Robust Deep Graph Based Learning for Binary Classification

[1,3]Minxiang Ye, [1]Vladimir Stankovic, *Senior Member, IEEE,* [1]Lina Stankovic, *Senior Member, IEEE,* [2]Gene Cheung, *Senior Member, IEEE*

***Abstract*—Convolutional neural network (CNN)-based feature learning has become the state-of-the-art for many applications since, given sufficient training data, CNN can significantly outperform traditional methods for various classification tasks. However, feature learning is more challenging if training labels are noisy as CNN tends to overfit to the noisy training labels, resulting in sub-par classification performance. In this paper, we propose a robust binary classifier by learning CNN-based deep metric functions, to construct a graph, used to clean the noisy labels via graph Laplacian regularization (GLR). The denoised labels are then used in two proposed loss correction functions to regularize the deep metric functions. As a result, the node-to-node correlations in the graph are better reflected, leading to improved predictive performance. The experiments on three datasets, varying in number and type of features and under different levels of noise, demonstrate that given a noisy training dataset for the semi-supervised classification task, our proposed networks outperform several state-of-the-art classifiers, including label-noise robust support vector machine, CNNs with three different robust loss functions, model-based GLR, and dynamic graph CNN classifiers.**

***Index Terms*—deep learning, graph Laplacian regularization, binary classification, semi-supervised learning**

I. Introduction

Supervised and semi-supervised deep learning techniques have shown excellent performance for feature extraction and classification tasks [2], but are particularly sensitive to the quality of the training dataset, since they tend to overfit the models when learning from incorrect labels [3], [4], [5].

Conventional approaches to overcome model overfitting, based on various regularization techniques, e.g., l_1- or l_2-norm penalty on weights [6], dropout [7], batch normalization [8], skip-connections [9], [10] etc., are not effective in mitigating the effects of incorrect labels. This has given rise to different approaches to learning using noisy training labels. These approaches can be grouped into methods based on: (a) inserting additional "trusted" labels, e.g., [11], [12] and (b) loss function correction, e.g., [13], [14], [15]. In [11], a probabilistic model is integrated into the deep neural network (DNN) to correct noisy labels. Similarly, in [12], a loss correction technique is introduced to mitigate the unreliability of noisy training labels. However, these methods require clean data to prevent the models from drifting away. As a representative of the second type of methods, in [13], a robust loss function that is less sensitive to label outliers is introduced. A combination of training labels and predicted labels is used in [14] to avoid directly modeling the noisy training labels, but requires pre-training to achieve good results. [15] adds another softmax layer to further augment the correction model via a noise transition matrix, which is hard to estimate in practice.

An alternative approach is to restore corrupted training labels by representing them as smooth signals on graphs and applying a graph signal smoothness prior [16], [17], [18], [19]. In [20], an image denoising scheme is proposed using graph Laplacian regularization (GLR). Building on [20], [21] integrates GLR into a DNN to perform semi-supervised classification of nodes in a citation network, considering, during label propagation, the local consistency of nodes with similar features. However, these hybrid methods [21], [20], are only performed and evaluated for a fixed graph (i.e., a 2D grid of image pixels). Without prior knowledge of the graph structure, in our conference paper [1], we propose a regularized triplet loss correction function to mitigate the effects of insufficient training samples via a sparse K-nearest neighbor (KNN) graph construction and GLR. [1] only tackled the problem of insufficient training samples and assumed clean training labels.

This paper addresses the additional problem of noisy training labels, which requires important changes in the architecture of [1]. Building on our initial design [1], we further extend the previous studies on binary classification in the presence of noisy labels with DNN-based classifier learning using GLR without prior knowledge of the underlying graph. After the noisy labels are restored via GLR, CNN is used again to improve the deep feature learning step, leading to an improved graph construction that is finally used for the GLR-based classification [17], [18], [19]. Similar to [1], we adopt deep feature learning to construct a similarity graph in Euclidean metric space via DNNs where the edge weights are assigned using a Gaussian kernel function. Instead of heuristically setting the scaling factor of the Gaussian kernel function, we propose a new optimization method to assign the edge weights to better distinguish the edges connecting the nodes with the same label from those connecting the nodes with opposite label. Similarly to [1] that uses GLR to restore *missing labels*, here we perform GLR to restore *noisy labels* and use an attention mechanism to regularize the DNNs via a regularized triplet loss function in case of sufficient training samples. To further facilitate the robust classifier learning against "noisy labels",

The authors [1] are with the Department of Electronic & Electrical Engineering, University of Strathclyde, Glasgow, G1 1XW, UK. The author [2] is with the Department of Electrical Engineering & Computer Science, York University, Toronto, M3J 1P3, Canada. The author [3] is with the Research Center for Intelligent Robot, Zhejiang Lab, Hangzhou, China.

The initial results were presented in [1].

the proposed scheme alternates between the feature learning step and graph update to improve graph data representation for semi-supervised classification. Specifically, we introduce edge convolutions on the aggregations of restored graph signals over the neighbors and the corresponding feature maps to better learn a smooth graph for classifier learning.

In summary, the main contributions of this paper are:

1) A graph-based regularized loss function that incorporates attention mechanism to regularize the proposed network used to avoid overfitting the classifier;
2) A graph-based semi-supervised classifier that performs both online denoising of training labels and classification to mitigate the effects of noisy training labels and improve the reliability of classifier learning;
3) A graph update procedure to better reflect the node-to-node correlation based on convolution of updated edges, to assign a degree of freedom for graph connectivity learning, that is robust to noisy labels.
4) A new graph learning approach by automatically setting the scaling factor to quantify node-to-node correlation by maximizing the margin between the edge weights assigned to edges connecting nodes of the same label and those connecting nodes with the opposite label.
5) A complete semi-supervised binary classification scheme, including implementation details and ablation study to examine the importance of each component of the overall architecture.
6) Extensive evaluation of our proposed architecture against several classic and state-of-the-art methods, such as support vector machines (SVM) [22], convolutional neural network (CNN), dynamic graph CNN [23], deep metric-based KNN classifier [24], label noise robust SVM [25], GLR-based approach [18], CNN with robust loss corrections [13], [14], [5], designed specifically for the "noisy label" problem for three datasets and a wide range of noise.

The rest of the paper is structured as follows. First, an overview of related work is provided in Section II. In Section III, we introduce the notation, formulate the "noisy label" classifier learning problem, and describe the concept of the proposed solution. In Section IV, we describe an implementation of the proposed network that is used in the experimental section. Finally, in Section V we evaluate and discuss the performance of the proposed classifier against state-of-the-art methods and discuss findings of the ablation study for three different datasets.

List of Symbols

$\mathbf{X}$	A set of observations
r	GLR iteration number
$\mathbf{Y}^r$	Labels corresponding to $\mathbf{X}$ at r-th GLR iteration
$\dot{\mathbf{X}}, \dot{\mathbf{Y}}^r$	A subset of $\mathbf{Y}^r$ corresponding to the training samples in $\dot{\mathbf{X}}$
$\mathbf{E} = \{e_{i,j}\}$	Binary matrix that represents the edge connectivity with each entry $e_{i,j}$ corresponding the edge connecting node i to node j
$\mathbf{W} = \{w_{i,j}\}$	A weight matrix with each entry $w_{i,j}$ assigned to the corresponding edge $e_{i,j}$
$\mathcal{G} = (\mathbf{\Psi}, \mathbf{E}, \mathbf{W})$	A undirected graph that comprises a set of nodes $\mathbf{\Psi}$, edge matrix $\mathbf{E}$ and corresponding weights $\mathbf{W}$
$\mathbf{L}$	Combinatorial graph Laplacian matrix
$\mathbf{A}$	Adjacency matrix
$\mathbf{D}$	Degree matrix
d_{max}	Maximum degree of node in $\mathcal{G}$
$\mathcal{D}(\cdot), \mathcal{H}_{\mathbf{U}}(\cdot)$	Deep feature maps
$\mathcal{V}^r(\cdot), \mathcal{C}^r(\cdot)$	Deep feature maps associated with r-th GLR iteration
$\mathcal{Z}_D(\cdot), \mathcal{Z}_{\mathcal{H}_{\mathbf{U}}}(\cdot)$	Shallow feature maps in $\mathcal{D}(\cdot)$ and $\mathcal{H}_{\mathbf{U}}(\cdot)$
$f^r(x)$	Observations associated with r-th GLR iteration
$g(\mathbf{x})$	Observations associated with graph update
$\mathbf{x}_a$	A random node a selected from $\mathbf{X}$
$\mathbf{x}_p$	A random node p selected from $\mathbf{X}$, with same label as $\mathbf{x}_a$
$\mathbf{x}_n$	A random node n selected from $\mathbf{X}$, with opposite label as $\mathbf{x}_a$
$\mathcal{P}$	A set of edges connecting nodes with same labels
$\mathcal{Q}$	A set of edges connecting nodes with opposite labels
$\alpha_{\mathbf{E}}, \alpha_{\mathbf{W}}$	Minimum margin between deep metric based distances of $\mathcal{P}$ and $\mathcal{Q}$ edges for $Loss_{\mathbf{E}}$ and $Loss_{\mathbf{W}}$
γ^r	Maximum number of nodes connected to each node in graph $\mathcal{G}^r$
$\mathbf{y}_i^1$	Encoded vector corresponding to the label y_i for node i
$\mathbf{y}_{\mathcal{U}}^1$	Encoded matrix corresponding to the labels for neighboring nodes
Θ	Activation function that estimates how much attention is paid on each edge
$\mathit{\Pi} = \{\pi_{i,j}\}$	Attention matrix with each entry $\pi_{i,j}$ corresponds to edge loss
$\pi_{a,p}, \pi_{a,n}$	Attentions on $\mathcal{P}$ and $\mathcal{Q}$ edges
Φ	Edge attention activation
κ	Conditional number
μ^r	Smoothness prior factor
ε^r	Thresholds in Θ and Φ

II. Related Work

In this section, we provide an overview of the related work on robust graph-based classifier learning and DNN-based classifier learning, in the presence of "noisy labels". Then we discuss graph-based methods integrated with DNN that do not consider noisy data. Finally, the weaknesses of the state-of-the-art classifiers are discussed to motivate the present work.

A. Robust Graph-based Learning

A label propagation method is proposed in [26] to evenly spread, throughout the graph, label distributions from selected labeled nodes, which are usually noisy and with heuristic information. A KNN-sparse graph-based semi-supervised

learning approach is proposed in [27] to remove most of the semantically-unrelated edges and adopt a refinement strategy to handle noisy labels.

To achieve robust binary classification, in [18], negative edge weights are introduced into the graph to separate the nodes in two different clusters. To perform classification via generalized GLR, a perturbation matrix Δ is introduced to preserve the eigen-structure of the original Laplacian $\mathbf{L}$, such that $\mathbf{L}+\Delta$ is positive semi-definite via the Haynsworth inertia additivity formula. Results demonstrate the applicability of negative edge weights for graph-based classifier learning for small amount of data without learning feature representation. We evaluate this approach in this paper when sufficient but noisy training labels are provided.

