLT and generic machine learning methods (*i.e.*, k-NN, SVM, DT, and NN) on 20 generic benchmark datasets, where NN-i means that the number of hidden layer in the dense NN is i. EDLT-i mean that the trained dense layer neural network (using features extracted from CNN) has i hidden layer(s).

In our experiments, we set the number of nearest neighbors to 5 for k-NN. For SVM, we use linear kennels and use one-vs-all to deal with multi-class tasks. We use CART (Classification

TABLE VIII
COMPARISONS BETWEEN EDLT AND DIFFERENT MACHINE LEARNING METHODS ON BENCHMARK DATASETS

Dataset	k-NN	SVM	DT	NN-1	NN-2	EDLT-1	EDLT-2
wall-following	85.1	78.8	99.7	67.94	83.42	91.75	91.13
vehicle	52.63	63.1	57.89	52.63	63.15	84.21	84.21
breast tissue	45.45	68	72.72	77.27	77.27	77.27	77.27
vowel	93.43	96.96	75.75	65.65	83.3	97.06	96.96
ecoli	85.29	80.8	82.35	89.7	88.23	85.29	83.08
wine quality-red	58.125	62.8	67.5	62.5	64.06	60.98	62
breast cancer wisconsin (Diagnostic)	95.6	97.36	91.22	97.36	97.36	98.24	98.24
wine	80.5	100	97.2	100	100	100	100
banknote authentication	100	97.8	99.27	94.54	100	100	100
vertebral column	77.41	74.19	77.41	74.19	79.03	81.84	82.25
yeast	54.2	58.24	49.83	58.24	60.94	60.04	59.81
seeds	85.7	85.7	95.23	95.23	95.23	90.47	90.47
climate model simulation crashes	96.29	95.37	88.8	94.4	98.1	97.22	97.22
glass identification	58.13	58.13	51.16	53.48	60.46	63.95	62.78
leaf	38.23	32.35	41.17	72.05	76.47	74.7	72.3
plrx	54.05	54.05	54.05	54.05	54.05	60.53	56.75
pima	77.9	82.46	78.57	79.87	80.51	78.83	78.69
iris	100	100	100	90	96.67	100	100
wireless indoor localization	97.5	96.75	96.25	96.25	97.25	98.5	97.5
sonar	83.3	83.3	64.28	88.09	85.71	90.47	88.09
Average	75.94	78.3	77.01	78.172	82.06	84.57	83.94

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 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. It is worth noting that for all traditional machine learning methods, including k-NN, SVM, DT, and NN, we use original features as their input. So we can justify whether EDLT can indeed help boost learning for generic classification tasks, by enabling deep learning.

The results from Table VIII show that among all methods, EDLT achieves the best performance gain. Compared to k-NN, SVM, DT, NN-1, and NN-2, the performance gains of EDLT-1 are 8.63%, 6.27%, 7.56%, 6.4% and 2.51%, respectively. Interestingly, the performance gain of EDLT-i is higher than that of NN-i, confirming that the features learned by utilizing the CNN are more effective than the original features in the generic dataset for classification. This further implies that converting each instance in the generic dataset into a synthetic matrix/image, like EDLT does, and further applying CNN to the converted data, and indeed enable deep learning for better classification accuracy for generic classification tasks.

E. 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 under 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 [27] as the original feature 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

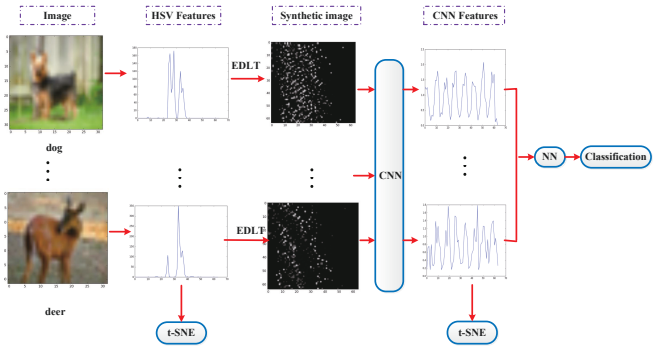

Fig. 7. Comparisons between original HSV features and CNN features learned from EDLT converted matrix/image ("Dog" *vs.* "Deer").

EDLT converts the original features to support deep learning. Therefore, we only use HSV features in our case study.

As shown in Fig. 7, we first extract the HSV features from each image. After that, the HSV features are used to generate a synthetic image by using EDLT. After that, 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 [28] to compare the difference between the HSV 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 $3 \leq 3$. The number of converlution filter are 16,32 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. 8 and 9, we further report the t SNE feature representation results between original HSV features and the

EDLT converted features. Fig. 8 shows that the CNN feature has better discrimination than the HSV feature for the two categories. Meanwhile, the results from Fig. 9 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.

