Instance GNN: A Learning Framework for Joint Symbol Segmentation and Recognition in Online Handwritten Diagrams

Xiao-Long Yun, Yan-Ming Zhang, Fei Yin, and Cheng-Lin Liu, *Fellow, IEEE*

***Abstract*—Online handwritten diagram recognition (OHDR) has attracted considerable attention for its potential applications in many areas, but it is a challenging task due to the complex 2D structure, writing style variation, and lack of annotated data. Existing OHDR methods often have limitations in modeling and learning complex contextual relationships. To overcome these challenges, we propose an OHDR method based on graph neural networks (GNNs) in this paper. In particular, we formulate symbol segmentation and symbol recognition as node clustering and node classification problems on stroke graphs and solve the problems jointly under a unified learning framework with a GNN model. This GNN model is denoted as Instance GNN since it gives the symbol instance label as well as the semantic label. Extensive experiments on two flowchart datasets and a finite automata dataset show that our method consistently outperforms previous methods with large margins and achieves state-of-the-art performance. In addition, we release a large-scale annotated online handwritten flowchart dataset, CASIA-OHFC, and provide initial experimental results as a baseline.**

***Index Terms*—Online handwritten diagram recognition, symbol segmentation, symbol recognition, freehand sketch analysis, graph neural networks.**

I. Introduction

HANDWRITING is one of the most natural and efficient ways for humans to record information. Due to the widespread use of smartphones, tablets and electrical white-boards, writing with a stylus on touch screens is becoming increasingly popular, resulting in the generation of large numbers of online handwritten documents or digital ink documents. Unlike scanned document images, online handwritten documents are composed of ordered stroke trajectory points and characterized by high sparsity and rich dynamic information.

Online handwritten diagrams are common elements in online handwritten documents and have applications in many areas such as human-machine interfaces, education, office automation and conference systems. Therefore, automatic recognition and understanding of diagrams have long been a focus of considerable attention. Despite progress, the recognition of handwritten diagrams such as flowcharts, circuits and music scores (which usually consist of graphic symbols and texts) remains challenging because of the complex 2D structures, large variations in writing style and the scarcity of annotated data.

OHDR involves two interdependent tasks: segmenting an input diagram into semantically meaningful components (symbol segmentation) and recognizing the categories of the components (symbol recognition). Accordingly, existing OHDR methods can be roughly divided into two categories: bottom-up methods and top-down methods. Bottom-up methods [1]–[9] sequentially perform a symbol segmentation step and a symbol recognition step. By contrast, top-down approaches [10]–[14] unify the two steps using syntactic grammars or probabilistic graphical models (PGM) and perform segmentation and recognition simultaneously. However, these methods are mostly based on traditional machine learning techniques and have limitations in learning high-level representations and modeling complex structures, which are important for high-accuracy diagram recognition. In terms of methodology, OHDR is also closely related to online handwritten mathematical expression recognition [15] and free-hand sketch analysis [16]–[18]. We review these methods in more detail in the next section.

OHDR is also difficult due to the lack of large-scale annotated diagram datasets. Existing online handwritten diagram datasets with annotations [6], [9], [11] are small in size and simple in layout. This prevents researchers from exploring the power of deep neural networks, which usually require large-scale labeled datasets. Our experimental results verify this limitation.

In this paper, we propose a top-down OHDR method based on graph neural networks (GNNs) [19], [20] to overcome the aforementioned problems. In particular, we represent each diagram as an undirected graph in which nodes denote strokes and edges describe the temporal/spatial relationships between strokes. Based on the stroke graph representation, we formulate diagram segmentation and recognition as node clustering and node classification problems, respectively, and design a novel GNN-based model and three learning tasks to solve these problems jointly. Compared with previous methods, our method is more powerful and flexible in learning the stroke representation and exploiting the contextual relationship. In addition, unlike PGM-based methods, the learning and inference of our method are rather simple and efficient and therefore suitable for large-scale applications.

Xiao-Long Yun is with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China, and the National Laboratory of Pattern Recognition, Institute of Automation of Chinese Academy of Sciences, Beijing 100190, China; Yan-Ming Zhang and Fei Yin are with the National Laboratory of Pattern Recognition, Institute of Automation of Chinese Academy of Sciences, Beijing 100190, China; Cheng-Lin Liu is with the National Laboratory of Pattern Recognition, Institute of Automation of Chinese Academy of Sciences, Beijing 100190, China, the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China, and the Center for Excellence of Brain Science and Intelligence Technology, Beijing 100190, China. (e-mail: {xiaolong.yun, ymzhang, fyin, liucl}@nlpr.ia.ac.cn)

We highlight the main contributions of this work as follows.

1) We propose a novel GNN-based method for online handwritten diagram recognition. Experiments on popular benchmark datasets show that the proposed method dramatically outperforms existing methods and achieves state-of-the-art results.
2) We release a large-scale online handwritten flowchart dataset CASIA-OHFC with both semantic- and instance-level annotations to foster research and applications in automatic handwritten diagram recognition. The dataset is available at http://www.nlpr.ia.ac.cn/databases/CASIA-OHFC/.

In the remainder of this paper, we first review related works in Sect. II. Sect. III describes the details of the proposed method. Sect. IV introduces the dataset CASIA-OHFC. Sect. V presents experimental results, and Sect. VI provides concluding remarks.

II. Related Works

A. Diagram Recognition

Handwritten diagram recognition methods can be categorized based on the data modality: online methods for stroke trajectories and offline methods for raster images. Online recognition has received more attention than offline recognition.

Online handwritten diagram recognition (OHDR) involves two closely related tasks: symbol segmentation and symbol recognition. OHDR methods can be categorized into bottom-up methods and top-down methods. Bottom-up methods [1]–[9] sequentially perform symbol segmentation and recognition. They first generate a large number of symbol hypotheses (SHs) using the over-segmentation technique and then apply trained classifiers to recognize the SHs and reject falsely segmented SHs. Bottom-up methods vary in how to generate and recognize SHs. Feng et al. [3] used 2D dynamic programming (DP) to generate SHs. Carton et al. [5] proposed a method based on the Description and Modification of Segmentation (DMOS) method [21], which uses a grammatical language to describe circular symbols and quadrilateral symbols. Costagliola et al. [7] introduced visual languages with local context to check the validity of graphical symbols. In [6], [8], [9], strokes were first classified as text or non-text; then, SVM was used to cluster non-text strokes and classify uniform symbols, and finally arrows were detected. For structure analysis, Bresler et al. modeled a whole flowchart excluding texts as a max-sum problem and applied integer linear programming to solve it [22].

Top-down methods unify symbol segmentation and recognition in an optimization framework and solve two tasks jointly. Awal et al. [11] used a time-delayed neural network and DP to jointly solve segmentation and recognition. Lemaitre et al. [10] integrated structural and syntactic priors of a flowchart with EPF (Enhanced Position Formalism) language [21] and then used the DMOS method to segment and recognize the flowchart in one step. Wang et al. [12], [13] proposed a general model, max-margin MRF, that combines Markov random fields (MRF) and structural SVM to perform stroke segmentation and recognition simultaneously. By exploiting the temporal and spatial relationships between strokes, their model greatly improved the stroke labeling accuracy. Recently, Julca-Aguilar et al. [14] proposed to represent a graphic as a labeled graph with graph grammars [23] and convert the symbol segmentation and recognition problem into a graph parsing problem.

For offline handwritten diagram recognition, Wu et al. [24] proposed an efficient shapeness estimation method for fast SH selection and then applied symbol classifiers for symbol recognition. Bresler et al. [25] used an online recognition method for offline recognition via stroke reconstruction for flowchart images. Other works applied deep learning-based general object detection frameworks, such as the Faster R-CNN model [26], for handwritten diagram recognition. Julca-Aguilar et al. [27] directly applied the Faster R-CNN to detect online handwritten symbols by converting the online data into offline images. Despite the overall high performance of flowchart symbol detection, the detection of arrows was not satisfactory because of their very high variation in shape. Schäfer et al. [28] applied keypoint estimation methods in human pose estimation for arrow detection. They proposed Arrow R-CNN for offline handwritten flowchart recognition and extended the method for online flowchart recognition. Their method achieved state-of-the-art performance in both offline and online diagram recognition on a public flowchart dataset. However, similar to previous works [3], [6]–[9], different detectors were required for different components of the diagram.

Similar to [12], [13], our proposed method performs symbol segmentation and recognition simultaneously based on graph representation. However, we use the GNN model, which is more powerful and flexible in modeling the spatial and temporal relationships between strokes than traditional PGM. Moreover, similar to bottom-up methods [1]–[9], our method adopts the over-segmentation technique in postprocessing to further improve the performance.

B. Mathematical Expression Recognition

Similar to diagrams, mathematical expressions also involve rich 2D structures. Handwritten mathematical expression recognition (HMER) consists of three major tasks [29]: symbol segmentation, symbol recognition, and structural analysis. Early methods of HMER used mathematical expression grammars to guide the segmentation and analysis of symbols. A variety of human-designed grammars have been proposed for HMER, such as stochastic context-free grammar [30], relational grammar [31] and graph grammars [14], [23]. However, the grammars could not be automatically learned from data, and the design of high-efficiency grammars is a difficult task.

Recently, inspired by works in machine translation, researchers have proposed new HMER methods under the deep learning-based encoder-decoder framework. These works [15], [32]–[36] treat HMER as image-to-sequence or sequence-to-sequence problems and use a convolutional neural network (CNN) or recurrent neural network (RNN) encoder to embed the input formula and an attention-based sequential decoder to generate the LaTeX markup. Compared with grammar-based methods, these methods achieve significant advances in recognition accuracy. However, they cannot generate symbol segmentation nor build the correspondence between the predicted output labels and the input regions. Therefore, the resulting outputs lack interpretability.

