2018 17th IEEE International Conference on Machine Learning and Applications

Learning Convolutional Neural Networks From Ordered Features Of Generic Data

Eric Golinko, Thomas Sonderman, and Xingquan Zhu
Dept. of Computer & Electrical Eng. and Computer Science, Florida Atlantic University
Boca Raton, Florida 33431, USA
{egolinko, tsonderm, xzhu3}@fau.edu

***Abstract*—Convolutional neural networks (CNN) have become very popular for computer vision, text, and sequence tasks. CNNs have the advantage of being able to learn local patterns through convolution filters. However, generic datasets do not have meaningful local data correlations, because their features are assumed to be independent of each other. In this paper, we propose an approach to reorder features of a generic dataset to create feature correlations for CNN to learn feature representation, and use learned features as inputs to help improve traditional machine learning classifiers. Our experiments on benchmark data exhibit increased performance and illustrate the benefits of using CNNs for generic datasets.**

I. Introduction

Convolutional neural networks have become mainstream classification approaches in numerous fields, including computer vision and text mining [8] . In some tasks [4] CNNs have shown even better performance than human vision. Additional vision applications have included using CNNs for video classification tasks for large multiclass problems [7]. Additionally, in text mining, text embedding methods have shown substantial results for sentiment identification. A key characteristic of text problems have been to create a meaningful embedding of words [10] based on the vocabulary of the underlying data. Once a meaningful embedding is accomplished, a CNN architecture may be used to create features based upon the relative relationships that occur in the word proximity and context.

Though these methods have shown extremely promising results, there has been little study about applying convolutional methodology to learn features for generic non-image or non-text data. We seek to examine a novel way to reorder features such that a CNN representation can be constructed, and be extended to traditional machine learning algorithms such as Nearest Neighbor (NN), Random Forest (RF), or Support Vector Machines (SVM).

Generic datasets do not have have clear temporally or spatially correlated features. In fact, oftentimes it is assumed that variables are independent. For data that do not have order, CNN features do not build meaningful representations typically. Therefore, we wish to develop a meaningful ordering of features for generic data. By creating an ordering of the features, we can use CNNs to create a new representation of the data and use subsequent deep learning frameworks typically reserved for images or text embedding. In our research we illustrate a technique that can order features of a generic dataset, such that a meaningful CNN representation many be used. In order to create this ordering, we utilize our previous feature embedding approach [5] to aid in creating this ordering. This technique is similar to principal component analysis [1] and factor analysis [3] in that we are able to correlate original features of the data with the embedded features.

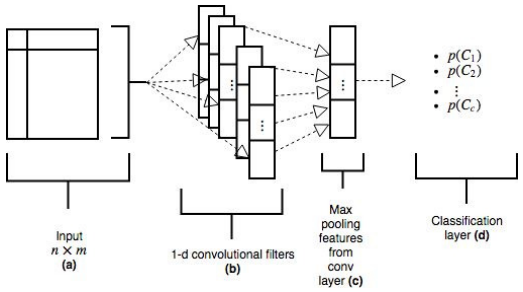

Fig. 1. A conceptual view of 1-d CNN learning. The training dataset contains n instances and m features. 1-d CNN treats each instance as an input of $m \times 1$ (m time steps of a single feature) or $1 \times m$ (one time step of m features).

II. Convolutional Neural Networks for 1-Dimensional Data

While CNNs are typically used for two dimensional data, such as images, recent research has extended CNNs to text and time series prediction. In these one dimensional data applications, data occur in a series that has a meaningful spatial or temporal relationship. In summary, CNNs for

978-1-5386-6805-4/18/$31.00 ©2018 IEEE
DOI 10.1109/ICMLA.2018.00145

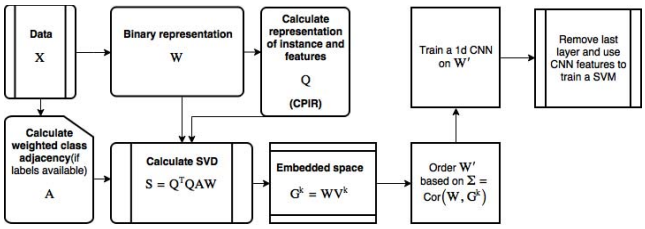

Fig. 2. Workflow of the proposed method which finds a meaningful ordering of original features to train a 1-d CNN and learn new features for classification.

sequential data have four main components: a) input layer, b) convolutional layer, c) pooling layer, and d) activation and classification. This setup may be extended by adding dense layers, additional convolutional layers, and adjusting parameters to each layer. We illustrate in Fig. 1 the architecture of the general model we used.