B. Robust DNN-based Classifier Learning

Many studies investigate methods to accommodate a wide range of label noise levels and types, which often focus on data augmentation, network design-based regularization and loss correction. Data augmentation techniques have been successfully used in [28], [29], [30] to automatically annotate unlabeled samples and use these samples for retraining. In [31], training examples are assigned weights by a proposed meta-learning algorithm to minimize the loss on a clean unbiased validation set based on gradient direction.

The effectiveness of dropout regularization for cleaning noisy labels is shown in [32]. For image classification, [33] indicate that increasing the batch size and downscaling the learning rate is a practical approach to mitigate the effects of label noise for a DNN-based classifier, given a sufficiently large training set.

The loss correction approach of [13] proposes a boosting algorithm 'SavageBoost' that is less sensitive to outliers and converges faster than conventional methods, such as Ada, Real, or LogitBoost. A noise-aware model is formulated in [34] to handle label omission and registration errors for improving labeling of aerial images. A dimensionality-driven learning strategy is discussed in [5] to avoid overfitting by identifying the transition from an early learning stage of dimensionality compression to an overfitting learning stage when the local intrinsic dimensionality steadily increases. Unlike the above loss correction studies to handle noisy and incomplete labeling, [14] use a combination of training labels and the prediction from the current model to update the training targets and perform weakly-supervised learning. Similarly, [35] integrate the Expectation-Maximization (EM) algorithm into CNN to detect and correct noisy labels, but require a properly pre-trained model. [36] propose an iterative learning framework to facilitate 'robustness to label noise' classifier learning by jointly performing iterative label detection, discriminative feature learning and re-weighting.

C. Graph-based classifier learning with DNN

Recent years have seen integration of graph-based learning with deep learning. Given a fixed graph structure, [37], [38], [39] design CNNs for feature learning by feeding a polynomial of the graph Laplacian. [40] adopt edge convolution to learn combinational spatial features from neighboring nodes given a fixed skeleton graph. Based on the ideas of edge convolution, [23] propose a deeper CNN model to learn the underlying KNN graph structure of point cloud data by iteratively updating the graph. The results demonstrate the capability of edge convolution for feature generalisation on point-cloud data. [20] propose a deep image denoising framework that couples encapsulation of the fully-differentiable GLR layer and learning 8-connected pixel adjacency graph structures via CNNs. The results indicate that given a small dataset, the method of [20] outperforms state-of-the-art approaches.

The problem of insufficient data or incorrect labels has not been investigated in the above graph-based hybrid methods. For incomplete or imprecise categories of labels (observations) in the training samples, [41] combine CNN and GLR using the sum of the cross-entropy loss and the GLR term for multi-label image annotation, where CNN is used to construct the fully-connected similarity-based graph.

Unlike [41], in our conference paper [1], we integrate GLR into CNN with a graph-based loss correction function to tackle the problem of insufficient training samples through semi-supervised graph learning.

D. Novelty with respect to reviewed literature

In this paper, we further extend our conference contribution [1] to a more generalized end-to-end CNN-based approach given noisy binary classifier signal, to alternate between GLR (similar to [20]) as a classifier signal restoration operator and deep feature metric learning to update the underlying graph. Compared to the previous graph-based classifiers [26], [16], [37], [42], [43], [38], [18], [21], [19], by adopting edge convolution, alternately updating graph and performing GLR, we learn a deeper feature representation, and assign the degree of freedom for learning the underlying data structure. Given noisy training labels, in contrast to the classical robust DNN-based classifiers [13], [14], [35], [36], [31], [5], we bring together the regularization benefits of GLR and the benefits of the proposed loss functions to perform more robust deep metric learning. We further adopt a rank-sampling strategy to find those training samples with high predictive performance that benefit inference.

III. Robust Deep Graph Based Classifier Learning

In this section, we first introduce notation and formulate the robust classifier learning problem following the related work [16], [18], [19], [20], [24]. Then, we describe the main concept behind the proposed Dynamic Graph Laplacian Regularization (DynGLR) neural network that learns robust deep feature map to effectively perform GLR when parts of the labeled data available to train the model are noisy.

A. Problem Formulation and Notation

Given observations $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$, where $\mathbf{x}_i \in \mathcal{R}^n$, $i = 1, \ldots, N$, the task of a binary classifier is to learn an approximate mapping function that maps each observation $\mathbf{x} \in \mathbf{X}$ into a corresponding binary discrete variable

$y \in \mathbf{Y} = \{y_1, \ldots, y_N\}$, called classification label, where $y_i \in \{-1, +1\}$, $i = 1, \ldots, N$.

Let $\dot{\mathbf{Y}}^0 = \{-1, 1\}^M = \{y_1, \ldots, y_M\} \subset \mathbf{Y}$, $0 < M < N$, be a set of known (possibly noisy) labels that correspond to instances $\dot{\mathbf{X}} = \{\mathbf{x_1}, \ldots, \mathbf{x_M}\} \subset \mathbf{X}$ used for training. Let $\mathbf{Y}^0 = \{\dot{\mathbf{Y}}^0, \mathbf{0}^{N-M}\}$, where we set to zero all $N-M$ unknown labels (to be estimated during testing).

Given $\mathbf{X}$, the problem addressed in this paper, is to learn the robust mapping function to assign a classification label to each observation $\mathbf{x} \in \mathbf{X}$ when some classification labels $y \in \dot{\mathbf{Y}}^0$, used for training the model, are incorrect.

Let $\mathcal{G} = (\mathbf{\Psi}, \mathbf{E}, \mathbf{W})$ be an undirected graph, where $\mathbf{\Psi} = \{\psi_1, \ldots, \psi_N\}$ is a set of nodes, each corresponding to one instance in $\mathbf{X}$, $\mathbf{E} = \{e_{i,j}\}, i, j \in \{1, \ldots, N\}$, is a matrix representing the edge connectivity of $\mathcal{G}$; that is, $e_{i,j} = 1$ if there is an edge connecting vertices i and j and $e_{i,j} = 0$ otherwise; and each entry $w_{i,j}$ in the weight matrix $\mathbf{W} = \{w_{i,j}\}, i, j \in \{1, \ldots, N\}$ corresponds to the weight associated with edge $e_{i,j}$. Then, $\mathbf{Y}^0$ can be seen as a graph signal that indexes the graph $\mathcal{G}$, that is, each node $i \in \Psi$ will be assigned to a label $y_i \in \mathbf{Y}^0$. The combinatorial graph Laplacian matrix is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{A}$ is a symmetric $N \times N$ adjacency matrix with each entry $a_{i,j} = max(w_{i,j} \cdot e_{i,j}, w_{j,i} \cdot e_{j,i})$, and $\mathbf{D}$ is a degree matrix with entries $d_{i,i} = \sum_{j=1}^{N} a_{i,j}$, and $d_{i,j} = 0$ for $i \neq j$.

Similarly to [24], we define *triplets* as observations $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$, $\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n \in \mathbf{X}$ corresponding to vertices $\psi_a, \psi_p, \psi_n \in \mathbf{\Psi}$, respectively, such that $y_a = y_p \neq y_n$, and $y_a, y_p, y_n \in \dot{\mathbf{Y}}$. Let $\mathcal{P}$ be a set of all edges $e_{a,p}$, such that $y_a = y_p$, and $\mathcal{Q}$ a set of all edges $e_{a,n}$, for which $y_a \neq y_n$, that is, $\mathcal{P}$ and $\mathcal{Q}$ are sets of all edges that connect nodes with the same and opposite labels, respectively.

Motivated by CNNs ability to extract discriminative features and GLRs to 'clean' unreliable labels, we formulate graph-based classifier learning as a two-stage learning process: (1) *graph learning* - extract deep feature maps, i.e., find a deep metric function that returns the most discriminative feature maps, and then generate an initial graph by learning the underlying $\mathbf{E}$ to maximize/minimize similarity between any two nodes in $\mathcal{G}$ that are indexed by the same/opposite labels. (2) *classifier learning* - alternate between refining the graph (via deep feature metric learning) and performing GLR to restore the corrupted classifier signal. In the following, we describe these two stages.

B. Initialization and Graph Sparsification

Given $\mathbf{X}$ and $\dot{\mathbf{Y}}^{\mathbf{0}}$, the first task is to learn a discriminative feature map $\mathcal{V}^0(\cdot)$ and generate an initial underlying, similarity, graph for the learnt feature map. In the following, we use superscript to denote the iteration number, where ^{0} corresponds to the initialization step. Let

$$d_{i,j}(\mathcal{V}^0) = \|\mathcal{V}^0(\mathbf{x}_i) - \mathcal{V}^0(\mathbf{x}_j)\|_2^2,$$

be the Euclidean distance between the feature maps corresponding to data samples $\mathbf{x}_i$ and $\mathbf{x}_j$.

To achieve computational efficiency, it is desirable the underlying similarity graph to be a sparsely connected graph [18], [20]. To construct a sparse graph, various graph sparsification methods are possible, the most popular being a KNN graph, where an edge $e_{i,j}$ is kept only if node ψ_j is one of the k closest neighbours (in terms of distance $d_{i,j}$) to node ψ_i. That is, each graph edge $e_{i,j}$ is set to:

$$e_{i,j} = \begin{cases} 1, \text{if } \psi_j \text{ is one of } \gamma \text{ closest neighbour to } \psi_i \\ 0, \text{otherwise.} \end{cases} \tag{1}$$

Once an edge matrix $\mathbf{E}^0 = \{e^0_{i,j}\}$ is computed through $\mathcal{V}^0(\cdot)$ and (1), we obtain an initial undirected and unweighted graph $\mathcal{G}^0 = (\mathbf{\Psi}, \mathbf{E}^0, \mathbf{W}^0)$, where $\mathbf{W}^0$ is set to an $N \times N$ all-ones matrix with each element being set to one. The block diagram is shown in Fig. 1. $\gamma^0 = \gamma$ is learnt as explained in Sec. IV-A. We note that resulting $\mathcal{G}^0$ is not necessarily, a K-NN graph.

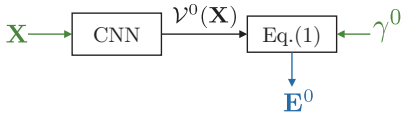


Fig. 1: The block diagram of the unweighted graph generation scheme. $\mathcal{V}^0(\cdot)$ is a CNN-based feature map learnt by minimizing a loss function in order to reflect the node-to-node correlation. The implementation of the proposed CNN and loss function is described in Sec. IV-A.

C. Proposed Classifier Learning with Iterative Graph Update

If the noisy training labels are seen as a smooth graph signal, $\mathbf{Y}^0$, then one can alternately perform GLR for denoising the labels and performing semi-supervised classification, and feature learning for refining the set of deep feature maps and the underlying graph.