C. Freehand Sketch Analysis

Another closely related problem is freehand sketch analysis, which mainly focuses on sketch recognition, retrieval, perceptual grouping, semantic segmentation and generation tasks. Interested readers can refer to recent survey papers [37], [38]. Here we briefly review some works on sketch semantic segmentation, which aims to predict the semantic label of each stroke (or trajectory point). Traditional methods first extract conventional handcrafted features from sketches and then apply mixed integer programming or conditional random fields to make the prediction [16], [17]. Deep learning methods can be roughly grouped into four categories: image-based methods [18], [39], sequence-based methods [40], graph-based methods [41] and multimodal methods [42]. Image-based methods treat the task as an image semantic segmentation problem and adopt a CNN to solve the problem. Sequence-based methods treat the task as a sequence prediction problem and solve it with an RNN. Graph-based methods represent each sketch as a graph in which the nodes denote the 2D point set and the edges denote the graphical relationships between points; then, a GNN is applied for stroke feature fusion and node classification. Multimodal methods exploit multiple complementary modes of sketch data, such as static images and dynamic point set, for the segmentation task.

Although some tasks of sketch analysis (such as grouping and semantic segmentation) and OHDR are similar, diagrams have more complex structures and typically contain much more strokes and symbols than freehand sketches. Therefore, existing sketch analysis methods cannot be directly extended to OHDR[1].

D. Graph Neural Networks

In recent years, GNNs have received enormous attention and become one of the most popular research focuses in the field of artificial intelligence. Due to their ability to capture the dependency between objects and operate in non-Euclidean domains [44], GNNs have obtained great success in many tasks, such as relational reasoning [45] and text classification [46]. Most of the pioneering GNN models, including GCN [19], GAT [20] and GraphSAGE [47], only exploit the node features in graph learning, and the edge features of the graph are inadequately utilized. Here we briefly review some GNN models that incorporate node features and edge features in graph learning. Xu et al. [48] proposed an end-to-end model architecture for scene graph generation from images. Their model iteratively updates the node and edge features in the graph through a message pooling function to infer the object categories, bounding boxes and relationships between objects. Hamrick et al. [49] constructed a deep reinforcement learning agent via a graph network to solve the gluing task [49], which demonstrated the importance of relational knowledge for performing interactive structured reasoning problems. Gong et al. [50] explored edge features with doubly stochastic normalization in current GNN frameworks and obtained better performance compared with state-of-the-art methods. Recently, Ye et al. [51] proposed a new GAT framework for stroke classification, which demonstrated the great potential of GNN for online handwritten document recognition.

The core structure of our InstGNN model is based on GAT [20] and its variant [51]. Different from previous works, InstGNN is equipped with multiple branches to simultaneously perform node classification, edge classification and node embedding.

[1]The number of points for each sketch is set to 256 at most in [41], and it is difficult to train the model beyond 300 data points in [43]. A diagram usually contains hundreds of strokes with thousands of points, even if resampled.

III. Proposed Method

TABLE I
Summary of Important Notations.

Notation	Description
$\mathcal{D} = \{D^i \mid i = 1, ..., N\}$	Diagram dataset with N samples
X^i_s, Y^i_s, I^i_s	Raw feature, semantic label and instance label of stroke s in diagram i
M_i	Number of strokes in diagram i
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$
$\mathcal{N}(v)$	Neighbors of node v, including v
$\mathcal{N}_{\mathrm{T}}(v)$, $\mathcal{N}_{\mathrm{S}}(v)$	Temporal and spatial neighbors of node v
K_{T}, K_{S}	Number of temporal and spatial nearest neighbors
$Y^i_e \in \{-1, 1\}$	Ground truth label of edge e in diagram i
$\overrightarrow{h}_i$	Feature vector of node $v_i \in \mathcal{V}$
$\overrightarrow{f}_{ij}$	Feature vector of edge $e_{ij} = \{v_i, v_j\} \in \mathcal{E}$
$\mathbf{W}$	Learnable matrix
$\overrightarrow{a}$, $\overrightarrow{b}$	Learnable vector
$\sigma(\cdot)$	Leaky ReLU activation function
$\mathbb{I}(\cdot)$	$\mathbb{I}(\text{True}) = 1$; $\mathbb{I}(\text{False}) = 0$
$\odot$	Element-wise matrix multiplication
$\oplus$	Concatenation operator
$\Vert \cdot \Vert$	ℓ_2-norm
$[x]_+ = \max(0, x)$	Hinge function
K	Number of attention heads in node learning layer

A. Problem Definition

We are given N labeled online handwritten diagrams $\mathcal{D} = \{D^i \mid i = 1, ..., N\}$. Each diagram D^i contains a sequence of strokes $\{X^i_s \mid s = 1, ..., M_i\}$, where M_i is the number of strokes in diagram D^i. A stroke X^i_s consists of ordered trajectory points between a 'pen-down' and a 'pen-up', and each point records the (x,y)-coordinates, time, pressure and state of the pen movement (such as pen-down or pen-up). In addition, stroke X^i_s also has a semantic-level label Y^i_s, which indicates its stroke class of diagram symbol (such as '*text*', '*process*', '*decision*' and '*arrow*'), and an instance-level label I^i_s, which indicates the specific symbol it belongs to. Thus, diagram D^i can be represented as $\{(X^i_s, Y^i_s, I^i_s) \mid s = 1, ..., M_i\}$.

Our target is to learn a model from $\mathcal{D}$ that can group the strokes of test diagrams into symbols and also predict the labels of the symbols with high accuracy.

Unless specified otherwise, the key mathematical notations used throughout this paper follow the definitions in Tab. I.

B. System Overview

Essentially, our method represents each diagram as an attributed graph in which nodes (vertexes) represent strokes and edges represent the temporal/spatial relationship between strokes. Then, we formulate stroke classification and symbol segmentation as node classification and node clustering problems, respectively, and design a novel GNN-based model, named Instance GNN (InstGNN for short), to perform the two tasks jointly. Finally, we obtain the symbol segmentation and recognition results through postprocessing. An overview of the proposed framework is shown in Fig. 1. It is composed of three key modules: the diagram graph building, the InstGNN model and the postprocessing procedure, which will be introduced separately in the following subsections.

C. Graph Building

Each handwritten diagram is represented as a space-time relationship graph (STRG), where each stroke s_i is represented as a vertex v_i and the relationship in space and time between strokes s_i and s_j is encoded as edge $e_{ij} = \{v_i, v_j\}$. The resulting graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}$ is the vertex set and $\mathcal{E} = \{e_{ij}\}$ is the edge set. The nodes (vertexes) and edges are further described by real-valued feature vectors. We explain the edge construction and feature extraction methods below.

1) Edge Construction

We consider two different edge construction approaches to build the edge set.

K nearest temporal edges. For each stroke s, its forward K_T nearest strokes and backward K_T nearest strokes in the drawing sequence are regarded as the temporal neighbors $\mathcal{N}_\text{T}(s)$ of s. Therefore, $\mathcal{N}_\text{T}(s) = \{s-K_\text{T}, ..., s-1, s+1, ..., s+K_\text{T}\}$. The edge set $\{e_{sj} | j \in \mathcal{N}_\text{T}(s)\}$ is added to $\mathcal{E}$ for each stroke in the diagram.

K nearest spatial edges. For each stroke s, its K_S nearest strokes in Euclidean space are regarded as the spatial neighbors $\mathcal{N}_\text{S}(s)$ of s. The distance between two strokes is defined as the minimal Euclidean distance among all pairwise points on the strokes. The edge set $\{e_{sj} | j \in \mathcal{N}_S(s)\}$ is added to $\mathcal{E}$ for each stroke in the diagram.

K_T and K_S are two integer hyperparameters that control the number of edges in the graph, which need to be tuned on the validation set. Fig. 2 shows an example of a diagram rendered from original data and its corresponding STRG.

2) Feature Extraction

We extract hand-crafted features [52]–[54] for nodes and edges as their initial representations. For each node, 13 geometric features, 8 contextual features and 6 positional features are extracted from the corresponding stroke. For each edge, 19 features are extracted to model the relationship in space and time between the two linked strokes. The details of these node features and edge features are depicted in Tab. XI and XII in Appendix A. To extract the contextual features for nodes, two hyperparameters—the spatial neighbor threshold and the temporal neighbor threshold—are needed to decide whether two strokes are spatial neighbors or temporal neighbors or not. In this paper, we simply set them to be equal to the aforementioned K_S and K_T, respectively.

Following previous works [52], [53], the extracted features are preprocessed by power transformation with the coefficient 0.5 and normalization with mean $\vec{\mu}$ and standard deviation $\vec{\sigma}$. In this way, the original feature $\vec{x}$ becomes $\vec{x}''$:

$$\vec{x}' = \text{sign}(\vec{x})\sqrt{|\vec{x}|}, \tag{1}$$

$$\vec{x}'' = (\vec{x}' - \vec{\mu}) / \vec{\sigma}, \tag{2}$$

where sign$(\cdot)$ is the sign function, $\vec{\mu}$ and $\vec{\sigma}$ are estimated from the training data. Here, all operators run in element-wise style.

D. Instance Graph Neural Network

The instance graph neural network (InstGNN) is constructed by stacking multiple node learning layers and edge learning layers. Specifically, InstGNN is composed of one shared backbone and three interactive branches corresponding to three tasks defined in Sect. III-E: node classification, edge classification and node representation learning. The backbone, node classification branch, and node representation branch are composed of node learning layers, while the edge classification branch consists of several edge learning layers. The architecture of InstGNN is shown in Fig. 1.

The input to each node/edge learning layer is a set of node features $\mathbf{H} = \left\{\vec{h}_i \in \mathbb{R}^P | v_i \in \mathcal{V}\right\}$ and a set of edge features $\mathbf{F} = \left\{\vec{f}_{ij} \in \mathbb{R}^Q | e_{ij} \in \mathcal{E}\right\}$, where P, Q are the dimensionalities of the node features and edge features, respectively. The node learning layer outputs a new set of node features $\mathbf{H}' = \left\{\vec{h}'_i \in \mathbb{R}^{P'} | v_i \in \mathcal{V}\right\}$, where P' is the dimensionality of the output node features. Likewise, the edge learning layer outputs a new set of edge features $\mathbf{F}' = \left\{\vec{f}'_{ij} \in \mathbb{R}^{Q'} | e_{ij} \in \mathcal{E}\right\}$, where Q' is the dimensionality of the output edge features. We give detailed descriptions of the learning layers in the following.