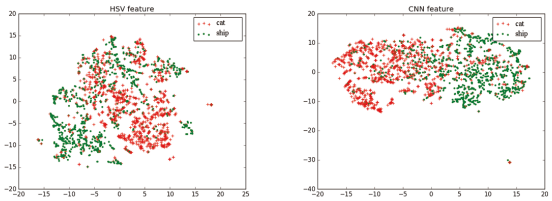

Fig. 8. Comparisons between original features (HSV) *vs.* EDLT converted CNN features("Cat" *vs.* "Ship")

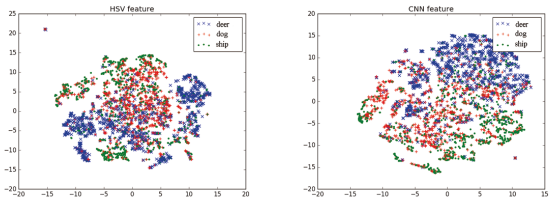

Fig. 9. Comparisons between original features (HSV) *vs.* EDLT converted CNN features("Deer", "Cat", and "Ship")

V. CONCLUSION

In this paper, we proposed to enable deep learning for generic data classification where data used for learning are not images/videos, but are already represented instance-feature tabular format. 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 propose 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 20 generic datasets confirm that enabling deep learning to generic datasets has clear performance gain, compared to classifiers learned from the original feature space.

VI. ACKNOWLEDGEMENT

This research is sponsored by the US National Science Foundation (NSF) through Grants IIS-1763452, CNS-1828181, and by the National Natural Science Foundation of China under Grant 61671345.

REFERENCES

[1] Y. LeCun, Y. Bengio, G. Hinton, "Deep learning, " *Nature 521*, pp. 436-444, 2015.
[2] J. Schmidhuber, "Deep learning in neural networks: An overview, " *Neural Networks*, vol.61, pp.85-117, 2015.
[3] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning, " in *NIPS*, 2014, pp. 3338-3346.
[4] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition, " in *ECCV*, 2014.
[5] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, " in *ICML*,2009, pp. 609-616.
[6] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, " *The Journal of Machine Learning Research*, pp. 3371-3408, 2010
[7] K. Kavukcuoglu, et al., "Learning convolutional feature hierarchies for visual recognition, " in *NIPS*, 2010, pp. 1090-1098.
[8] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders, " in *ICML*, 2008, pp. 1096-1103.
[9] Graves, A., Mohamed, A.-R. and Hinton, G., "Speech recognition with deep recurrent neural networks, " in *International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645-6649.
[10] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet classification with deep convolutional neural networks, " in *NIPS*,2012, pp. 1106-1114.
[11] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification, " in *CVPR*,2012.
[12] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks., " in *NIPS*,2012.
[13] 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 *CCBD*, 2016, pp. 87-92.
[14] X. Ding, Y. Zhang, T. Liu, J. Duan, "Deep Learning for Event-Driven Stock Prediction, " in *IJCAI*, 2015, pp. 2327-2333.
[15] Nasser M. Nasrabadi, "Pattern Recognition and Machine Learning, " *Journal of Electronic Imaging*, vol. 16, no. 4, 2007.
[16] V. Rokhlin, A. Szlam, and M. Tygert, "A randomized algorithm for principal component analysis, " *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 3, pp.1100-1124, 2009.
[17] D. Lunga, S. Prasad, M. M. Crawford, and O. Ersoy, "Manifold Learning-Based Feature Extraction for Classification of Hyperspectral Data: A Review of Advances in Manifold Learning, " *IEEE Signal Processing Magazine*, vol. 31, no.1, pp. 55-66, 2014.
[18] J. Hauke, T. Kossowski, "Comparison of values of Pearson's and Spearman's correlation coefficient on the same sets of data, " *Quaestiones Geographicae*, vol. 31, no.2, pp. 87-93, 2011.
[19] M. Goemans and D. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, " *ACM*, vol. 42, no.6, pp. 1115-1145, 1995.
[20] L. Sven, "Integrating SQP and branch-and-bound for mixed integer nonlinear programming, " *J. Computational Optimization & Applications*, vol. 18, no.3, pp. 295-309, 1998.
[21] M. Abadi, A. Agarwal, P. Barham, et al., "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org* vol. 1, 2015.
[22] D. Newman, S. Hettich, C. Blake, and C. Merz,, "UCI repository of machine learning databases, Irvine, CA: UCI Mach. Learn," 1998. [Online]. Available: http://www.ics.uci.edu/ mlearn/MLRepository.html
[23] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis* Dept. of Computer Science, Univ. of Toronto, 2009.
[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models, " in *ICML*, 2013.
[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, "Scikit-learn: Machine learning in Python, " *Journal of Machine Learning Research*, 12, pp. 2825-2830, 2011.
[27] R. Brunelli, O. Mich, "Histograms Analysis for Image Retrieval, " *Pattern Recognition*, vol. 34, no. 8, pp. 1625-1637, 2001.
[28] L. van der Maaten, and G.E. Hinton, "Visualizing data using t-SNE, " *J. Mach. Learn.Research 9*, pp. 2579-2605, 2008.