Let $r > 0$ be the iteration index, initialized to 1, and let $\mathcal{G}^r = (\mathbf{\Psi}, \mathbf{E}^{r-1}, \mathbf{W}^r)$ be the graph, with $\mathbf{W}^r$ to be learnt, and $\mathbf{Y}^r$ the noisy labels in the r-th iteration. Thus, $\mathcal{G}^1$ is an N-node graph with edges set by (1). In the r-th iteration, each vertex ψ_i is indexed by a label $y_i^{r-1} \in \mathbf{Y}^{r-1}$ (graph signal), and is associated to a feature vector $\mathcal{V}^r(\mathbf{x}_i)$.

Typically, the edge weight is computed using a Gaussian kernel function with a fixed scaling factor σ, i.e., $\exp\Big(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\Big)$, to quantify the node-to-node correlation. Instead of using a fixed σ as in [16], [18], [19], motivated by [44], we introduce an auto-sigma Gaussian kernel function to assign edge weight $w^r_{i,j}$ in $\mathcal{G}^r$ by maximizing the margin between the edge weights assigned to $\mathcal{P}$-edges and $\mathcal{Q}$-edges, as:

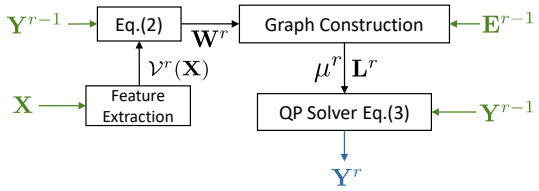
$$\begin{aligned} \sigma^* &= \underset{\sigma, e_{a,p} \in \mathcal{P}, e_{a,n} \in \mathcal{Q}}{\arg\max} \Big[\exp\Big(-\frac{\omega^2_{\{a,p\}}}{2\sigma^2}\Big) - \exp\Big(-\frac{\omega^2_{\{a,n\}}}{2\sigma^2}\Big)\Big], \\ w^r_{i,j} &= \exp\Big(-\frac{\|\mathcal{V}^r(\mathbf{x}_i) - \mathcal{V}^r(\mathbf{x}_j)\|_2^2}{2\sigma^{*2}}\Big) \end{aligned} \tag{2}$$

where $\omega_{\{a,p\}}$ and $\omega_{\{a,n\}}$ denote the mean Euclidean distances between nodes ψ_a and ψ_p connected by $\mathcal{P}$-edges and between nodes ψ_a and ψ_n connected by $\mathcal{Q}$-edges, respectively. By setting the first derivative to zero, we obtain the resulting optimal $\sigma^* = \sqrt{\frac{\omega^2_{\{a,n\}} - \omega^2_{\{a,p\}}}{2\log(\omega^2_{\{a,n\}}/\omega^2_{\{a,p\}})}}$, which is used to assign edge weights of the graph.

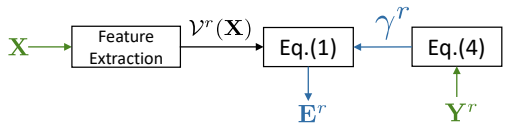
After calculating $w_{i,j}^r$ as in Eq. 2, $\mathbf{A}^r$ and $\mathbf{D}^r$ are calculated (see Section IIIA) and $\mathbf{L}^r = \mathbf{D}^r - \mathbf{A}^r$. $\mathbf{L}^r$ is fixed for the optimization problem in Eq. (3). The optimal solution to Eq. (3) can be found for a given, fixed, $\mathbf{L}^r$. $\mathbf{L}^r$ is determined by the underlying, similarity graph, which is determined via deep feature learning and sparsified via Eq. (2) and Eq. (1). Then, closely following related work [18], [20], we obtain the restored classifier signal by finding the smoothest graph signal $\mathbf{Y}^r$ as:

$$\mathbf{Y}^r = \arg\min_{\mathbf{B}} \ (\|\mathbf{Y}^{r-1} - \mathbf{B}\|_2^2 + \mu^r \mathbf{B}^\top \mathbf{L}^r \mathbf{B}). \tag{3}$$

The minimization above finds a solution that is close to the observed set of labels in the previous iteration, $\mathbf{Y}^{r-1}$, while preserving smoothness. To guarantee that the solution $\mathbf{Y}^r$ to the quadratic programming (QP) problem (3) is numerically stable, we adopt Theorem 1 from [20] by setting an appropriate conditional number κ. The maximum value of the smoothness prior factor μ^r is then calculated as: $\mu_{max}^r = (\kappa - 1)/(2d_{max}^r)$, where d_{max}^r is the maximum degree of the vertices in graph $\mathbf{G}^r$. See Fig. 2(a).



(a) The block diagram of the proposed classifier scheme. CNN weights are learnt by minimizing the loss function to better reflect the node-to-node correlation. The edge matrix $\mathbf{E}^{r-1}$ is used as a mask when assigning edge weights to construct adjacency matrix $\mathbf{A}^r$. We perform GLR to restore the corrupted classifier signal $\mathbf{Y}^{r-1}$ given the resulting sparse graph Laplacian $\mathbf{L}^r$ and apply the constrained smoothness prior factor μ^r to ensure the numerical stability of QP solver. The implementation of $\mathcal{V}^r(\cdot)$ varies depending on the data scale and the dimension of the input observations. The output is the new set of 'denoised' labels $\mathbf{Y}^r$.



(b) The block diagram of the proposed graph update scheme. Based on the adjacency matrix $\mathbf{A}^r$ and restored classifier signal $\mathbf{Y}^r$, we learn a CNN to better refine the graph structure. The edge matrix $\mathbf{E}^r$ is updated via (4) and (1) based on both the previous restored classifier signal and the regularized deep feature map. The output of this block is the new edge matrix $\mathbf{E}^r$ that will be used in the next iteration.

Fig. 2: The proposed graph-based classifier and graph update scheme. The green and blue colors denote input and output, respectively. The implementation details are given in Sec. IV-B.

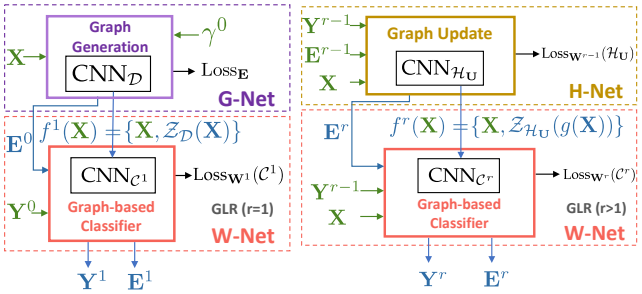
Between two GLR iterations, we use CNN to refine the feature map based on the denoised label signal, $\mathbf{Y}^{r-1}$ obtained in the previous GLR iteration. See an illustration in Fig. 2(b) for the graph update after the r-th GLR iteration. We update the individual degree of Vertex ψ_i as:

$$\begin{aligned} \mathring{e}_{i,j}^r &= \begin{cases} 1, & \text{if } e_{i,j}^{r-1} \in \mathcal{P}^r \ \& \ a_{i,j}^r > 0.1 \\ 0, & \text{if } e_{i,j}^{r-1} \in \mathcal{Q}^r \ \& \ a_{i,j}^r \leq 0.1, \end{cases} \\ \gamma_i^r &= \sum_{j=1}^{N} \mathring{e}_{i,j}^r, \end{aligned} \tag{4}$$

where $\mathcal{P}^r$ and $\mathcal{Q}^r$ sets are formed based on the denoised classifier signal $\mathbf{Y}^r$. The edge $e_{i,j}^r$ is heuristically removed if it connects vertices with opposite labels or the corresponding entry to adjacency matrix is less than a hard threshold of 0.1. Note that the value of the threshold is not critical during inference and affects only the early few training batches in H-Net. Therefore, it is suggested to be a very small value to determine whether an edge should exist in $\mathcal{G}^r$.

IV. Proposed Network

Based on the concepts described in the previous section, next we present the algorithmic flow and describe the architecture used to implement the proposed DynGLR network.



(a) GLR (r=1) **(b)** GLR ($r = 2$)

Fig. 3: The block diagram of the proposed DynGLR-Net. As shown in (a), given observations $\mathbf{X}$, G-Net (see Subsec. IV-A) first learns an initial undirected and unweighted KNN-graph by minimizing $\text{Loss}_{\mathbf{E}}$. The resulting edge matrix $\mathbf{E}^0$ is then used in the following, first, GLR iteration. The learned shallow feature map $f^1(X) = \{X, \mathcal{Z}_{\mathcal{D}}(X)\}$ is then used as input to learn a $\text{CNN}_{\mathcal{C}^1}$ network for assigning weights to the initial graph edges. Given a subset of, potentially noisy labels, $\dot{\mathbf{Y}}$, we perform GLR on the constructed undirected and weighted graph to restore the labels. The resulting restored labels are the prediction results for (a). To facilitate the regularization, one can use the restored labels to update the graph edge sets by minimizing $\text{Loss}_{\mathbf{W}^1}(\mathcal{H}_{\mathbf{U}})$ given neighbor information for each node based on the resulting denoised classifier signal from the previous GLR iteration. We then reassign edge weights to the updated graph edge sets to perform better node classification as shown in (b). Theoretically, (b) can be applied multiple times to improve predictions until convergence is reached.

The block diagram of the proposed DynGLR-Net is presented in Fig. 3. Our overall network consists of three subnetworks: (1) **G-Net** (graph generator network) used to learn a deep metric function to construct an initial undirected and unweighted KNN graph $\mathcal{G}^0 = (\mathbf{\Psi}, \mathbf{E}^0, \mathbf{W}^0)$. (2) **W-Net** (graph weighting and classifier network) used to assign edge weights (see Eq. 2) for effectively performing GLR to restore the corrupted classifier signal $\mathbf{Y}^r$. (3) **H-Net** (graph update network) used to refine $\mathbf{E}^r$ to better reflect the node-to-node correlation based on the restored classifier signal in

the previous iteration $\mathbf{Y}^{r-1}$. We clarify each network in the following subsections.

A. G-Net

Given an observation set $\mathbf{X}$, the first task is to to learn the feature metric space, for which we use a CNN, denoted by $\text{CNN}_{\mathcal{D}}$, to learn a mapping function $\mathcal{D}(\cdot)$. The architecture of $\text{CNN}_{\mathcal{D}}$ is shown in Fig. 4.

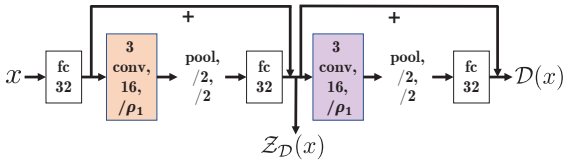


Fig. 4: $\text{CNN}_{\mathcal{D}}$ neural network: 'pool/q/w' refers to a max-pooling layer with q=pool size and w=stride size. 'x conv y/ρ_1' refers to a 1D convolutional layer with y filters each with kernel size x and stride size ρ_1. 'fc x' means the fully connected layer with x=number of neurons. The stride size ρ_1 varies depending on the input data (see details in Sec. V-A).