Similar to the CNN for images, the 1-d convolution takes a vector that is one dimension. This can be interpreted as one feature over multiple time steps or as a singe instance that has many features or channels. To draw comparison with the image CNN network, the input shape would have multiple dimensions, *e.g.* $28 \times 28 \times 3$, to represent images with red, green, and blue channels. For one dimensional data, we could have an input of $m \times 1$ which represent multiple steps of a sequence or times series. Alternatively, we can use an input of dimension $1 \times m$, indicating that an instance only has one step but has multiple features or channels. The distinction we make is the size of the kernel for the 1-d cases. We can apply filters of up to size m for the case where our input is $m \times 1$, however where we only have one time stamp, $1 \times m$ our kernel size is only 1. Therefore, we are able to learn the feature or channels in small increments.

III. Proposed Methodology

In order to use CNNs for one dimension data, we first must develop a meaningful ordering of generic features that do not naturally have a temporal ordering or a pre-defined spatial relationship. In the case of 1-d or temporal CNN, the convolution window moves along each channel or feature in an instance in short strides that are defined and initiated as a parameter in a network setting. For data that does not have order, CNN features do not naturally build meaningful representations. Therefore, we reorder features to create correlations such that we can utilize the deep learning representation using CNN architectures. Fig. 2 illustrates work-flow of our proposed methodology.

A. Feature Order From Embedding

In order to create a meaningful reordering of features, we must first represent generic data, which often contain numerical and categorical features, in a way for numerical analysis. Therefore we use our feature embedding approach [5] to create an embedding that will aid in feature ordering.

In general the process is as follows 1) one-hot encode a dataset X as W, 2) create an embedded space G^k from W such that k embedded features are created, 3) correlate the top embedded feature with the features of W, 4) let W' be the ordered features of W based on feature correlation with G^k.

The embedded space G^k is the eigenvector solution Eq. 1 to the represented space of W. The derived equation follows from three distinct matrices; 1) Q is the class partitioned row and feature representation developed from W, 2) A is a weighted adjacency matrix where an edge is represented by instances having the same class label, and 3) W the one-hot encoded data of X. From the work generated in [5] this approach uses class labels along with distance relationships to form an embedded space. Specifically, the methodology aims to work with sparse categorical features that are generated by one-hot encoding.

$$S = Q^T QAW \tag{1}$$

More formally, let w_i be the one-hot encoded features of W. Then with out loss of generality we calculate the correlations as:

$$\Sigma^i = Cor(w_i, G^k) \text{ for } i \in 1 \dots m \tag{2}$$

the index of one-hot encoded features of W.

Following calculating all of the correlations, we then rank order all of Σ^i as:

$$\Sigma^i_{r_p}, \text{ where } \Sigma^i_{r_1} \geq \cdots \geq \Sigma^j_{r_m}, \ i \neq j \tag{3}$$

This indicates the feature w_i is most correlated with the most import embedded feature as signified by r_1, and w_j is the least correlated noted by its correlation rank of r_m.

B. 1-d CNN and Classification

To create CNN features, we use a simple architecture based on examples from the Keras package [2]. We make a slight change of how the array of data is being shaped, this is done by considering each instance to have 1 step in time and m features. Typically, sequential data will be considered to have one feature and multiple time steps, therefore our implementation can be seen as a transposition of typical sequential implementations. As input the first layer takes

a input of 1 time step with m features. Arbitrarily, the number of output layers is chosen as a parameter f. In our experiments we set $f = 10, 50, 100, 200, 300, 400, 500$. Specifically, since we treat each instance as 1 time step, the size of the width of the convolutional filter will be 1. After the convolutional layer, we use a max pooling layer followed by a softmax layer. The network is trained, then the softmax layer is popped off, and our subsequent f weights create the feature representation from W' as $\mathcal{W}$. From $\mathcal{W}$ we are able to now apply and classify the data using SVM, random forest, or 1-nearest neighbor. A visualization of our architecture is shown in Fig. 2.