1) Node Learning Layer

Basically, the node learning layer updates the representation of each node by aggregating information from its neighbors with the attention mechanism. Its structure is a variant of GAT [20], [55]. Specifically, to compute attention coefficients, we first utilize self-attention to measure the similarity between node v_i and node v_j as follows:

$$c_{ij} = \sigma\left(\vec{a}_{h_1}^T \left(\mathbf{W}_{h_1} \vec{h}_i \oplus \mathbf{W}_{h_1} \vec{h}_j\right)\right), \tag{3}$$

where $\mathbf{W}_{h_1} \in \mathbb{R}^{P' \times P}$ and $\vec{a}_{h_1} \in \mathbb{R}^{2P'}$ are learnable parameters. $\sigma(\cdot)$ is the activation function, and $\oplus$ is the concatenation operator.

In addition to classic self-attention, we also incorporate edge features to measure the importance of neighbor v_j by the following equation:

$$c'_{ij} = \sigma\left(\vec{a}_{h_2}^T \sigma\left(\mathbf{W}_{h_2} \vec{f}_{ij} + \vec{b}_h\right)\right), \tag{4}$$

where $\vec{a}_{h_2}, \vec{b}_h \in \mathbb{R}^{P'}, \mathbf{W}_{h_2} \in \mathbb{R}^{P' \times Q}$ are all learnable parameters.

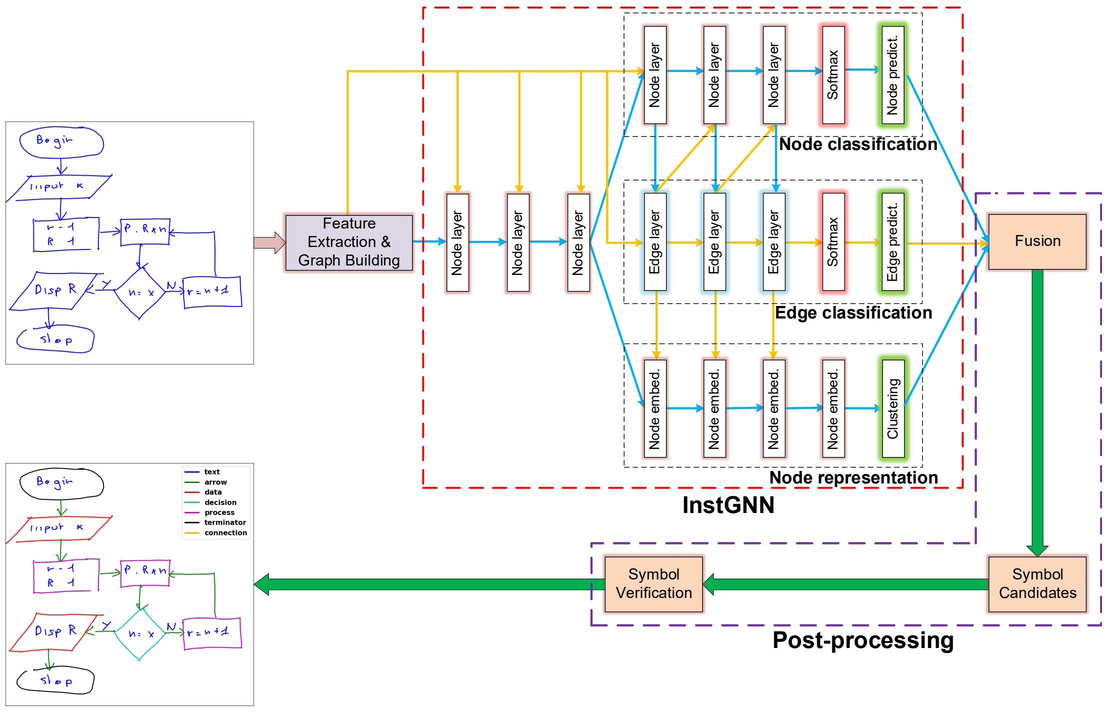


Fig. 1. Overview of the proposed framework. Yellow lines indicate the edge information flow and blue lines indicate the node information flow.

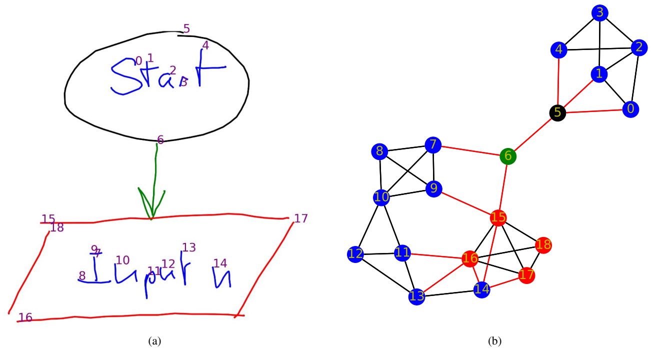


Fig. 2. An example of a handwritten diagram and its corresponding space-time relationship graph (STRG). The numbers in the figure indicate the temporal order of strokes. Different colors denote different stroke and edge classes. (a) The original handwriting diagram and (b) the corresponding STRG.

Since the coefficients above are not comparable across different nodes, we normalize them by *softmax*:

$$\alpha_{ij} = \operatorname{softmax}_j\left(c_{ij} + c'_{ij}\right) = \frac{\exp\left(c_{ij} + c'_{ij}\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(c_{ik} + c'_{ik}\right)}, \quad (5)$$

where $\mathcal{N}(i)$ is the set of neighbors of node v_i .

The output features are computed by aggregating node features of neighbors with attention coefficients:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}_{h_1}\vec{h}_j\right), \quad (6)$$

Following Veličković et al. [20], we also employ K independent attention heads in our model:

$$\vec{h}'_i = \oplus_{k=1}^{K}\sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^k\mathbf{W}_{h_1}^k\vec{h}_j\right), \quad (7)$$

where α_{ij}^k is the normalized attention coefficients calculated by the k-th attention head.

In the last node learning layer, we perform average pooling instead of concatenation and remove the nonlinear transformation:

$$\vec{h}'_i = \frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^k\mathbf{W}_{h_1}^k\vec{h}_j. \quad (8)$$

2) *Edge Learning Layer*

Intuitively, if the representation of the edge between different symbols is clearly distinguishable from the edge within a symbol, the stroke graph can be easily segmented into symbols by performing edge classification. Motivated by this, we design an independent edge learning layer to explicitly learn edge representations. The edge learning is first performed as follows:

$$\vec{m}_{ij} = \sigma\left(\mathbf{W}_{f_1}\vec{f}_{ij} + \vec{b}_{f_1}\right), \quad (9)$$

where $\mathbf{W}_{f_1} \in \mathbb{R}^{U\times Q}$ and $\vec{b}_{f_1} \in \mathbb{R}^U$ are learnable parameters, $\vec{m}_{ij} \in \mathbb{R}^U$ is the intermediate representation of edge e_{ij} and U is the dimensionality of $\vec{m}_{ij}$.

To exploit more contextual information for edge representation learning, we consider combining node features. Particularly, we measure the similarity between node v_i and node v_j as follows:

$$\vec{r}_{ij} = \sigma\left(\mathbf{W}_{f_2}[(\vec{h}_i - \vec{h}_j)\odot(\vec{h}_i - \vec{h}_j)]\right), \quad (10)$$

where $\mathbf{W}_{f_2} \in \mathbb{R}^{Z \times P'}$ is a learnable matrix, Z is the dimension of $\overrightarrow{r}_{ij}$, and $\odot$ is element-wise matrix multiplication.

The final edge representation is calculated by combining $\overrightarrow{m}_{ij}$ and $\overrightarrow{r}_{ij}$ as follows:

$$\overrightarrow{f}'_{ij} = \sigma\left(\mathbf{W}_{f_3}(\overrightarrow{m}_{ij} \oplus \overrightarrow{r}_{ij})\right), \tag{11}$$

where $\mathbf{W}_{f_3} \in \mathbb{R}^{Q' \times (U+Z)}$ is a learnable matrix.

In this work, we use Leaky ReLU [56] as the activation function, defined as $\sigma(x) = \max(0, x) + 0.2\min(0, x)$. For each node learning and edge learning layer, residual connection and batch normalization techniques are applied to facilitate training. We also employ dropout with probability 0.1 for all layers except the input layer.

E. Loss Function

We train the three branches of InstGNN to master three tasks: Node Classification (NC), Edge Classification (EC) and Node Embedding Learning (NE).

1) Node Classification Loss

We train the NC branch to predict the semantic labels of strokes. Since the numbers of strokes in different classes are highly imbalanced, we adopt the cross-entropy loss with median frequency balancing [57]:

$$\mathcal{L}_{\text{NC}} = -\sum_{i=1}^{N}\sum_{s=1}^{M_i} \overrightarrow{w}[Y_s^i] \log \overrightarrow{p}_s[Y_s^i], \tag{12}$$

where $\overrightarrow{w}[Y_s^i]$ is the weight for class Y_s^i, computed as the ratio of the median of class frequencies and the class frequency of Y_s^i. $\overrightarrow{p}_s[Y_s^i]$ is the predicted probability for class Y_s^i, computed by applying *softmax* to the output of the NC branch.

2) Edge Classification Loss

The edges in STRG can be naturally divided into two classes: edges linking strokes from two different symbols (negative edge) and edges linking strokes from the same symbol (positive edge). If all negative edges are removed, STRG is then segmented into several connected subgraphs, each of which corresponds to one symbol.

This observation motivates us to train the EC branch to predict whether or not an edge links two strokes from the same symbol. Since there are many fewer positive edges than negative edges, the weighted cross-entropy loss is used:

$$\mathcal{L}_{\text{EC}} = -\sum_{i=1}^{N}\sum_{e \in \mathcal{E}^i} \overrightarrow{w}[Y_e^i] \log \overrightarrow{p}_e[Y_e^i], \tag{13}$$

where $Y_e^i \in \{-1, 1\}$ is the ground truth label of edge e, which is inferred from the training data; $\overrightarrow{w}[Y_e^i]$ is the weight of class Y_e^i, which is inversely proportional to the number of the edges in training data; $\overrightarrow{p}_e[Y_e^i]$ is the predicted probability for class Y_e^i; and $\mathcal{E}^i$ is the corresponding edge set of diagram D^i.