For a random observation triplet $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$, such that $e_{a,p} \in \mathcal{P}$ and $e_{a,n} \in \mathcal{Q}$, we minimize the following loss function to learn the feature map:

$$Loss_{\mathbf{E}} = \sum_{a,p,n} \left[\alpha_{\mathbf{E}} - \|\mathcal{D}(\mathbf{x}_a) - \mathcal{D}(\mathbf{x}_n)\|_2^2 + \|\mathcal{D}(\mathbf{x}_a) - \mathcal{D}(\mathbf{x}_p)\|_2^2\right]_+, \tag{5}$$

where $\mathcal{D}(\cdot)$ is a CNN-based feature map function, to be learnt, that returns a feature vector corresponding to the input observation, $\alpha_{\mathbf{E}}$ is the minimum margin, and operator $\left[\cdot\right]_+$ is a Rectified Linear Units (ReLU) activation function which is equivalent to $\max(\cdot, 0)$. Let $\mathcal{Z}_{\mathcal{D}}(\mathbf{x})$ be the learnt feature map output at the second to the last layer of $\text{CNN}_{\mathcal{D}}$ (see Fig. 4) obtained by minimizing the loss (5).

The loss function (5) promotes a community structure graph that has relatively small Euclidean distance between the feature maps of vertices connected by the edges in $\mathcal{P}$, and a large distance between the vertices connected by the edges in $\mathcal{Q}$, while keeping a minimum margin $\alpha_{\mathbf{E}}$ between these two distances; i.e., if the class boundaries are below $\alpha_{\mathbf{E}}$, a penalty will be applied. Note that we use triplet loss [24], which is different from the standard hinge loss. The standard hinge loss focuses on finding the optimal decision boundary between two classes, while the triplet loss introduces a penalty when the decision boundaries between different classes are smaller than the pre-defined margin α_E.

Since we do not have a priori knowledge of the connectivity of the nodes, we generate the initial graph as a fully connected graph; justification for starting with a fully connected graph is provided in [45]. A sparse $\mathbf{E}^0$ minimizes the number of $\mathcal{Q}$-edges by keeping only the connections with γ^0 neighbors per individual node. We adopt a K-NN-graph construction based on (1), where maximum number of neighbors γ^0 is obtained via grid-search by evaluating classification accuracy of the KNN classifier using the validation data with the same amount of noisy labels as the training dataset. Note that, as we do not have any prior knowledge of an appropriate maximum degree of each individual node, we initially set all $\gamma^0 = \gamma_1 = \ldots = \gamma_N$. Once the number of neighbors γ^0 is obtained, the resulting graph edges $\mathbf{E}^0$ are used in the following section for pruning edge weights during edge weighting and are updated based on the regularized metric function and the difference between the classifier signal, before and after GLR.

B. W-Net

Starting with iteration $r = 1$, to assign edge weights $\mathbf{W}^r$ to the graph $\mathcal{G}^r$, we first employ a CNN, denoted by $\text{CNN}_{\mathcal{C}^r}$, to learn a deep metric function. We propose a robust graph-based triplet loss function to better learn feature map $\mathcal{V}^r$, as:

$$\begin{aligned} Loss_{\mathbf{W}^r}(\mathcal{V}) = \sum_{\psi_a,\psi_p,\psi_n} & \left[\alpha_{\mathbf{W}} - \|\mathcal{V}^r(f^r(\mathbf{x}_a)) - \mathcal{V}^r(f^r(\mathbf{x}_n))\|_2^2 \right. \\ & \cdot \pi_{(\psi_a,\psi_n | e_{a,n} \in \mathcal{Q})} + \|\mathcal{V}^r(f^r(\mathbf{x}_a)) - \mathcal{V}^r(f^r(\mathbf{x}_p))\|_2^2 \\ & \left. \cdot \pi_{(\psi_a,\psi_p | e_{a,p} \in \mathcal{P})}\right]_+ \\ \Pi^r = & \{\pi_{\psi_i,\psi_j}\} = \{\Theta(\dot{y}_i^r, \dot{y}_i^{r-1}, \dot{y}_j^r, \dot{y}_j^{r-1})\}. \end{aligned} \tag{6}$$

Θ is an edge attention activation function (see (7) for the particular function used) that estimates how much attention should be given to each edge and $\mathbf{Y}^r = \{\dot{\mathbf{Y}}^r = [-1,1]^M, [-1,1]^{N-M}\}$ is the restored classifier signal obtained via (3) starting from the classifier signal in the previous iteration, $\mathbf{Y}^{r-1}$. π_{ψ_i,ψ_j} is the amount of attention, i.e., edge loss weights, assigned to the edge connecting vertices ψ_i and ψ_j. Let $\mathcal{C}^r(\cdot)$ be the feature map learnt by minimizing Eq. (6).

Note that, as defined in Sec.III.A *triplets* as observations (x_a, x_p, x_n), $x_a, x_p, x_n \in \mathbf{X}$ corresponding to vertices $\psi_a, \psi_p, \psi_n \in \mathbf{\Psi}$, respectively, such that $y_a = y_p \neq y_n$, and $y_a, y_p, y_n \in \dot{\mathbf{Y}}$. Following the above definitions, given one set of triplets (x_a, x_p, x_n) and their features, we compute two distances: (1) Euclidean distance between x_a and x_p, i.e., distance between nodes with same label (2) Euclidean distance between x_a and x_n, i.e., distance between nodes with opposite labels. α_E and α_W are defined as minimum margins between those two aforementioned distances in triplet loss Eq.(5) and its weighted version Eq.(6). That is, if the class boundaries are below these margins, a penalty will be applied.

The architectures for $r = 1$ and $r = 2$ are shown in Fig. 5. Since, at the first iteration $r = 1$, we expect many noisy labels, the network architecture will be different to the $r > 1$ case.

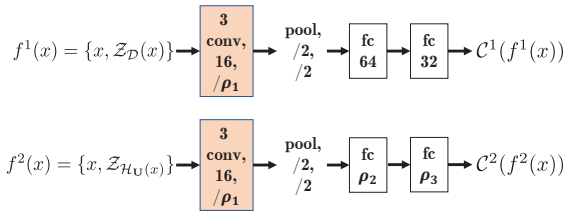


Fig. 5: $\text{CNN}_{\mathcal{C}^r}$ neural nets. The stride ρ_1 and the number of neurons ρ_2, ρ_3 vary depending on the input data (see details in Sec. V-A).

The architecture presented in Fig. 5 (top) is used as the feature map $\mathcal{C}^1(\cdot)$, after G-Net, to construct the graph $\mathcal{G}^1 = (\mathbf{\Psi}, \mathbf{E}^0, \mathbf{W}^1)$ by minimizing $\text{Loss}_{\mathbf{W}^1}(\mathcal{C}^1)$ taking as input undirected graph $\mathcal{G}^0$ learned via G-Net. The input to $\text{CNN}_{\mathcal{C}^1}$ is the concatenated observation $\mathbf{X}$ and "shallow feature maps"

learned via G-Net, i.e., the output of the second to last layer of $\text{CNN}_{\mathcal{D}}$ as presented in Fig. 4, denoted by $\mathcal{Z}_{\mathcal{D}}(\mathbf{X})$.

The $r = 2$ architecture is shown in Fig. 5 (bottom), with observation $\mathbf{X}$ and "shallow feature maps" learned via H-Net (described in the next subsection) to facilitate the regularization of $\text{CNN}_{\mathcal{C}^2}$ by minimizing $\text{Loss}_{\mathbf{W}^2}(\mathcal{C}^2)$ based on the denoised labels, convolution on both feature maps, denoised classifier signal and their differences across neighbors.

Note that as we compare our results with the related work [24] "DML-KNN", we evaluate the similarities of flatten feature maps in Euclidean space. Using the triplet loss proposed in [24], we observe that fully connected layer with heuristically 32 neurons in $\text{CNN}_{\mathcal{D}}$ achieve the best results in the validation set with clean labels. Therefore, in f^1, we heuristically add another fully connected layer with $64 = 32 \cdot 2$ neurons before the second fully connected layer with 32 neurons to expand the model capacity for learning deeper metric space.

Differently, the input feature maps to f^2 comprise two parts. The first part is the original input features to the overall network. The second part is the concatenated feature maps of the neighbor shallow feature maps from the previous H-Net. The number of combined input feature maps to the second W-Net can be much larger than the inputs to the first W-Net. As a result, ρ_2 and ρ_3 can be set differently to appropriately match model capacity for feature representation.

Unlike [24], we introduce edge attention activation Θ in (6) to dropout some edges with relatively large changes between $\dot{\mathbf{Y}}^r$ and $\dot{\mathbf{Y}}^{r-1}$ via GLR. This helps to focus learning on edges with high confidence given noisy training labels. Therefore, the overall training performance is better than the standard dropout layer approach, which drops out random neuron units in the network. We implement the edge attention activation Θ and Φ as:

$$\Phi(\dot{y}_i^{r-1}, \dot{y}_i^r) = \begin{cases} 1, & \text{if } |\dot{y}_i^{r-1} - \dot{y}_i^r| \leq \varepsilon^r \\ 0, & \text{if } |\dot{y}_i^{r-1} - \dot{y}_i^r| > \varepsilon^r \end{cases} \tag{7}$$

$$\Theta(\dot{y}_i^{r-1}, \dot{y}_i^r, \dot{y}_j^{r-1}, \dot{y}_j^r) = \min(\Phi(\dot{y}_i^{r-1}, \dot{y}_i^r), \Phi(\dot{y}_j^{r-1}, \dot{y}_j^r)),$$

where threshold ε^r is used to determine whether a node's label can be trusted and also helps to control the sparsity of edge attention matrix $\mathit{\Pi}^r$. That is, if the difference between the signal label in the previous and current iteration is large, this means that the label most likely changed sign (from -1 to +1 or vice versa) and is unreliable in this iteration. To reflect the fact that there might be many noisy (unreliable) labels at the start we heuristically set $\varepsilon^1 = 0.6$ for the first GLR iteration (see the results in Sec. V). Since after applying GLR the classification signal is expected to be cleaner, we heuristically set threshold $\varepsilon^2 = 0.15$ for the second stacked W-Net during training to ensure that we regularize CNNs with less concern about the over-fitting issue introduced by noisy labels.

Note that the proposed G-Net is trained to generate a K-NN graph that maximizes the K-NN classifier's accuracy in the validation set using a standard triplet loss function. When the training labels are potentially noisy, G-Net can be overfitted to the noisy labels. Hence, we propose a regularized triplet loss by using the denoised labels via GLR and Eq. (7). Alternatively, instead of the triplet loss function, the smoothness loss from [46] or [47], could be used create the deep metric space that generates a smooth graph. For example, for high-dimensional image data, instead of training G-Net, one can use the method of [47] or a pre-trained feature extraction CNN to construct a K-NN graph to perform GLR.