IV. Experiments

In this section we report results over benchmark data utilizing baseline machine learning algorithms, including support vector machine (SVM), 1-nearest neighbor (1-NN), and random forest (RF), as classifiers. Multiclass accuracy was used on the test set to measure performance.

A. Benchmark Methods

In the experiments, we implement three benchmark methods for comparisons. More specifically, for each generic dataset, we use its original features to train classifiers for classification, and denote this method by "OrgFeature" (*i.e.* Original Features). Meanwhile, for each dataset, we use the original order of the features and feed the data to the 1-d CNN to learn new features (in other words, ordering of the features is taken into the CNN architecture exactly as the data read-in from the file), and denote this method by "Org-CNN". To evaluate the effectiveness of the feature ordering, we use the proposed approach to order features (we used the first eigenvector embedding vector in our experiments) and feed the ordered data to the 1-d CNN to learn new features, and denote this method by "Ord-CNN". By comparing the performance of the three benchmark methods, we can conclude (1) whether CNN can indeed help improve classification accuracy for generic datasets, and (2) whether the proposed feature ordering method can indeed help create local correlation to improve CNN learning.

B. Experimental Settings

Benchmark data used in the experiment include 28 datasets collected from the UCI data repository [9], and several other sources [6]. The datasets cover data from different domains, including text, biomedical, pattern recognition, and many others. The datasets include tasks from binary labels to up-to 50 class labels, and features vary from 4 to 2,633 dimensions. We intentionally select benchmark data with diverse characteristics in order to validate the algorithm's genuine performance of both classifiers and feature orderings.

For each dataset, we first create its one-hot encode, and repeat the following experiment ten times for each feature set of $f = 10, 50, 100, 200, 300, 400, 500$. 1) Create a random 80/20 split of the data, 2) order features according to "OrgFeature", "Org-CNN", and "Ord-CNN", 3) for "OrgFeature" train each classifier SVM, 1-NN, and RF on the same train split, 4) for "Org-CNN", and "Ord-CNN" utilize the CNN architecture to create feature order and train with each respective classifier, 5) For each model, evaluate the model on the test data, and 6) record the multiclass accuracy.

TABLE I
Overall Accuracy

Method	SVM	1-NN	RF
OrgFeature	0.76	0.72	0.81
Org-CNN	0.43	0.56	0.46
Ord-CNN	0.85	0.81	0.83

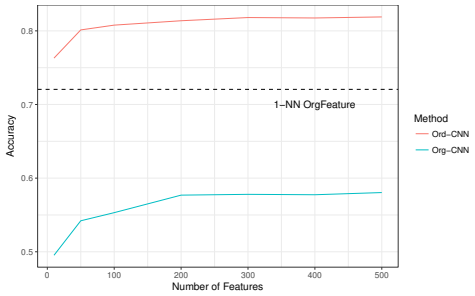

Fig. 3. Average accuracy for 1-NN over number of features and datasets.

C. Results

Table I reports the overall accuracy for all the learner and feature order combinations. From the table we can see that the ordering "Ord-CNN" is better than the reference "OrgFeature" as well as "Org-CNN", in addition, "Ord-CNN", performs better overall for all the datasets among the three learners.

Illustrating the performance over all the datasets we can visualize the performance of 1-NN among the "Org-CNN" and "OrgFeature", and the "Ord-CNN" ordered features, represented in Fig. 3. In this experiment as well for the other two learners, this same trend was shown. The embedding "Ord-CNN" features consistently perform "Org-CNN" and "OrgFeature". It is clear from Fig. 3 that when the number of features reaches 100, the average accuracy begins to level out.