3) Node Embedding Loss

While the EC branch utilizes local information to perform segmentation, here we design a complementary approach that explores the global information. Inspired by works on instance segmentation for images [58]–[61], we propose to embed nodes of STRG into a vector space such that nodes of the same symbol lie close while nodes of different symbols stay far away. Subsequently, segmentation can be obtained by running a clustering algorithm in the embedding space.

Specifically, we train the NE branch under the metric learning framework. The embedding loss $\mathcal{L}_{\text{NE}}$ is a weighted sum of three terms: (1) an intrasymbol variance loss $\mathcal{L}_{\text{var}}$ that pulls embeddings of the same symbol together, (2) an intersymbol distance loss $\mathcal{L}_{\text{dist}}$ that keeps embeddings of different symbols apart, and (3) a regularization loss $\mathcal{L}_{\text{reg}}$ that punishes the magnitude of embeddings.

$$\begin{aligned}
\overrightarrow{\mu}_t &= \frac{1}{|S_t|}\sum_{i \in S_t} \overrightarrow{h}_i, \\
\mathcal{L}_{\text{var}} &= \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|S_t|}\sum_{i \in S_t}\left[\|\overrightarrow{\mu}_t - \overrightarrow{h}_i\| - \delta_{\text{var}}\right]_+^2, \\
\mathcal{L}_{\text{dist}} &= \frac{1}{T(T-1)}\sum_{t_A=1}^{T}\sum_{\substack{t_B=1 \\ t_B \neq t_A}}^{T}\left[2\delta_{\text{dist}} - \|\overrightarrow{\mu}_{t_A} - \overrightarrow{\mu}_{t_B}\|\right]_+^2, \\
\mathcal{L}_{\text{reg}} &= \frac{1}{T}\sum_{t=1}^{T}\|\overrightarrow{\mu}_t\|, \\
\mathcal{L}_{\text{NE}} &= \mathcal{L}_{\text{var}} + \mathcal{L}_{\text{dist}} + \gamma_{\text{reg}}\mathcal{L}_{\text{reg}}.
\end{aligned} \tag{14}$$

Here, $\overrightarrow{h}_i$ is the embedding of stroke i and is generated by the NE branch. $\overrightarrow{\mu}_t$ is the embedding of symbol t and is calculated as the mean of its stroke embeddings. S_t denotes the set of strokes in symbol t, and T is the number of ground truth symbols in a diagram. Furthermore, $\|\cdot\|$ denotes the ℓ_2-norm, and $[x]_+ = \max(0, x)$ is the hinge function. The hyperparameters δ_{var} and δ_{dist} define the margins for the variance and distance loss, respectively. Intuitively, δ_{var} encourages the stroke embeddings of the same symbol to stay close while allowing them to have a distance smaller than $2\delta_{\text{var}}$. δ_{dist} requires the symbol embeddings to stay $2\delta_{\text{dist}}$ apart while allowing them to move freely in the feature space. To make a stroke embedding closer to the stroke embeddings of its own symbol than the stroke embeddings of other symbols, we have to set $\delta_{\text{dist}} > 2\delta_{\text{var}}$ [58]. In all experiments, we set $\delta_{\text{var}} = 0.5, \delta_{\text{dist}} = 1.5, \gamma_{\text{reg}} = 0.001$. Note that $\mathcal{L}_{\text{NE}}$ is defined for each diagram and should be summed over the entire training set. For brevity, we still use $\mathcal{L}_{\text{NE}}$ to represent the total node embedding loss in the following.

4) Joint Loss

We jointly minimize the NC loss, the EC loss and the NE loss during training. The total loss of InstGNN is given by the following:

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{NC}} + \mathcal{L}_{\text{EC}} + \mathcal{L}_{\text{NE}}. \tag{15}$$

F. Network Training

All learnable parameters are initialized with Glorot initialization [62] and optimized with the Adam optimizer [63]. The early stopping strategy is applied to avoid overfitting based on the stroke classification accuracy on the validation set. The feature extraction module is implemented using C++. Both training and inference algorithms are implemented with Deep Graph Library (DGL) [64] and its Pytorch backend [65].

Algorithm 1 EC-based symbol segmentation and recognition

```
Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node prediction $\{\overrightarrow{p}_i | v_i \in \mathcal{V}\}$
    and edge prediction $\{\overrightarrow{p}_{ij} | e_{ij} \in \mathcal{E}\}$, threshold $T_+$
Output: subgraphs $\mathcal{G}' = \{g_c\}$ with class predictions $\{\overrightarrow{p}_c\}$
 1: for all edge $e_{ij} \in \mathcal{E}$ do
 2:     // $p_{ij}^{+}$: the confidence of positive edge $e_{ij}$
 3:     if $p_{ij}^{+} < T_+$ then
 4:         remove edge $e_{ij}$ from $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
 5:     end if
 6: end for
 7: Find all connected subgraphs $\mathcal{G}' = \{g_c\}$ in $\mathcal{G}$
 8: for all subgraph $g_c$ in $\mathcal{G}'$ do
 9:     // Calculate class prediction for subgraph $g_c$
10:     $\overrightarrow{p}_c = \frac{1}{|g_c|} \sum_{i \in g_c} \overrightarrow{p}_i$
11: end for
```

All experiments are conducted on a server with an NVIDIA Geforce TITAN X GPU, except the training of `CASIA-OHFC` with a TITAN RTX. The details of the network architectures and configurations are summarized in Tab. XIII in Appendix B.

G. Inference and Postprocessing

Given the output of InstGNN, we introduce three methods to generate the final symbol segmentation and recognition result: an EC-based method, an NE-based method and their combination.

1) InstGNN-EC: EC-based Method

Given the outputs of the EC branch, we can obtain connected subgraphs by removing all negative edges from the original STRG. Then, each subgraph can be regarded as a symbol instance, and its label can be inferred from the node predictions. Alg. 1 summarizes the symbol segmentation and recognition procedure.

2) InstGNN-NE: NE-based Method

Given the outputs of the NE branch, we perform the mean-shift clustering algorithm [66] to group embeddings into clusters. Each cluster is then considered as a symbol, and its label is obtained by averaging the node predictions.

3) InstGNN: Combination of NE and EC

In spite of its simplicity, the NE-based method often produces poor clusters in practice. We introduce two additional operations to overcome the drawbacks of direct clustering by incorporating EC results:

Rule 1. ***Group*** *the node embeddings into clusters by mean-shift.*

Rule 2. ***Split*** *each cluster into sub-clusters until there are no negative edges within the cluster.*

Rule 3. ***Merge*** *two clusters into a new cluster if there are one or more positive edges connecting them and the labels of two clusters are the same.*

We can sequentially apply **Rule 2** and **Rule 3** to the clusters generated by **Rule 1** and produce a hard segmentation that splits the whole diagram into disjoint subsets. However, inspired by handwritten Chinese text recognition [67], we make use of the over-segmentation strategy to boost performance. Specifically, we apply **Rule 2** and **Rule 3** in parallel to the clusters produced by **Rule 1** and then take the union of the generated clusters as symbol candidates. Since the candidates may overlap with each other, we have to search for the optimal segmentation path. To reduce the search space, symbol candidates are partitioned into three subsets: (1) exclusive candidates, (2) joint candidates and (3) other candidates. An exclusive candidate is a candidate that does not overlap with any other candidates. Exclusive candidates are regarded as real symbols in the diagram and are directly added to the final symbol set.

A joint candidate is a symbol candidate that contains or is contained by another candidate. In practice, we observe that candidates often form triplets where one large candidate is composed of two small disjoint candidates. For such cases, we simply select the optimal candidates according to the bounding boxes of candidates in the triplet. More specifically, if the bounding boxes of the two small disjoint candidates are not overlapping, the two small candidates are preserved, and the large one is removed, and vice versa.

Since the number of type-3 candidates is very limited, we search all legal paths and score them in a brute force manner. Here, the legal path is defined as several mutually disjoint symbol candidates. The score of a legal path is calculated by:

$$Score_{\text{path}} = \frac{1}{N} \sum_{i=1}^{N} p_i, \tag{16}$$

where p_i is the score of candidate i given by a symbol classifier and N is the number of candidates in the path. We train a ResNet-18 CNN classifier [68] for symbol candidate verification utilizing offline symbol images rendered from online data.

Finally, the results of three subsets are merged as the global optimal symbol set. The overview of the symbol segmentation and recognition process is shown in Fig. 3.

4) Symbol Verification

The label of a symbol can be inferred from the node classification as in EC-based and NE-based methods. To further improve the recognition accuracy, we verify all symbols by the trained ResNet-18 CNN classifier. Specifically, if a recognized symbol's confidence inferred by the InstGNN is less than 90%, we choose the prediction of the CNN classifier; otherwise, we choose the prediction of InstGNN as the final result. The text instances enclosed by one symbol are finally merged into one text instance. We will verify the impact of every part of postprocessing in the experiments section.