C. H-Net

Edge convolution has recently been shown to be a rich feature representation method [40], [23], [48]. We adopt edge convolution in deeper feature map learning, i.e., after GLR $r = 1$. That is, given $\mathbf{A}^1$, from the first GLR iteration, each node's feature representation is enhanced by considering observations of both $\mathbf{X}$ and classifier signal $\mathbf{Y}^1$ from its six nearest neighbors (set heuristically), which are most-likely to have the same label.

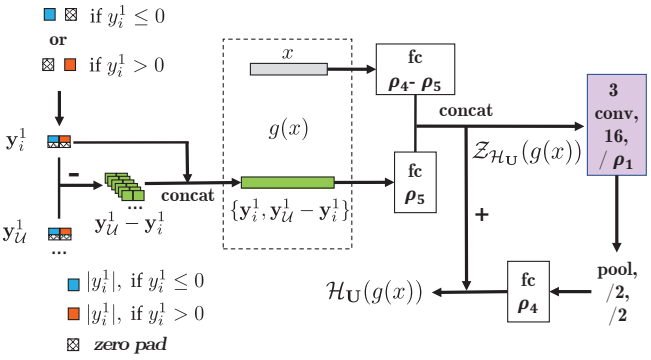


Fig. 6: $\text{CNN}_{\mathcal{H}_\mathbf{U}}$ neural nets. The stride size ρ_1 and the number of neurons ρ_4, ρ_5 vary depending on the input data (see details in Sec. V-A).

Incorporating an additional CNN, denoted by $\text{CNN}_{\mathcal{H}_\mathbf{U}}$, shown in Fig. 6, we construct a richer feature representation $\mathcal{H}_\mathbf{U}(g(\mathbf{x}))$ to enhance the graph-based classifier learning with a single input to the network, $g(x)$, comprising $\mathbf{x}_i$ and $\{\mathbf{y}_i^1, \mathbf{y}_\mathcal{U}^1 - \mathbf{y}_i^1\}$, where $\mathbf{y}_i^1$ denotes a tuple $(y_i, 0)$, if $y_i > 0$ or $(0, y_i)$ otherwise. $\mathbf{y}_\mathcal{U}^1$ is a 6×2 matrix formed by concatenating $\mathbf{y}_i^1$ with the nearest six neighboring nodes. Finally, $\mathbf{y}_\mathcal{U}^1 - \mathbf{y}_i^1$ is obtained by subtracting each row of $\mathbf{y}_\mathcal{U}^1$ by $\mathbf{y}_i^1$.

Graph edge $\mathbf{E}^{r-1}$ is updated by (4) based on the learnt regularized feature map $\mathcal{H}_\mathbf{U}(\cdot)$ in order to better reflect the node-to-node correlation. The new edge matrix $\mathbf{E}^1$ and the denoised classifier signal $\mathbf{Y}^1$ are then used in the second graph-based classifier iteration.

The general intuition behind H-Net is as follows. The classifier output Y^1 and the input X are aggregated as inputs to the H-Net for regularizing the feature representations. As proposed in [26,43,52], the edge convolution is performed on aggregated feature maps over neighboring edges. Instead we use the node feature maps and the corresponding classifier outputs over its neighboring edges. After GLR, the denoised classifier output Y^1 can better reflect the actual labels, which is now not overfitted to the potential noisy training labels. Therefore, the lower-dimensional Y^1 can better propagate the edge information of each node and facilitate the classifier learning for quicker convergence rather than performing edge convolutions on a higher dimensional feature space.

Note that, in our experiments, $\text{CNN}_{\mathcal{D}}$, $\text{CNN}_{\mathcal{C}^1}$, $\text{CNN}_{\mathcal{H}_U}$, $\text{CNN}_{\mathcal{C}^2}$ are implemented for 1D data; hence, we use asymmetric convolutions. However, the architecture design would vary according to the structure of the input data. For instance, one can use standard 3x3 convolutions on image data and pooling schemes such as Global Average/Max Pooling [49].

Though we can continue iterating between W-Net and H-Net as a refined graph-based classifier cascade (see Fig. 3b), in our practical implementation, for our test datasets, after two iterations, we did not observe much difference between input and output graph classifier signal when $r > 2$ (see our evaluations of graph update block and graph spectrum analysis in Sec. V-B).

V. Simulations

In this section, we present our simulation results, including the ablation study, visualization results, and comparison of the performance against different, classic and state-of-the-art classifiers, under different label noise levels.

A. Simulation setup: Datasets, Benchmarks, Parameters and Performance Measure

1) Datasets: We select three binary-class datasets from Knowledge Extraction based on Evolutionary Learning dataset (KEEL) [50] that vary in the number and type of features; these sets are, from low dimensional feature sets to higher ones: (1) **Phoneme**: contains nasal (class 0) and oral sounds (class 1), with 5404 instances (frames) described by 5 phonemes of digitized speech. (2) **Magic**: contains images generated by primary gammas (class 0) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (class 1), where 19020 instances are generated for simulation using the imaging technique, with each instance containing 10 attributes to characterize simulated images. (3) **Spambase**: to determine whether an email is spam (class 0) or not (class 1), with 4597 email messages summarized by 57 particular words or characters. (4) In addition to KEEL dataset, we include the **CIFAR-10** dataset for the purpose of demonstrating effectiveness of the proposed approach on 2D image data; **CIFAR-10** contains 50000 pre-defined training images and 10000 testing images in 10 classes [51]. We use all images in class 'airplane' and 'ship' to construct a binary classification dataset.

2) Benchmark classifiers: We compare the proposed network against 10 different classification methods: (1) SVM with radial basis function kernel (SVM-RBF) and linear kernel (SVM-Linear) (2) a classical CNN, consisting of two CNN blocks and two fully connected layers afterwards, where each CNN block has a convolution layer, a max pooling layer and one dropout layer (3) a graph CNN with multiple graph construction blocks, where each block constructs a KNN graph based on multiple graph structures learnt via edge convolution; batch normalization with decay is used (called DynGraph-CNN [23]) (4) a KNN classifier using CNN-based deep metric learning (used CNN is the same as $\text{CNN}_{\mathcal{D}}$) [24] (called DML-KNN) (5) a rank-sampling [52] based KNN classifier using CNN-based deep metric learning (CNN used is same as in DynGraph-CNN), where sampling is performed on the training set by calculating the resulting classification accuracy using randomly sampled samples. We use the top 480 training samples with relatively high classification accuracy in the validation set. During inference, 480 selected training samples are divided into 6 equal-size batches by stratified random sampling and the predictions using each batch are averaged to obtain the final decision (6) label noise robust SVM with RBF kernel [25] (LN-Robust-SVM-RBF) and linear kernel (LN-Robust-SVM-Linear) (7) a graph-based classifier with negative edge weights assigned between the centroid sample pairs and between the boundary sample pairs (named Graph-Hybrid [18]) (8) a CNN network (same as in DynGraph-CNN) trained by savage loss (called CNN-Savage [13]) (9) a CNN network (same as in DynGraph-CNN) trained by bootstrap-hard loss (called CNN-BootStrapHard [14]) (10) a CNN network (same as in DynGraph-CNN) trained via dimensionality-driven learning strategy (called CNN-D2L [5]). Note that Classifiers (1)-(4) are classical methods for normal, 'noise-free', conditions, and methods (5)-(9) are proposed to avoid overfitting under noisy training labels. All CNN-based methods adopt l_2 regularization for each layer. Similar to Benchmark (5), we also adopt rank-sampling technique on the training set to select trusted samples that will further facilitate the predictive performance and consistency of our model, denoted by 's' appended to the model name.

3) Ablation study: To understand how different components of our proposed architecture affect the results, we perform an ablation study by removing some components. The resulting architectures are denoted by DynGLR-G-number, where 'G' refers to Graph generation and '1' refers to edge weighting, '2' to GLR, and '3' graph update. That is, the following variants of the proposed scheme are compared: (1) DynGLR-G-2: we import the unweighted graph $\mathcal{G}^0$ generated by G-Net (see Fig. 1) into GLR for classification. (2) DynGLR-G-12: we assign weights to the unweighted graph $\mathcal{G}^0$ via an adaptive Gaussian kernel function (see (Eq. 2)); the resulting undirected and weighted graph is then used to perform node classification via GLR. (3) DynGLR-G-1232: we update the graph edge sets by considering the neighbors of each node with denoised classifier signal and observed feature maps (see Fig. 2); the resulting unweighted graph is then used for classification. (4) DynGLR-G-12312: we reassign weights to the updated unweighted graph to effectively perform classification; we perform rank-sampling for all architectures to evaluate the benefits, denoted by 's' appended to the name of each proposed architecture.

4) Simulation setup, performance measure and parameters: We design our experiments by splitting the first three datasets into training, validation and testing sets with 40%, 20%, 40% of instances, respectively, with balanced class distribution. We adjust the implementation of Fig. 5 to input 2D image data for **CIFAR-10** dataset, follow the configurations of [51] for the experimental setup and use 10% of training images as validation set. The validation sets are used to perform early stop [53] for all CNN-based methods. To evaluate the robustness of different classification methods against label noise, we randomly sample subsets of instances from both training and validation sets and

reverse their labels. Classification error rates are measured by running 20 experiments per label noise level (0% to 25%). We use the same random seed setting across all classification methods and remove all duplicated instances to ensure a fair comparison.

Hyper-parameters used for each experiment are obtained from the validation sets by grid search. All used parameters for the KEEL datasets are listed in Table I.

TABLE I: Parameters for the proposed architectures. $x \Rightarrow y$ means that the learning rate decreases linearly from x to y with the epoch number.

Hyper-parameters	***Phoneme***	***Magic***	***Spambase***
$\rho_1, \rho_2, \rho_3, \rho_4, \rho_5$	1,256,64,256,6	1,128,32,128,4	2,32,32,64,6
G-Net learning rates	0.02⇒0.01	0.02⇒0.01	0.02⇒0.01
G-Net epochs	160	160	60
W-Net(r=1) learning rates	0.02⇒0.01	0.02⇒0.01	0.02⇒0.012
W-Net(r=1) epochs	320	320	80
H-Net learning rates	0.002⇒0.001	0.002⇒0.001	0.002⇒0.001
H-Net epochs	120	180	100
W-Net(r=2) learning rates	0.01⇒0.002	0.01⇒0.002	0.02⇒0.01
W-Net(r=2) epochs	60	40	40

For the image dataset, CIFAR-10, we use Resnet-152 (pre-trained on ImageNet dataset, following [54]) as CNN_D to extract the feature embeddings as the inputs to all methods. We adjust the implementations [1] in Fig.5 and Fig.6 to target image data for CIFAR-10 datasets. For example, input vector f^1 is of length 2048, and is reshaped to 32x64. The first fully connected layer is replaced by a global average pooling layer and a Residual block is added before the pooling layer.