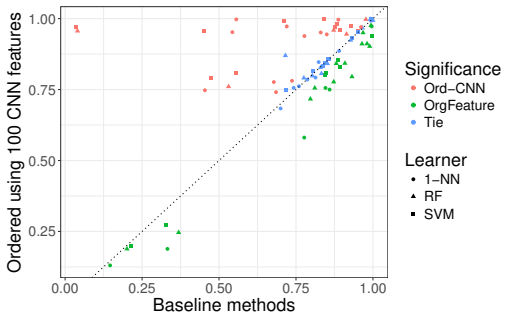

Fig. 4. Head to head t-test comparisons between "Ord-CNN" and "OrgFeature" on all datasets. Each dot denotes a classifier. Dots are color and shaped-coded. Shape denotes different learners and color denotes t-test significance.

TABLE II
OVERALL ACCURACY OF METHOD AND NUMBER OF FEATURE COMBINATIONS FOR RANDOM FOREST (BASELINE 0.811)

Method\Feature	10	50	100	200	400
Org-CNN	0.45	0.46	0.46	0.46	0.46
Ord-CNN	0.79	0.82	0.83	0.84	0.84

In Fig. 4, we compare accuracy of classifiers trained from "Ord-CNN" (using 100 features) $vs.$ "OrgFeature" across all datasets. Each dot in Fig. 4 denotes the accuracy of a classifier on a dataset, and a value above $y = x$ line indicates that "Ord-CNN" outperforms "OrgFeature".

As an example of performance over the feature sets, we highlight the Random Forest "OrgFeature" versus the average accuracies over all datasets and the respective learner and feature ordering. The results in Table II show that the accuracy of "Ord-CNN" orderings consistently outperform "Org-CNN" orderings. Meanwhile, the accuracy of "Ord-CNN" orderings is also better than "OrgFeature" baselines, except when the number of features was ten, the accuracy was slightly less than "OrgFeature" (which is 0.811). Additionally, the t-tests with respect to each learner and each dataset in Table III shows that for both 1-NN and SVM the performance increased significantly, however for Random Forest, the t-test showed that the "OrgFeature" performed better on a dataset by dataset level.

TABLE III
t-TEST FOR LEARNERS AND DATASETS. ORD-CNN METHOD WITH 100 CNN FEATURES VERSUS BASELINE(ORGFEATURE)

	1-NN	RF	SVM
Ord-CNN	13	5	11
OrgFeature	6	13	7
Tie	9	10	10

Overall, our experiments showed that feature orders play an important role for CNNs to learn new features. From this ordering, the "Ord-CNN" embedding results showed performance improvements over "OrgFeature" baseline methods. The only exception being that on a dataset by dataset t-test comparison for Random Forest, the baseline "OrgFeature" performed better. However, we also observed a general trend for all data, method, and learner instances that the ordered methods predominately always had accuracy in greater magnitude overall the experiments and number of features. Therefore, without significant parameter tuning, the feature order methodology has shown substantially improved results in implementation.

V. CONCLUSION

In this study, we explore creating CNN features from generic datasets. To do this we first transform the data to one-hot encoded features, from this we are able to generate an embedded space, such that we can rank order the correlations of the one-hot features to the most important embedded features. The correlations then form a natural order of these features which we use in creating a representation using a 1-d CNN architecture and traditional machine learning classifiers. The results show that ordering of features for CNN learning resulted in increased performance.

ACKNOWLEDGMENTS

This work is sponsored by the US National Science Foundation (NSF) through grant CNS-1828181.

REFERENCES

[1] Rasmus Bro and Age K Smilde. Principal component analysis. *Analytical Methods*, 6(9):2812–2831, 2014.
[2] Francois Chollet. *Deep learning with Python*. Manning Publications, 2018.
[3] Anna B Costello and Jason W Osborne. Best practices in exploratory factor analysis: Four recommendations for getting the most from your analysis. *Practical assessment, research & evaluation*, 10(7):1–9, 2005.
[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
[5] Eric Golinko and Xingquan Zhu. Generalized feature embedding for supervised, unsupervised, and online learning tasks. *Information Systems Frontiers*, pages 1–18, 2018.
[6] Kaggle. https://www.kaggle.com, 2017.
[7] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.
[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
[9] M. Lichman. UCI machine learning repository, 2013.
[10] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.