Remarks

On the one hand, InstGNN-EC can be seen as a local method that decides whether or not two connected nodes belong to the same symbol. Although the edge classifier can be learned to give high accuracy, the performance of InstGNN-EC is sensitive to graph building parameters K_T and K_S as shown in the experiments. On the other hand, InstGNN-NE can be regarded as a global method that requires the stroke features of the same symbol to be close and the stroke features of different symbols to be dissimilar. Importantly, $\mathcal{L}_\text{NE}$

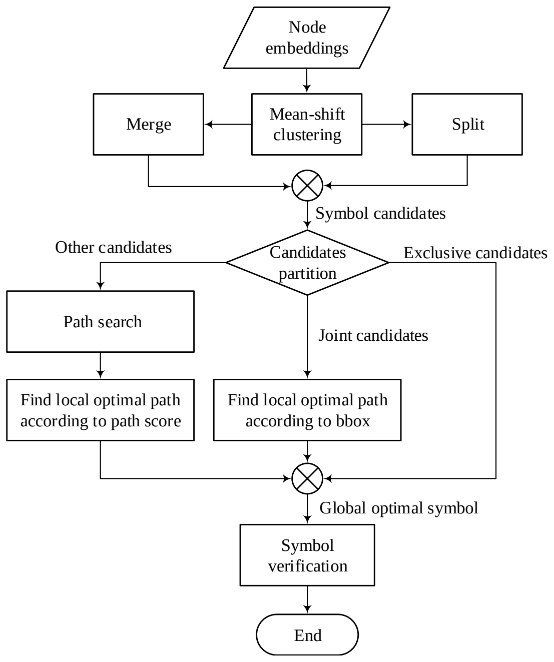


Fig. 3. Symbol segmentation and recognition process. Mean-shift clustering, Split and Merge corresponding to the aforementioned **Rule 1**, **Rule 2** and **Rule 3** in Sect. III-G, respectively.

defined in Eq. 14 is irrelevant to the edges of STRG, and therefore the performance of InstGNN-NE is robust to the edge number. However, the NE task is harder than the EC task, and InstGNN-NE sometimes groups different symbols of the same class into one cluster or splits one symbol into pieces. InstGNN overcomes the above problems by combining the NE and EC results, and this improves the performance accordingly as shown by experiments.

IV. CASIA-OHFC Dataset

Popular handwritten diagram datasets (FC_A [11], FC_B [9], and FA [6]) are small in size and simple in layout and thus are insufficient for the faithful evaluation of different diagram recognition methods. Recently, Gervais et al. [69] released a large-scale digital ink diagram dataset—DIDI that contains a total of 58,655 drawings. However, DIDI only provides raw annotation information without stroke-level and symbol-level annotations, which are very important for supervised learning and model evaluation.

In this work, we release a large-scale online handwritten flowchart dataset called CASIA-OHFC to provide a new benchmark for evaluating different systems and fostering research on diagram recognition. CASIA-OHFC contains 2,957 diagrams that were created from approximately 600 flowchart templates of varying complexity. The diagrams were drawn by 205 writers using Huawei tablets, and each writer drew an average of 15 different diagrams given 15 different templates. CASIA-OHFC involves 31 classes of symbols including common graphic symbols and text. Instance-level annotations of symbols are also provided. We compare CASIA-OHFC with the aforementioned datasets in Tab. II and introduce CASIA-OHFC in more detail in the following.

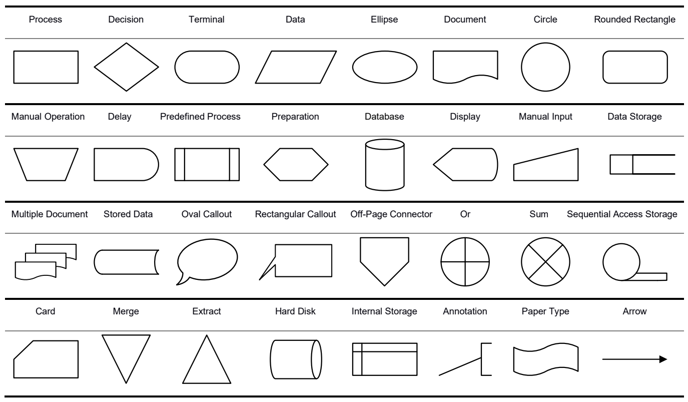
TABLE II
OVERVIEW OF ONLINE HANDWRITTEN DIAGRAM DATASETS.

Dataset	#Classes	Partition	#Writers	#Templates	#Diagrams	#Strokes	#Symbols
FC_A	7	Train	31	14	248	23355	5540
		Test	15	14 others	171	15696	3791
FC_B	7	Train	10	28	280	30443	6195
		Validation	7	28	196	20102	4342
		Test	7	28	196	20139	4343
FA	4	Train	11	12	132	6792	3631
		Validation	7	12	84	4059	2307
		Test	7	12	84	4125	2323
CASIA-OHFC	31	Train	143	600	2073	592267	63368
		Validation	20	215	286	86139	8728
		Test	42	385	598	171313	18280

A. Template Collection

We collected 600 printed flowchart images from the Internet as the templates of CASIA-OHFC. The templates include text and 32 classes of graphic symbols. A full list of the symbol set can be found in Tab. III. The following criteria were taken into consideration when collecting the templates: (1) **Standardization**: The templates should cover most standard graphical symbols in flowcharts. Furthermore, the distribution of symbols (including text) in the templates should match reality as much as possible. (2) **Diversity**: The templates should be from multiple application domains (e.g. engineering, management, programming) and have different layouts. Symbols should have various shapes and scales, especially for arrows. Texts should contain Chinese, English, mathematical notation and formulas, etc. (3) **Complexity**: Practical diagrams have a moderate number of non-arrow symbols. Thus we prefer templates that contain no more than 10 non-arrow symbols. In CASIA-OHFC, a typical template has approximately 10 graphic symbols, several connecting arrows, and some instances of text (inside the graphic symbols or beside the arrows indicating the meaning of every operation).

TABLE III
FLOWCHART SYMBOL SET.



Process	Decision	Terminal	Data	Ellipse	Document	Circle	Rounded Rectangle
Manual Operation	Delay	Predefined Process	Preparation	Database	Display	Manual Input	Data Storage
Multiple Document	Stored Data	Oval Callout	Rectangular Callout	Off-Page Connector	Or	Sum	Sequential Access Storage
Card	Merge	Extract	Hard Disk	Internal Storage	Annotation	Paper Type	Arrow

B. Diagram Drawing

To cover as diverse writing styles as possible, we recruited 205 writers to draw the diagrams. Each individual template was drawn by 5 different writers at most, and each writer drew 15 different diagrams. We used Huawei tablets with a stylus to collect the diagrams. The online diagram was archived in a .txt format file where every stroke is composed of a sequence of points that contain the information of (x,y) coordinates, time, pressure and state of the pen tip.

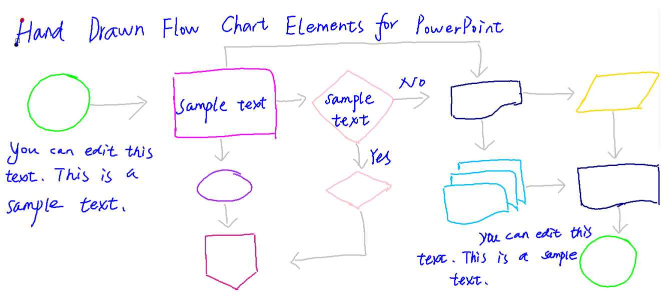

Hand Drawn Flow Chart Elements for PowerPoint
Sample text
Sample text
No
Yes
You can edit this text. This is a sample text.
You can edit this text. This is a sample text.

Fig. 4. An example of an annotated online flowchart in CASIA-OHFC. Symbol classes are denoted by different colors. Symbol IDs are omitted for brevity of display.

C. Preprocessing and Annotation

Before annotating the data, several preprocessing steps are performed. First, isolated dots and small noisy segments are removed according to the pressure and prior knowledge of stroke density. Strokes that belong to more than one symbol are manually split into multiple segments to ensure that every stroke only belongs to one symbol. Furthermore, some ambiguous or carelessly written strokes and symbols are eliminated manually. Finally, two extremely scarce classes (i.e., '*data storage*' and '*sequential access storage*') are removed from the raw data. Finally, there are 30 symbol classes.

We provide two kinds of annotations for each stroke: the semantic class and instance ID of its associate symbol. We do not provide the detailed labels of '*arrows*' ('*head*', '*shaft*') and the contents of '*texts*', nor the relationships between symbols in the current edition, unlike previous datasets [6], [9]. The dataset is released in Ink Markup Language standard [2]. Fig. 4 shows an example of an annotated online diagram.

D. Data Partitioning

To prevent the writers' drawing styles from affecting performance evaluation, the diagrams are randomly divided into three subsets—a training set, a validation set and a test set—at a ratio of 7:1:2 according to the 205 writers. Therefore, there are 143/20/42 writers in the training/validation/test set, respectively. The statistical details of the CASIA-OHFC dataset are shown in Tab. IV.[3]

[2]http://www.w3.org/TR/InkML

[3]Note '*other*' includes 13 small classes: 'stored data', 'oval callout', 'rectangular callout', 'off page connector', 'or', 'summing junction', 'card', 'internal storage', 'merge', 'extract', 'hard disk', 'annotation' and 'paper type'.

TABLE IV
STATISTICS OF STROKES AND SYMBOLS IN THE CASIA-OHFC DATASET.

	Train		Validation		Test	
	Strokes	Symbols	Strokes	Symbols	Strokes	Symbols
Text	494983	25312	72395	3502	141164	7388
Arrow	47310	19945	6594	2713	13887	5768
Process	23089	8242	3298	1111	7800	2499
Decision	10604	3430	1273	439	3369	986
Terminal	4474	1954	479	238	1247	479
Data	3278	1058	560	178	994	293
Ellipse	1056	984	158	142	300	268
Circle	451	431	73	71	155	141
Document	1583	466	196	75	455	118
Predefined Process	997	163	124	26	282	51
Rounded Rectangle	585	287	59	33	286	93
Preparation	556	145	77	23	176	43
Database	445	102	109	24	144	33
Delay	332	171	64	31	117	58
Manual Operation	349	109	59	19	138	37
Manual Input	179	60	20	8	86	24
Display	229	86	16	4	164	33
Multiple Document	512	49	244	24	193	19
Other	1255	344	244	67	356	81
Total	592267	63368	86139	8728	171313	18280

V. EXPERIMENTS

A. Datasets

We evaluate our method on CASIA-OHFC and three public online handwritten diagram datasets: FC_A [11], FC_B [9] and FA [6]. FC_A and FC_B are two flowchart datasets that contain '*text*' and 6 types of graphical symbols: '*terminator*', '*connection*', '*decision*', '*data*', '*process*' and '*arrow*'. FA is a finite automata dataset that encompasses four classes of symbols: '*state*' (circle), '*final state*' (pairwise concentric circles), '*arrow*' and '*label*'. Tab. II shows the details of the three datasets.