To guarantee the solution $\mathbf{Y}^r$ to (3) is numerically stable, we heuristically set conditional number $\kappa = 60$ and $\mu^r = 0.67\mu^r_{max}$ in all our experiments. Based on the results on the validation set, we use the distance margin $\alpha_{\mathbf{E}} = \alpha_{\mathbf{W}} = 10$ in both (5) and (6). For each epoch, we use batch size of 16, each batch comprising 80 labeled instances from training set and 20 unlabeled instances from validation set; thus we randomly select $16 \cdot 100$ instances. This results in 16 graphs to regularize training per batch.

All CNNs are learnt by ADAM optimizer, classification accuracy and classifier signal changes are used as metric for rank-sampling for further improving the predictive performance.

B. Results and Discussion

This section is organized as follows. As part of the ablation study, Subsecs. V-B1 and V-B3 evaluate the ability of the proposed schemes to clear the noisy labels and observation samples during the training phase, respectively. We analyze the sensitivity of hyper-parameters that affect the regularization performance in Subsec. V-B2. We show the effectiveness of our iterative graph update scheme in Subsec. V-B4 by visualizing the learnt underlying graph in spectral domain. To analyze the impact of different components on the classification accuracy, we show the classification error rates in Subsec. V-B5 and discuss the findings of our ablation study during the testing phase and show comparison with state-of-the-art schemes.

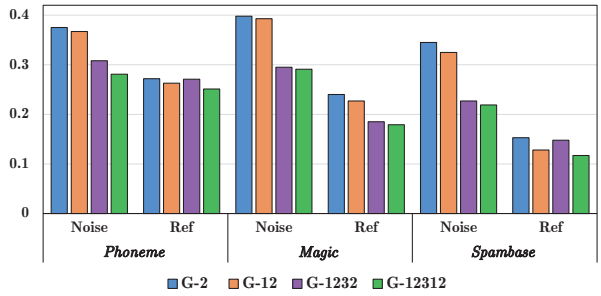


Fig. 7: Mean Edge Weight Proportion ϱ for the proposed DynGLR-Nets for all three datasets when 25% labels used for training are wrong (denoted by 'Noise') or when all training labels are correct (denoted by 'Ref' without use of GLR). G-2, G-12, G-1232, and G-12312 schemes are described in the previous subsection.

1) Evaluating graph update block: First, we evaluate ability of the graph update block to clear noisy labels. We use the mean edge weight proportion measure defined as:

$$\varrho = \frac{\sum_{p,n}^{N} w_{p,n}}{\sum_{i,j}^{N} \mathbb{1}(w_{i,j} > 0)}, \tag{8}$$

where ψ_p and ψ_n are two nodes with the opposite labels, $w_{p,n}$ is the weight of the edge $e_{p,n}$ and $\mathbb{1}(c)$ is an operator that returns 1 if condition c is fulfilled, and 0 otherwise. The results are shown in Fig. 7, which shows that, for all datasets, the number of connections between nodes with opposite labels decreases with iterations and becomes similar to that without any noise, indicating that the graph update manages to restore the noisy labels.

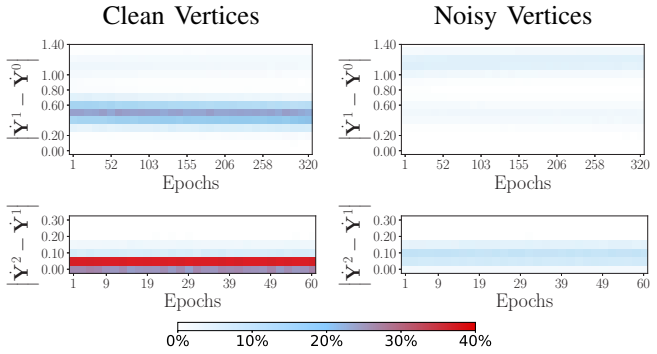


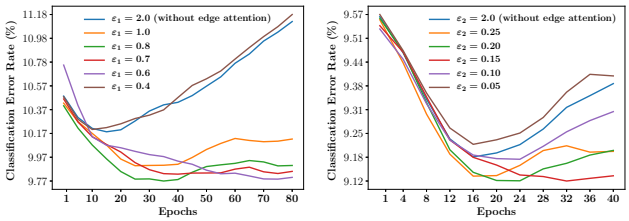
Fig. 8: Classifier signal changes $|\dot{\mathbf{Y}}^{r-1} - \dot{\mathbf{Y}}^{r}|$ density visualization for ***Phoneme*** dataset after the first (top row) and second (bottom row) GLR iteration during the training period. Vertices with clean/incorrect labels are shown in the first/second column. The intensity of the classifier signal changes across all experiments are represented through colormaps.

In Eq. (7) we use a threshold to distinguish reliable nodes from unreliable nodes. In order to evaluate the used approach and to set thresholds, we show the change of the classifier signal $|\dot{\mathbf{Y}}^{r-1} - \dot{\mathbf{Y}}^{r}|$ during the first two iterations in Fig. 8. We can see that when all the labels are clean (left column) the difference between the signals before and after GLR is mainly below 0.6 and 0.15, in the first and the second iteration, respectively. Thus, by setting the thresholds at $\varepsilon^1 \approx 0.6$

[1]Code available:https://github.com/yemx21/DynGLR

and $\varepsilon^2 \approx 0.15$ for the first and the second GLR iteration, respectively, we can distinguish the vertices with potentially noisy labels. Similar observations are made for the ***Spambase*** and ***Magic*** datasets.

Heuristically, we observed that as more GLR iterations are performed, the overlap between the clean and noisy vertices distribution of classifier signal changes is high, resulting in reduced ability to use thresholding for distinguishing if a node is sufficiently cleaned. That is why in all our experiments, to reduce complexity, we perform the graph update only after the first iteration, i.e., for r=1. We next assess sensitivity of the network to threshold values.



(a) GLR (r=1) **(b)** GLR (r=2)

Fig. 9: Classification Error Rate (%) for ***Spambase*** dataset using different $\varepsilon^{1,2}$ in DynGLR-G-12312.

2) Hyper-parameter sensitivity: We use attention activation (Eq. 7) to detect the label of vertex ψ_i as possibly noisy if $\Phi(\dot{y_i}^{r-1}, \dot{y_i}^{r}) = 1$. To analyze the sensitivity of hyper-parameters ε^1 and ε^2 (thresholds) in (Eq. 7), we show the classification error rate for the ***Spambase*** dataset during training using different values of ε^1 and ε^2 in Fig. 9. We observe that the thresholds $\varepsilon^1 = 0.6, \varepsilon^2 = 0.15$, from the density visualization of classifier signal changes $|\dot{\mathbf{Y}}^{r-1} - \dot{\mathbf{Y}}^{r}|$ of Fig. 8, are appropriate to improve the regularization of CNNs. We also show that the classification error rate reduces from first GLR iteration to second GLR iteration.

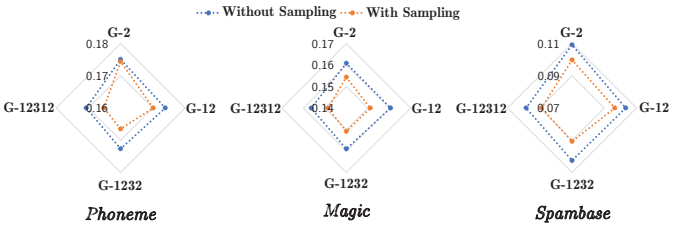


Fig. 10: The mean noise level of the training labels after GLR is performed in all proposed DynGLR-Nets when 25% labels used for training are incorrect.

3) Evaluating Rank-sampling: To evaluate the denoising effects of GLR, in Fig. 10, we show the mean noise level of the training labels after GLR is performed across all our proposed architectures with and without sampling. It is clear from Fig. 10, for all three datasets, that the mean noise level is lower with rank-sampling than without. This confirms that with rank-sampling, one can further reduce the effects of noisy training labels by dropping out the less reliable training samples.

4) Graph spectrum visualization: Graph Fourier Transform (GFT) is another approach to represent the smoothness and connectivity of an underlying similarity graph. As in [55], we visualize the magnitude of the GFT coefficients in Figures 11 and 12 along the graph update iterations.

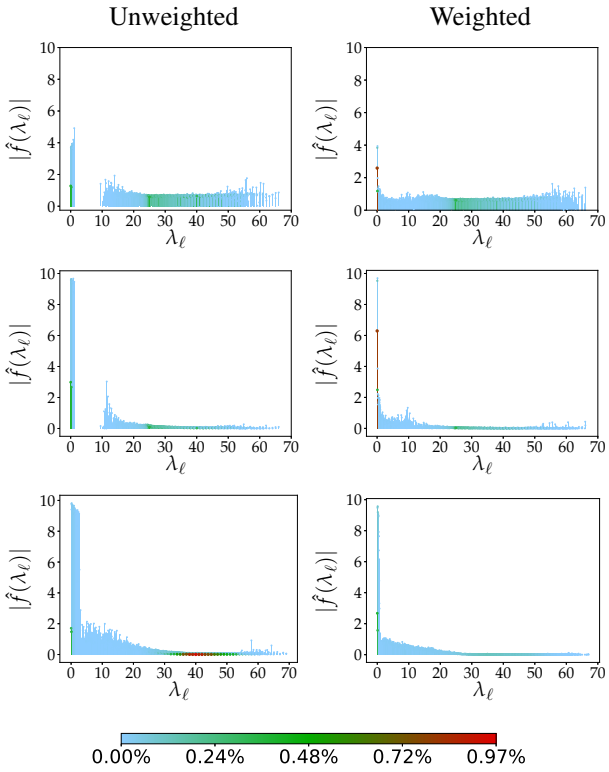


Fig. 11: The magnitude of the Graph Fourier Transform coefficients for ***Phoneme*** Dataset. The density of each Eigenvalue λ_ℓ across all experiments on the testing sets is represented through colormaps. Top row shows the result after initialization and before GLR (G-Net output) and the second and third row show the result after the first ($r = 1$) and the second iteration ($r = 2$), respectively.

In accordance with [55], where it is shown that the magnitude of GFT coefficients decay rapidly for a smooth signal, in our results, we can clearly observe that the magnitude of GFT coefficients is decaying more rapidly along spectral frequencies once the graph is updated (in iteration $r = 1$). Furthermore, comparing the visualization results in the first and the second column, we can see that the graph weighting (W-Net) smooths the graph data, with low frequency components becoming more prominent.

5) Classification Error Rate Comparison: Tables II, IV, VI, VIII show comparison between the proposed DynGLR networks and all the benchmarks in terms of classification error rate. From the tables, it can be seen that the proposed DynGLR networks outperform all the benchmarks. In the KEEL datasets, we observe a 1-2% reduction in error rates. For the CIFAR-10 dataset, the reduction in error rates is 1.5-14.6%. There is only a 0.25% loss between the no-noise and 25% label noise case with the proposed scheme. Note that for CIFAR-10 dataset, we use ResNet152 (pre-trained on ImageNet dataset) to perform feature extraction on which the predictive performance of SVM benchmarks heavily rely on. Without this, SVM benchmarks would perform much worse.