B. Evaluation Metrics

Following previous works, we adopt stroke classification accuracy (SCA) to evaluate stroke classification performance and use symbol recognition recall (SRR) and symbol recognition precision (SRP) to evaluate symbol segmentation and recognition performance.

For stroke classification metrics, per-class SCA, overall SCA and class-averaged SCA are calculated as follows.

1) *Per-Class Stroke Classification Accuracy*

$$SCA_{c}=\frac{\sum_{i=1}^{N}\sum_{s=1}^{M_i}\mathbb{I}(Y_s^i=c)\mathbb{I}(\widehat{Y}_s^i=Y_s^i)}{\sum_{i=1}^{N}\sum_{s=1}^{M_i}\mathbb{I}(Y_s^i=c)}$$

2) *Overall Stroke Classification Accuracy*

$$SCA_{\text{overall}}=\frac{\sum_{i=1}^{N}\sum_{s=1}^{M_i}\mathbb{I}(\widehat{Y}_s^i=Y_s^i)}{\sum_{i=1}^{N}M_i}$$

3) *Average Stroke Classification Accuracy*

$$SCA_{\text{avg}}=\frac{1}{C}\sum_{c=1}^{C}SCA_c$$

Here, $\{1,...,C\}$ is the set of all stroke classes, N is the number of diagrams, M_i is the number of strokes in diagram i, Y_s^i and $\widehat{Y}_s^i$ are the ground truth label and prediction for stroke s in diagram i, and $\mathbb{I}(\cdot)$ is the indicator function, i.e., $\mathbb{I}(\cdot)=1$ when the condition is true and zero otherwise.

For symbol recognition metrics, per-class SRR/SRP, overall SRR/SRP, and class-averaged SRR/SRP are calculated as follows. The definitions of symbol segmentation metrics are similar to those for symbol recognition and are omitted for conciseness.

4) Per-Class Symbol Recognition Recall

$$SRR_{\text{c}} = \frac{N^c_{\text{corr}}}{N^c_{\text{gt}}}$$

5) Per-Class Symbol Recognition Precision

$$SRP_{\text{c}} = \frac{N^c_{\text{corr}}}{N^c_{\text{pred}}}$$

6) Overall Symbol Recognition Recall

$$SRR_{\text{overall}} = \frac{\sum_{c=1}^{C} N^c_{\text{corr}}}{\sum_{c=1}^{C} N^c_{\text{gt}}}$$

7) Average Symbol Recognition Recall

$$SRR_{\text{avg}} = \frac{1}{C}\sum_{c=1}^{C} SRR_c$$

The overall SRP and class-averaged SRP are defined similarly. Here, N^c_{corr} is the number of correctly recognized symbols of class c, and N^c_{gt} and N^c_{pred} are the numbers of ground truth symbols of class c and predicted symbols of class c, respectively. Note that a symbol is segmented correctly if and only if the predicted symbol and the ground truth symbol contain identical strokes. A symbol is recognized correctly if and only if it is segmented correctly and the predicted class is the same as the ground truth.

C. Ablation Study

We perform three groups of ablation experiments on `FC_B` to evaluate different components of the proposed method.

1) Effect of Graph Construction

We examine the effect of different graph construction methods on the performance of InstGNN. Specifically, we perform experiments by building STRG graphs with different numbers of temporal neighbors K_{T} or spatial neighbors K_{S}. When evaluating the impact of spatial edges, we set $K_{\text{T}} = 1$ for node feature extraction but do not include any temporal edges in STRG. Likewise, when evaluating the impact of temporal edges, we set $K_{\text{S}} = 1$ for node feature extraction but do not include any spatial edges in STRG. For each parameter setting, we repeat the experiment 5 times and report the mean and standard deviation in Fig. 5. To obtain deep insight into InstGNN, we report the node/edge classification accuracy and SRR_{overall} of InstGNN-EC, InstGNN-NE, and InstGNN, respectively.

We make the following observations from Fig. 5. **(1)** The K_{S}-performance curves for spatial edges consistently exceed the K_{T}-performance curves, and the variances of the K_{S}-performance curves are much smaller. This suggests that spatial edges are more effective and informative than temporal edges. **(2)** For both types of graphs, SRR of InstGNN is greater than SRR of InstGNN-NE, while SRR of InstGNN-NE is greater than SRR of InstGNN-EC. **(3)** Although the node and edge classification accuracy rises steadily with increasing K, SRR of InstGNN-EC degrades due to the growth of misclassified edges.

2) Effect of Attention and Edge Learning

To investigate the effect of the attention block and the edge learning layer in InstGNN, we perform experiments with different model architectures with the InstGNN-NE method. Tab. V shows the results under different metrics. If the node attention (Eq. 3), edge attention (Eq. 4) and edge learning branch are all disabled in InstGNN-NE, the performance is extremely poor. The results demonstrate that the node attention and edge attention play important roles in the node embedding learning and node classification. Although edge classification is not performed in InstGNN-NE, the edge learning layer still leads to superior performance and can accelerate the training procedure.

TABLE V
SYMBOL SEGMENTATION AND RECOGNITION RESULTS UNDER DIFFERENT ARCHITECTURE CONFIGURATIONS ON `FC_B` (%). SSP: SYMBOL SEGMENTATION PRECISION, SSR: SYMBOL SEGMENTATION RECALL.

Experiment	Node attention	Edge attention	Edge learning	SCA	Segmentation		Recognition	
					SSP	SSR	SRP	SRR
1:baseline				79.55	33.88	39.01	33.30	38.34
2	✓			92.26	55.30	64.93	54.44	63.92
3	✓	✓		98.65	80.10	87.52	79.28	86.62
4	✓	✓	✓	**99.26**	**97.98**	**98.48**	**97.50**	**98.00**

3) Effect of Postprocessing

We further examine the effect of different postprocessing steps on symbol segmentation and recognition results. Experiments 1-5 in Tab. VI show the symbol segmentation and recognition results only with node embedding (i.e., only **Rule 1** as in Sect. III-G), while experiments 6-12 exploit the node embedding and edge prediction (i.e., three segmentation **Rules** aforementioned). The experimental results justify the effectiveness of the proposed three symbol candidate generation rules, the strategy of combining the CNN classifier and node predictions to obtain symbol recognition results, and the text merging procedure. In particular, the inference with all three rules and postprocessing steps gives the best symbol recognition result.

TABLE VI
SYMBOL SEGMENTATION AND RECOGNITION RESULTS UNDER DIFFERENT POSTPROCESSING CONFIGURATIONS ON `FC_B` (%). CLUSTER, SPLIT AND MERGE CORRESPONDING TO **RULE 1**, **RULE 2** AND **RULE 3** AS MENTIONED IN SECT. III-G. SSP: SYMBOL SEGMENTATION PRECISION, SSR: SYMBOL SEGMENTATION RECALL.

Experiment	Candidate generation rule			Symbol prediction		Text merge	Segmentation		Recognition	
	Cluster	Split	Merge	CNN	Node prediction		SSP	SSR	SRP	SRR
1: baseline	✓				✓		97.80	98.41	96.16	96.75
2	✓			✓			97.80	98.41	96.59	97.19
3	✓			✓	✓		97.80	98.41	97.32	97.93
4	✓				✓	✓	**97.98**	**98.50**	96.34	96.85
5	✓			✓	✓	✓	**97.98**	98.48	**97.50**	**98.00**
6	✓	✓		✓	✓		96.25	98.00	95.70	97.44
7	✓		✓	✓	✓		98.28	98.50	97.79	98.02
8	✓	✓	✓		✓		97.97	98.80	96.21	97.03
9	✓	✓	✓	✓	✓		97.97	98.80	97.42	98.25
10	✓	✓	✓		✓	✓	**98.38**	**98.99**	96.61	97.21
11	✓	✓	✓	✓		✓	98.26	98.92	97.03	97.67
12	✓	✓	✓	✓	✓	✓	98.37	98.96	**97.83**	**98.41**

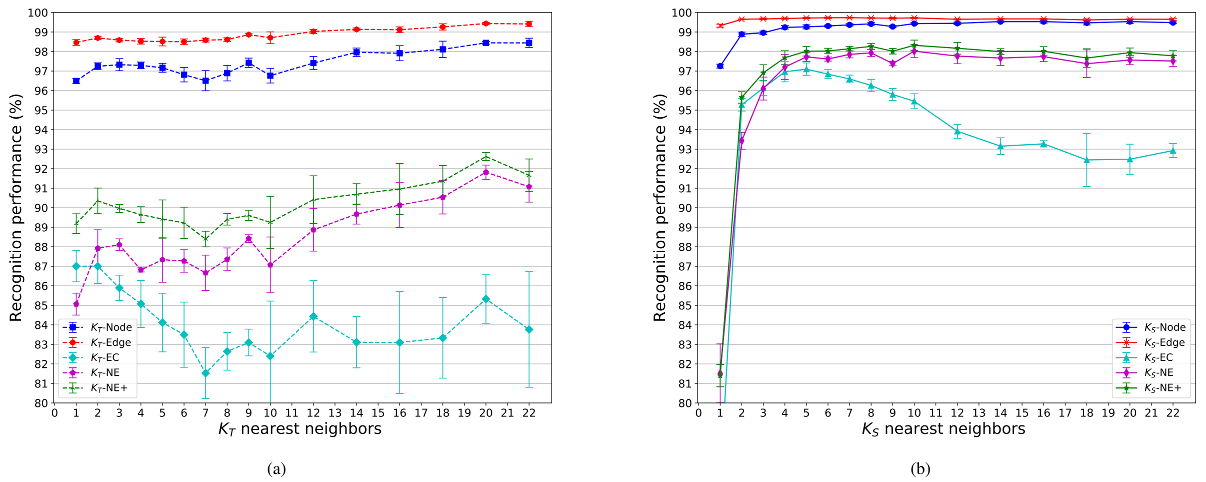


Fig. 5. Method performance vs. different graph building methods on FC_B. (a) STRG graphs built with $K_T > 0$ and $K_S = 0$. (b) STRG graphs built with $K_S > 0$ and $K_T = 0$. K-Node: node classification accuracy, K-Edge: edge classification accuracy. K-EC, K-NE, K-NE+: the overall symbol recognition recall of InstGNN-EC, InstGNN-NE and InstGNN, respectively.

D. Comparison with the State-of-the-Art

The stroke classification accuracy and symbol recognition results of InstGNN-EC, InstGNN-NE and InstGNN on three public benchmark datasets are reported in Tab. VII, VIII and IX [4], respectively. In the tables, numbers in boldface indicate the best results. The results of comparison methods are directly cited from the original papers. Previous works did not report the class-averaged values, which we calculate from the reported accuracies for all classes in the datasets.

TABLE VII
COMPARISONS WITH STATE-OF-THE-ART METHODS ON FC_A (%). WE REPORT THE OVERALL AND AVERAGE STROKE CLASSIFICATION ACCURACY (SCA) AND SYMBOL RECOGNITION RECALL (SRR) RESULTS. THE BEST RESULTS ARE IN **BOLD FACE**.

	Method	Arrow	Connection	Data	Decision	Process	Terminator	Text	Overall	Avg.
SCA	Lemaitre [10]	79.6	80.0	84.7	84	85.7	79.6	97.8	91.1	84.49
	Carton [5]	83.8	80.3	84.3	90.9	90.4	69.8	97.2	92.4	85.24
	Bresler [8]	88.7	94.1	96.4	90.9	95.2	90.2	99.3	96.5	93.54
	Wu [24]	87.4	73.7	87.6	89.7	91.8	91.6	98.8	94.9	88.66
	Wang [13]	92.1	79.3	87.6	89.7	93.5	78.9	99.0	95.8	88.59
	Bresler [9]	87.5	94.1	95.3	88.2	**96.3**	90.7	99.2	96.3	93.04
	Wang [12]	98.18	83.22	93.28	95.69	93.13	89.29	96.81	96.19	92.80
	Yun [55]	97.28	94.96	93.81	93.61	93.71	93.88	98.48	97.27	95.10
	Schäfer [28]	-	-	-	-	-	-	-	97.7	-
	Julca-Aguilar [14]	-	-	-	-	-	-	-	91.1	-
	InstGNN	**98.32**	**97.04**	**96.75**	**97.70**	94.25	**94.62**	**99.38**	**98.44**	**96.86**
SRR	Lemaitre [10]	68.9	81.4	80.4	66.5	81.3	70.3	71.7	72.4	74.36
	Carton [5]	70.2	82.4	80.5	80.6	85.2	72.4	74.1	75.0	77.91
	Bresler [8]	78.1	95.1	90.6	75.3	88.1	88.9	89.7	84.43	86.54
	Wu [24]	80.3	73.4	78.5	78.9	88.3	90.6	86.0	83.2	82.29
	Wang [13]	83.4	79.8	84.4	76.9	89.2	80.8	85.8	84.3	82.90
	Bresler [9]	76.6	95.1	90.5	72.9	88.6	89.0	89.7	84.2	86.06
	Wang [12]	76.89	92.74	89.80	82.07	89.95	77.83	78.93	80.85	84.03
	Schäfer [28]	-	-	-	-	-	-	-	93.5	-
	Julca-Aguilar [14]	-	-	-	-	-	-	-	85.5	-
	InstGNN-EC	93.49	**96.00**	**97.95**	87.68	97.29	**98.52**	94.89	94.75	95.12
	InstGNN-NE	92.06	**96.00**	95.90	88.15	97.29	**98.52**	91.09	92.85	94.14
	InstGNN	**93.73**	**96.00**	**97.95**	**89.10**	**97.78**	**98.52**	**95.12**	**95.04**	**95.46**

[4]General arrow and initial arrow classes are not distinguished in FA dataset v1.1.

TABLE VIII
COMPARISONS WITH STATE-OF-THE-ART METHODS ON FC_B (%). WE REPORT THE OVERALL AND AVERAGE (AVG.) STROKE CLASSIFICATION ACCURACY (SCA) AND SYMBOL RECOGNITION RECALL (SRR) RESULTS. THE BEST RESULTS ARE IN **BOLD FACE**.

	Method	Arrow	Connection	Data	Decision	Process	Terminator	Text	Overall	Avg.
SCA	Bresler [9]	98.3	88.4	96.1	90.3	**98.4**	99.7	99.6	98.4	95.83
	Yun [55]	97.16	93.39	**96.30**	95.94	96.90	97.89	**99.86**	99.04	96.78
	InstGNN	**99.21**	**98.21**	93.92	**98.59**	96.44	**100.00**	99.82	**99.26**	**98.03**
SRR	Bresler [9]	93.2	88.4	93.8	92.0	97.6	98.9	97.1	95.3	94.43
	InstGNN-EC	97.23	**97.32**	95.22	94.20	**97.89**	**100.00**	98.68	97.70	97.22
	InstGNN-NE	97.53	**97.32**	**96.63**	95.98	**97.89**	99.62	98.74	98.00	97.67
	InstGNN	**98.50**	**97.32**	**96.63**	**95.98**	**97.89**	**100.00**	**98.98**	**98.41**	**97.90**

TABLE IX
COMPARISONS WITH STATE-OF-THE-ART METHODS ON FA (%). WE REPORT THE OVERALL AND AVERAGE STROKE CLASSIFICATION ACCURACY (SCA) AND SYMBOL RECOGNITION RECALL (SRR) RESULTS. THE BEST RESULTS ARE IN **BOLD FACE**.

	Method	Label	Arrow	Initial arrow	State	Final state	Overall	Avg.
SCA	Bresler [6]	99.1	89.3	78.5	95.2	96.1	94.5	91.64
	Bresler [8]	99.8	94.9	85.0	96.9	99.2	97.4	95.16
	Wang [13]	99.0	97.7	-	91.6	96.5	97.8	96.20
	Bresler [9]	99.7	98.0	98.6	98.3	**99.2**	99.0	98.76
	Yun [55]	99.83	99.10	-	97.38	97.12	99.22	98.36
	InstGNN	**99.95**	**99.93**	-	**98.96**	98.46	**99.78**	**99.33**
SRR	Bresler [6]	96.0	84.4	80.0	94.5	93.8	91.5	89.74
	Bresler [8]	99.1	92.8	84.0	97.2	98.4	96.4	94.3
	Wang [13]	98.1	95.3	-	91.2	89.1	95.8	93.42
	Bresler [9]	99.2	97.5	97.3	98.2	**99.2**	98.5	98.28
	InstGNN-EC	99.46	97.74	-	**98.94**	98.45	98.75	98.65
	InstGNN-NE	97.67	97.86	-	98.24	98.45	97.85	98.05
	InstGNN	**99.91**	**98.62**	-	**98.94**	98.45	**99.27**	**98.98**

Our proposed three methods all achieve new state-of-the-art or comparable performance. Notably, InstGNN obtains the best results among the three methods and significantly outperforms previous works on all three datasets, especially on FC_A. Note that works [9], [28] used additional synthetic samples, which played an important role in their performance.

Another important metric is the whole diagram recognition accuracy, which refers to how many symbols are misrecognized for each diagram. We report the results of InstGNN in Fig. 6. Approximately 50%, 75% and 90% diagrams are perfectly recognized for FC_A, FC_B and FA, respectively. Most of the diagrams are recognized with less than 3 misrecognized symbols.

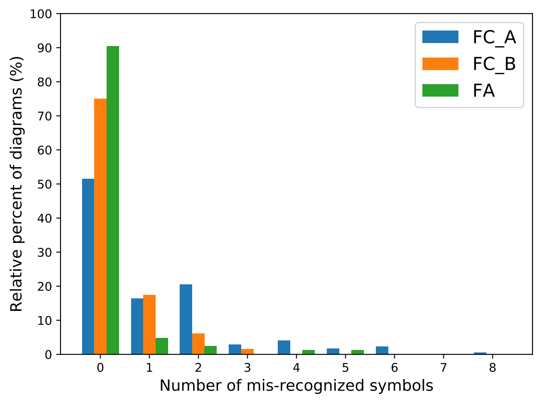


Fig. 6. The whole diagram recognition accuracy using InstGNN.

Furthermore, our method is computationally efficient in both training and testing. For example, on FC_A/FC_B/FA, it takes approximately 20/30/15 mins to train our model and 700/750/450 ms to obtain the symbol segmentation and recognition results with InstGNN for each diagram under the settings described in Sect. III-F.

To better understand InstGNN, we visualize the embeddings of nodes learned by the NE branch for a flowchart in Fig. 7. The 48D features are reduced to 2D with t-distributed stochastic neighbor embedding (t-SNE) [70] for visualization. The figure shows that the node embeddings of the same symbol are very close while the node embeddings belonging to different symbols are far apart.

Fig. 8 shows some examples of OHDR results from FC_A using InstGNN. Some segmentation errors or recognition errors are shown.

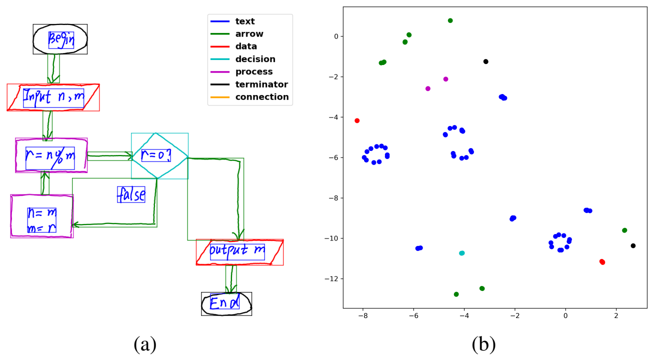


Fig. 7. Visualization of a recognized flowchart (a) and the corresponding learned node embeddings (b) with InstGNN. The colors encode the stroke/node classes, and the rectangles denote the recognized symbol instances.