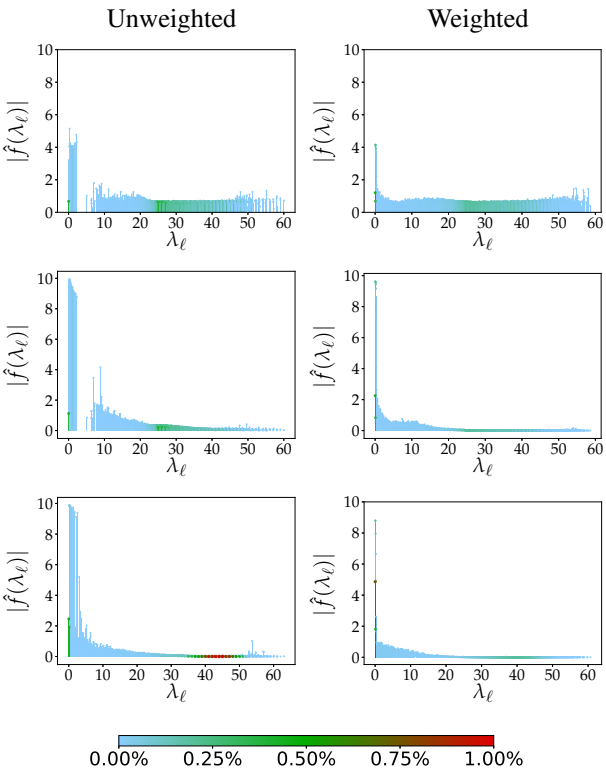


Fig. 12: The magnitude of the Graph Fourier Transform Coefficients for ***Spambase*** Dataset. The density of each Eigenvalue λ_ℓ across all experiments on the testing sets is represented through colormaps. Top row shows the result after initialization and before GLR (G-Net output) and the second and third row show the result after the first ($r = 1$) and the second iteration ($r = 2$), respectively.

TABLE II: Classification Error Rate (%) For ***Phoneme*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	18.33	18.75	19.13	19.57	20.07	20.87
CNN	17.58	17.77	18.00	18.57	19.01	20.00
DynGraph-CNN	17.66	19.04	20.80	22.44	25.20	28.84
DML-KNN	17.04	17.54	17.82	18.58	19.64	21.00
DML-KNN-s	17.01	17.49	17.71	18.43	19.24	20.41
LN-Robust-SVM-RBF*	18.57	19.42	19.65	19.70	20.03	20.28
Graph-Hybrid*	22.01	23.77	25.58	27.97	30.33	33.39
CNN-Savage*	17.52	17.72	18.04	18.51	19.02	19.87
CNN-BootStrapHard*	17.46	17.72	18.00	18.31	18.84	20.15
CNN-D2L*	17.47	17.80	17.96	18.41	18.91	20.04
DynGLR-G-2*	17.04	17.50	17.70	18.34	18.81	20.03
DynGLR-G-12*	16.93	17.36	17.64	18.23	18.52	19.59
DynGLR-G-12s*	16.89	17.36	17.62	18.21	18.52	19.54
DynGLR-G-1232*	16.90	17.29	17.36	18.16	18.48	19.47
DynGLR-G-12312*	16.87	17.19	17.34	18.03	18.38	19.43
DynGLR-G-12312s*	**16.87**	**17.18**	**17.32**	**17.91**	**18.24**	**19.18**

TABLE III: Area Under the Curve For ***Phoneme*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	0.817	0.813	0.809	0.804	0.799	0.793
CNN	0.824	0.822	0.819	0.812	0.808	0.799
DynGraph-CNN	0.823	0.810	0.792	0.776	0.748	0.712
DML-KNN	0.830	0.824	0.821	0.812	0.802	0.791
LN-Robust-SVM-RBF*	0.808	0.806	0.804	0.803	0.800	0.797
Graph-Hybird*	0.780	0.762	0.744	0.720	0.697	0.666
CNN-Savage*	0.824	0.822	0.818	0.813	0.808	0.800
CNN-BootStrapHard*	0.825	0.822	0.819	0.815	0.810	0.798
CNN-D2L*	0.825	0.821	0.819	0.814	0.809	0.799
DynGLR-G-2*	0.830	0.825	0.823	0.816	0.807	0.795
DynGLR-G-12*	0.831	0.826	0.824	0.815	0.807	0.796
DynGLR-G-12s*	0.831	0.826	0.825	0.816	0.809	0.800
DynGLR-G-1232*	0.831	0.827	0.826	0.818	0.812	0.805
DynGLR-G-12312*	0.831	0.828	0.827	0.820	0.813	0.806
DynGLR-G-12312s*	**0.831**	**0.829**	**0.827**	**0.822**	**0.814**	**0.808**

TABLE IV: Classification Error Rate (%) For ***Magic*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	18.42	19.07	19.63	20.18	20.61	21.13
CNN	16.45	16.87	16.91	17.62	18.09	18.86
DynGraph-CNN	17.74	18.69	19.33	21.05	24.15	27.40
DML-KNN	15.33	15.51	15.80	15.94	16.83	18.29
DML-KNN-s	15.33	15.51	15.78	15.89	16.58	17.08
LN-Robust-SVM-RBF*	18.57	18.70	18.80	19.05	19.39	19.82
Graph-Hybrid*	24.82	25.92	27.23	28.84	30.79	33.23
CNN-Savage*	16.31	16.74	16.99	17.41	18.10	18.87
CNN-BootStrapHard*	16.34	16.89	17.02	17.46	18.13	18.65
CNN-D2L*	16.34	16.79	17.21	17.48	18.20	18.75
DynGLR-G-2*	15.35	15.51	15.77	15.94	16.75	18.03
DynGLR-G-12*	15.22	15.47	15.68	15.85	16.60	17.33
DynGLR-G-12s*	15.22	15.47	15.68	15.83	16.52	16.91
DynGLR-G-1232*	15.22	15.46	15.66	15.85	16.58	17.18
DynGLR-G-12312*	15.22	15.46	15.66	15.85	16.55	17.17
DynGLR-G-12312s*	**15.22**	**15.45**	**15.65**	**15.83**	**16.49**	**16.85**

TABLE V: Area Under the Curve For ***Magic*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	0.816	0.809	0.804	0.798	0.794	0.789
CNN	0.834	0.830	0.829	0.823	0.818	0.811
DynGraph-CNN	0.823	0.813	0.807	0.789	0.759	0.726
DML-KNN	0.845	0.843	0.840	0.839	0.830	0.816
DML-KNN-s	0.845	0.843	0.840	0.839	0.833	0.828
LN-Robust-SVM-RBF*	0.814	0.812	0.812	0.809	0.806	0.802
Graph-Hybird*	0.752	0.741	0.728	0.712	0.692	0.668
CNN-Savage*	0.835	0.831	0.829	0.825	0.818	0.811
CNN-BootStrapHard*	0.835	0.830	0.828	0.824	0.818	0.813
CNN-D2L*	0.835	0.831	0.827	0.824	0.817	0.812
DynGLR-G-2*	0.844	0.843	0.840	0.839	0.831	0.819
DynGLR-G-12*	0.846	0.843	0.841	0.840	0.832	0.825
DynGLR-G-12s*	0.846	0.843	0.841	0.840	0.833	0.829
DynGLR-G-1232*	0.846	0.843	0.841	0.840	0.833	0.827
DynGLR-G-12312*	0.846	0.843	0.841	0.840	0.833	0.827
DynGLR-G-12312s*	**0.846**	**0.843**	**0.841**	**0.840**	**0.833**	**0.830**

C. Summary of findings

The DynGLR-G networks at the bottom of performance Tables. II, IV,VI,VIII show the outcomes of the ablation study. Specifically:

- DynGLR-G-12 consistently outperforms DynGLR-G-2, showing the positive influence of edge-weighting.
- The improvement due to graph update can be observed between DynGLR-G-12 and DynGLR-G-1232
- By comparing DynGLR-G-1232 and DynGLR-G12312, we observe small gains, except for low noise in Spambase dataset, due to iterative design, incorporating edge weighting.
- By comparing DynGLR-G-2 and DML-KNN, we can observe performance improvements due to replacing KNN-based classification with GLR; larger gains can be observed as noise level increases.
- Semi-supervised classifiers DML-KNN and all DynGLRs with sampling (DynGLR-G-12s and DynGLR-G-12312s) benefit from rank-sampling, which also reduces the scale of training set without sacrificing the performance.

Furthermore, our findings are that the importance of the following algorithmic steps, in order of highest to least importance, to the performance, can be summarized as: (I)

TABLE VI: Classification Error Rate (%) For ***Spambase*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	8.09	8.50	8.98	9.75	10.68	11.49
CNN	7.69	8.27	8.89	9.85	10.8	12.47
DynGraph-CNN	8.33	9.01	10.45	12.68	16.78	22.34
DML-KNN	7.84	8.41	8.49	8.98	9.83	11.02
DML-KNN-s	7.81	7.35	8.42	8.86	9.74	10.34
LN-Robust-SVM-RBF*	7.84	8.38	8.89	9.68	10.56	11.32
Graph-Hybrid*	18.34	19.19	20.37	21.85	24.17	26.68
CNN-Savage*	8.04	8.36	8.90	9.80	10.42	12.13
CNN-BootStrapHard*	7.69	8.30	9.24	9.68	10.05	12.01
CNN-D2L*	7.73	8.46	9.05	9.87	10.96	12.17
DynGLR-G-2*	7.84	8.37	8.42	8.95	9.85	10.83
DynGLR-G-12*	7.73	8.13	8.37	8.78	9.44	9.82
DynGLR-G-12s*	7.72	8.11	8.35	8.75	9.35	9.63
DynGLR-G-1232*	7.65	8.05	8.22	8.67	9.15	9.56
DynGLR-G-12312*	7.55	7.99	8.21	8.64	9.01	9.18
DynGLR-G-12312s*	**7.55**	**7.94**	**8.18**	**8.61**	**8.96**	**9.13**

TABLE VII: Area Under the Curve For ***Spambase*** Dataset

% label noise	0	5	10	15	20	25
SVM-RBF	0.919	0.915	0.910	0.903	0.893	0.885
CNN	0.923	0.917	0.911	0.902	0.892	0.875
DynGraph-CNN	0.917	0.910	0.895	0.873	0.832	0.777
DML-KNN	0.921	0.916	0.915	0.910	0.902	0.890
DML-KNN-s	0.922	0.926	0.915	0.911	0.903	0.897
LN-Robust-SVM-RBF*	0.922	0.916	0.911	0.903	0.894	0.887
Graph-Hybird*	0.817	0.808	0.796	0.781	0.758	0.733
CNN-Savage*	0.919	0.916	0.911	0.902	0.896	0.879
CNN-BootStrapHard*	0.923	0.917	0.907	0.903	0.900	0.880
CNN-D2L*	0.922	0.915	0.909	0.901	0.891	0.878
DynGLR-G-2*	0.922	0.916	0.915	0.911	0.902	0.893
DynGLR-G-12*	0.923	0.917	0.916	0.912	0.906	0.902
DynGLR-G-12s*	0.924	0.918	0.917	0.912	0.907	0.903
DynGLR-G-1232*	0.925	0.919	0.918	0.913	0.908	0.905
DynGLR-G-12312*	0.925	0.919	0.917	0.914	0.908	0.906
DynGLR-G-12312*	**0.925**	**0.920**	**0.917**	**0.915**	**0.909**	**0.907**

TABLE VIII: Classification Error Rate (%) For ***CIFAR-10*** Dataset

% label noise	0	5	10	15	20	25
SVM-Linear	2.12	5.46	7.66	9.95	12.96	16.75
CNN	2.67	2.9	3.16	3.66	4.36	6.26
DML-KNN	2.72	2.32	3.34	3.89	4.46	5.22
DML-KNN-s	2.69	2.88	3.12	3.54	3.96	4.51
LN-Robust-SVM-Linear*	2.93	3.25	3.5	3.78	3.97	4.46
Graph-Hybrid*	8.65	9.74	11.58	14.05	17.21	20.45
CNN-BootStrapHard*	2.29	2.65	3.11	3.78	4.34	4.61
CNN-D2L*	2.82	3.02	3.13	3.28	3.97	4.32
DynGLR-G-2*	6.64	6.68	6.74	6.76	7.00	7.08
DynGLR-G-12*	2.04	2.16	2.32	2.34	2.54	2.64
DynGLR-G-12s*	2.04	2.15	2.30	2.32	2.51	2.61
DynGLR-G-1232*	1.40	1.41	1.67	1.83	1.99	2.33
DynGLR-G-12312*	1.31	1.32	1.59	1.76	1.84	2.20
DynGLR-G-12312s*	**1.31**	**1.32**	**1.58**	**1.73**	**1.81**	**2.16**

alternate graph update Eq.(4) - disconnect $\mathcal{Q}$-edges and connect/reconnect $\mathcal{P}$-edges based on the restored labels after each GLR iteration to refine the graph structure, (II) edge convolution operation - performing feature and denoised label aggregation on neighboring nodes provides richer and smoother inputs and results in a spatially sparse graph, (III) edge attention function (Eq. 7) - to better reflect node-to-node correlation, regularizing CNN training by weighting the edge loss based on classifier signal changes before and after GLR.

VI. Conclusions

We introduced an end-to-end iterative graph-based deep learning architecture design to tackle the overfitting problem caused by the effects of noisy training labels. We first proposed a CNN-based graph generator G-Net to build an initial graph. Relying on the proposed graph-based regularized loss functions, we then proposed a graph-based classifier W-Net to perform online label denoising of the training samples that potentially have noisy labels. Based on the denoised training labels, we update the underlying graph structure by learning the proposed graph update H-Net. Finally, we learn a refined graph-based classifier W-Net to perform classification using the updated underlying graph structure. The validation on three different binary classification datasets demonstrate that our proposed architecture outperforms the state-of-the-art classification methods when partial training labels are incorrect. Furthermore, the rank-sampling method is proven to be another enhancement.

A possible direction for future work is to extend the concept to multi-class classification, which can be accomplished, for example by (1) using a one-vs-all scheme, (2) after W-Net is trained, using the output feature maps from $\text{CNN}_{\mathcal{C}^r}$ to train a standard multi-class classifier.

VII. Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734331. The University of Strathclyde gratefully acknowledges the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] M. Ye, V. Stankovic, L. Stankovic, and G. Cheung, "Deep graph based learning for binary classification," in IEEE ICASSP, 2019.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in NIPS, 2012, pp. 1097–1105.

[3] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," arXiv:1406.2080, 2014.

[4] G. Patrini, A. Rozza, A. K. Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in IEEE CVPR, 2017, pp. 2233–2241.

[5] X. Ma, Y. Wang, M. E. Houle, S. Zhou, S. M. Erfani, S.-T. Xia, S. Wijewickrema, and J. Bailey, "Dimensionality-driven learning with noisy labels," in ICML, 2018, pp. 3361–3370.

[6] P. Lemberger, "On generalization and regularization in deep learning," arXiv:1704.01312, 2017.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv:1207.0580, 2012.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in ICML, 2015, pp. 448–456.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE CVPR, 2016, pp. 770–778.

[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in IEEE CVPR, vol. 1, 2017, p. 3.

[11] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," in IEEE CVPR, 2015, pp. 2691–2699.

[12] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," arXiv:1802.05300, 2018.

[13] H. Masnadi-Shirazi and N. Vasconcelos, "On the design of loss functions for classification: theory, robustness to outliers, and savageboost," in NIPS, 2009, pp. 1049–1056.

[14] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," arXiv:1412.6596, 2014.

[15] J. Goldberger and E. Ben-Reuven, "Training deep neural-networks using a noise adaptation layer," 2016.

[16] A. Sandryhaila and J. M. F. Moura, "Classification via regularization on graphs," in IEEE GlobalSIP, 12 2013, pp. 495–498.

[17] C. Gong, T. Liu, D. Tao, K. Fu, E. Tu, and J. Yang, "Deformed graph laplacian for semisupervised learning," IEEE TNNLS, vol. 26, no. 10, pp. 2261–2274, 10 2015.

[18] G. Cheung, W. Su, Y. Mao, and C. Lin, "Robust semisupervised graph classifier learning with negative edge weights," IEEE Transactions on Signal and Information Processing over Networks, vol. 4, no. 4, pp. 712–726, 11 2018.

[19] C. Yang, G. Cheung, and V. Stankovic, "Alternating binary classifier and graph learning from partial labels," in APSIPA, 11 2018.

[20] J. Zeng, J. Pang, W. Sun, and G. Cheung, "Deep graph laplacian regularization for robust denoising of real images," arXiv:1807.11637, 2018.

[21] B. Jiang and D. Lin, "Graph laplacian regularized graph convolutional networks for semi-supervised learning," arXiv:1809.09839, 2018.

[22] C. Cortes and V. Vapnik, "Support-vector networks," Machine learning, vol. 20, no. 3, pp. 273–297, 1995.

[23] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," arXiv:1801.07829, 2018.

[24] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in International Workshop on Similarity-Based Pattern Recognition. Springer, 2015, pp. 84–92.

[25] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in ACML, vol. 20, South Garden Hotels and Resorts, Taoyuan, Taiwain, 11 2011, pp. 97–112.

[26] M. Speriosu, N. Sudan, S. Upadhyay, and J. Baldridge, "Twitter polarity classification with label propagation over lexical links and the follower graph," in Proceedings of the First workshop on Unsupervised Learning in NLP, 2011, pp. 53–63.

[27] J. Tang, R. Hong, S. Yan, T.-S. Chua, G.-J. Qi, and R. Jain, "Image annotation by knn-sparse graph-based label propagation over noisily tagged web images," ACM TIST, vol. 2, no. 2, p. 14, 2011.

[28] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari, "Learning object class detectors from weakly annotated video," in IEEE CVPR, 2012, pp. 3282–3289.

[29] I. Misra, A. Shrivastava, and M. Hebert, "Watch and learn: Semi-supervised learning for object detectors from video," in IEEE CVPR, 2015, pp. 3593–3602.

[30] A. Kuznetsova, S. Ju Hwang, B. Rosenhahn, and L. Sigal, "Expanding object detector's horizon: incremental learning framework for object detection in videos," in IEEE CVPR, 2015, pp. 28–36.

[31] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," arXiv:1803.09050, 2018.

[32] I. Jindal, M. Nokleby, and X. Chen, "Learning deep networks from noisy labels with dropout regularization," in IEEE ICDM, 2016, pp. 967–972.

[33] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," arXiv:1705.10694, 2017.

[34] V. Mnih and G. E. Hinton, "Learning to label aerial images from noisy data," in ICML, 2012, pp. 567–574.

[35] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," in IEEE CVPR, 2015, pp. 2691–2699.

[36] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia, "Iterative learning with open-set noisy labels," arXiv:1804.00092, 2018.

[37] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," arXiv:1312.6203, 2013.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," arXiv:1609.02907, 2016.

[39] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in NIPS, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 3844–3852.

[40] X. Zhang, C. Xu, and D. Tao, "Graph edge convolutional neural networks for skeleton based action recognition," arXiv:1805.06184, 2018.

[41] J. Mojoo, K. Kurosawa, and T. Kurita, "Deep cnn with graph laplacian regularization for multi-label image annotation," in ICIAR. Springer, 2017, pp. 19–26.

[42] V. N. Ekambaram, G. Fanti, B. Ayazifar, and K. Ramchandran, "Wavelet-regularized graph semi-supervised learning," in IEEE GlobalSIP, 2013, pp. 423–426.

[43] A. Gadde, A. Anis, and A. Ortega, "Active semi-supervised learning using sampling theory for graph signals," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 492–501.

[44] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering," in Proceedings of the 17th International Conference on Neural Information Processing Systems, ser. NIPS. MIT Press, 2004, pp. 1601–1608.

[45] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under laplacian and structural constraints," IEEE JSTSP, vol. 11, no. 6, pp. 825–841, 2017.

[46] M. El Gheche, G. Chierchia, and P. Frossard, "Orthonet: Multilayer network data clustering," IEEE Transactions on Signal and Information Processing over Networks, vol. 6, pp. 13–23, 2020.

[47] M. Bontonou, C. Lassance, G. B. Hacene, V. Gripon, J. Tang, and A. Ortega, "Introducing graph smoothness loss for training deep learning architectures," arXiv preprint arXiv:1905.00301, 2019.

[48] X. Zhang, C. Xu, X. Tian, and D. Tao, "Graph edge convolutional neural networks for skeleton-based action recognition," IEEE TNNLS, pp. 1–14, 9 2019.

[49] M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv:1312.4400, 2013.

[50] J. Alcala-Fdez, A. Fernández, J. Luengo, J. Derrac, S. Garc'ia, L. Sanchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," Journal of Multiple-Valued Logic and Soft Computing, vol. 17, pp. 255–287, 01 2010.

[51] "The cifar-10 dataset," https://www.cs.toronto.edu/~kriz/cifar.html, Aug 2020.

[52] Y. Wang, Z. Pan, and Y. Pan, "A training data set cleaning method by classification ability ranking for the k-nearest neighbor classifier." IEEE TNNLS, 2019.

[53] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in NIPS, 2001, pp. 402–408.

[54] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in IEEE CVPR, 2019, pp. 2661–2671.

[55] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," IEEE signal processing magazine, vol. 30, no. 3, pp. 83–98, 2013.