E. Results on CASIA-OHFC

We find that some classes in CASIA-OHFC are very difficult to recognize due to writing vagueness and the lack of training samples. To make the experimental results more stable, we merge the '*rounded rectangular*' symbol into the '*process*' class and combine the classes that have less than 90 symbol instances in the whole dataset into the '*other*' category. Therefore, there are only 18 classes including symbols and '*text*' in our experiment.

The stroke classification and symbol recognition results on the proposed CASIA-OHFC dataset are shown in Tab. X. The class distributions are highly imbalanced, particularly the strokes of '*text*', which account for approximately 80% of CASIA-OHFC. Although the overall SCA (95.80%) is excellent, the class-averaged SCA (70.87%) is not satisfactory due to the imbalance of symbol classes. The diagrams in CASIA-OHFC are more complex and variable than those in the three public datasets. For instance, forking or convergence arrows are included in CASIA-OHFC, and some handwritten symbols are indistinguishable. The best overall SRP/SRR among the three methods is 78.65%/79.47%. Furthermore, 36.5% of diagrams are recognized with no more than 3 misrecognized symbols. The training time and the average inference time per diagram are approximately 7 h and 3 s, respectively, because both the number of diagrams and the number of strokes in each diagram are much larger in CASIA-OHFC than in FC_A/FC_B/FA.

TABLE X
EXPERIMENT RESULTS ON CASIA-OHFC WITH THE THREE PROPOSED METHODS (%). WE REPORT THE OVERALL AND AVERAGE STROKE CLASSIFICATION ACCURACY (SCA), SYMBOL RECOGNITION PRECISION (SRP) AND SYMBOL RECOGNITION RECALL (SRR) RESULTS, RESPECTIVELY. THE BEST RESULTS ARE IN **BOLD FACE**.

	SCA	InstGNN-EC		InstGNN-NE		InstGNN	
		SRP	SRR	SRP	SRR	SRP	SRR
Text	98.40	79.85	71.09	84.33	77.40	**85.22**	**79.23**
Arrow	91.61	**68.54**	69.69	63.58	74.68	66.19	**74.70**
Process	73.53	90.72	81.89	90.92	87.75	**92.33**	**87.75**
Decision	95.87	77.95	79.61	76.64	82.86	**80.95**	**84.48**
Terminal	72.17	**88.58**	81.90	88.10	**82.97**	87.70	**82.97**
Data	78.67	**88.15**	81.79	84.72	**83.85**	87.32	82.82
Ellipse	82.00	83.51	89.81	**84.48**	**92.45**	84.43	92.08
Circle	86.45	**77.12**	83.69	73.49	**86.52**	73.33	85.82
Document	72.53	**75.41**	80.00	71.54	**80.87**	73.22	**80.87**
Predefined Process	85.11	66.04	70.00	50.00	68.00	**67.80**	**80.00**
Preparation	79.55	84.21	74.42	**87.50**	**81.40**	86.84	76.74
Database	60.42	37.50	63.64	56.52	**78.79**	**57.78**	**78.79**
Delay	40.17	**74.07**	70.18	66.13	71.93	72.41	**73.68**
Manual Input	41.86	76.19	66.67	76.19	66.67	76.19	66.67
Manual Operation	39.86	**65.71**	62.16	61.90	**70.27**	64.10	67.57
Display	61.59	67.86	59.38	61.11	68.75	**68.57**	**75.00**
Multiple Document	69.43	28.13	50.00	36.00	50.00	**40.74**	**61.11**
Other	46.42	**46.97**	39.24	37.23	44.30	38.30	**45.57**
Overall	95.80	77.32	73.29	76.63	78.62	**78.65**	**79.47**
Avg.	70.87	70.92	70.84	69.47	74.97	**72.41**	**76.44**

VI. CONCLUSION

In this paper, we have proposed a novel and general multitask learning framework based on GNNs for online handwritten diagram recognition. Experiments on two flowchart benchmark datasets and one finite automata dataset demonstrate that the proposed framework is capable of encoding complex spatial and temporal relationships efficiently for symbol segmentation and recognition. Our method outperforms several recently proposed methods by a prominent margin. Moreover, we have presented a large-scale online handwritten diagram dataset CASIA-OHFC that is publicly available and performed preliminary experiments on it to provide a benchmark. We believe that our proposed framework can be adequately extended and adapted to general online handwritten diagram and freehand sketch recognition tasks. In the future,

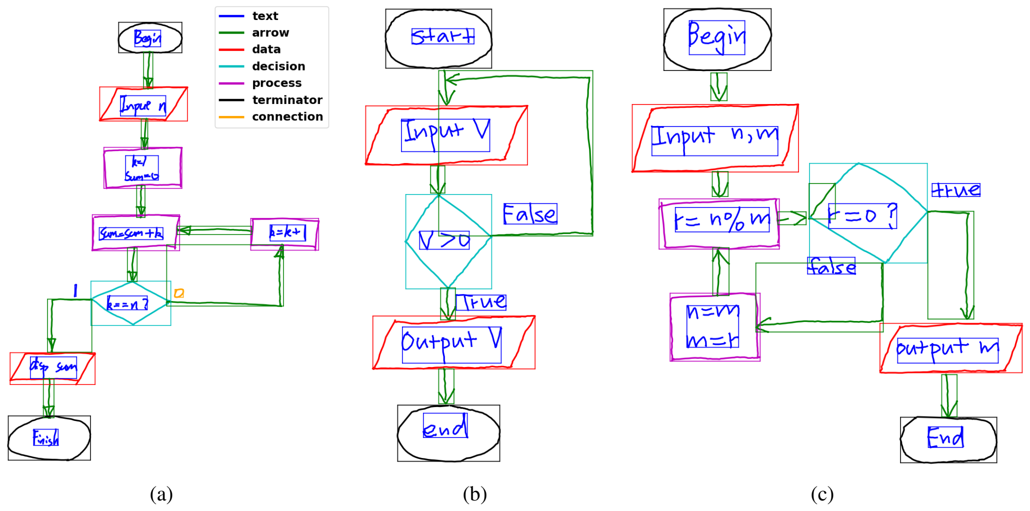


Fig. 8. Examples of recognized flowcharts from FC_A with InstGNN. (a) A '*text*' beside the 'decision' symbol (only one letter) is segmented correctly but is misclassified as a '*connection*'. (b) Two '*arrow*' (one of them is very short) are segmented as one. (c) One of the strokes in '*decision*' (one of its side line is split) is incorrectly classified as an '*arrow*', which results in an incomplete '*decision*'.

replacing handcrafted stroke and interstroke features with neural network-based feature learning can be considered to further improve the recognition performance, and we will explore how to recognize strokes and symbols interactively and dynamically while the diagram is drawn on the touch screen.

ACKNOWLEDGMENTS

This work is supported by the Major Project for New Generation of AI under Grant No. 2018AAA0100400 and the National Natural Science Foundation of China (NSFC) Grant Nos. 61773376 and 61721004.

REFERENCES

[1] L. Gennari, L. B. Kara, T. F. Stahovich, and K. Shimada, "Combining geometry and domain knowledge to interpret hand-drawn diagrams," *Comput. Graphics*, vol. 29, no. 4, pp. 547–562, 2005.

[2] Z. Yuan, H. Pan, and L. Zhang, "A novel pen-based flowchart recognition system for programming teaching," in *Proc. Workshop on Blended Learning*, 2008, pp. 55–64.

[3] G. Feng, C. Viardgaudin, and Z. Sun, "On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming," *Pattern Recognit.*, vol. 42, no. 12, pp. 3215–3223, 2009.

[4] H. Miyao and R. Maruyama, "On-line handwritten flowchart recognition, beautification and editing system," in *Proc. Int. Conf. Frontiers Handwriting Recognit.*, 2012, pp. 83–88.

[5] C. Carton, A. Lemaitre, and B. Coüasnon, "Fusion of statistical and structural information for flowchart recognition," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2013, pp. 1210–1214.

[6] M. Bresler, T. Van Phan, D. Průša, M. Nakagawa, and V. Hlavác, "Recognition system for on-line sketched diagrams," in *Proc. Int. Conf. Front. Handwrit. Recognit.*, 2014, pp. 563–568.

[7] G. Costagliola, M. De Rosa, and V. Fuccella, "Local context-based recognition of sketched diagrams," *J. Vis. Lang. Comput.*, vol. 25, no. 6, pp. 955–962, 2014.

[8] M. Bresler, D. Průša, and V. Hlavác, "Detection of arrows in on-line sketched diagrams using relative stroke positioning," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2015, pp. 610–617.

[9] M. Bresler, D. Průša, and V. Hlaváč, "Online recognition of sketched arrow-connected diagrams," *Int. J. Doc. Anal. Recognit.*, vol. 19, no. 3, pp. 253–267, 2016.

[10] A. Lemaitre, H. Mouchère, J. Camillerapp, and B. Coüasnon, "Interest of syntactic knowledge for on-line flowchart recognition," in *Proc. Int. Workshop on GREC*, 2011, pp. 89–98.

[11] A.-M. Awal, G. Feng, H. Mouchere, and C. Viard-Gaudin, "First experiments on a new online handwritten flowchart database," in *Proc SPIE Int. Soc. Opt. Eng.*, 2011, p. 78740A.

[12] C. Wang, H. Mouchère, A. Lemaitre, and C. Viard-Gaudin, "Online flowchart understanding by combining max-margin markov random field with grammatical analysis," *Int. J. Doc. Anal. Recognit.*, vol. 20, no. 2, pp. 123–136, 2017.

[13] C. Wang, H. Mouchere, C. Viard-Gaudin, and L. Jin, "Combined segmentation and recognition of online handwritten diagrams with high order markov random field," in *Proc Int. Conf. Front. Handwrit. Recognit.*, 2016, pp. 252–257.

[14] F. Julca-Aguilar, H. Mouchère, C. Viard-Gaudin, and N. S. T. Hirata, "A general framework for the recognition of online handwritten graphics," *Int. J. Doc. Anal. Recognit.*, vol. 23, pp. 143–160, 2020.

[15] J. Zhang, J. Du, and L. Dai, "Track, attend, and parse (tap): An end-to-end framework for online handwritten mathematical expression recognition," *IEEE Trans Multimedia*, vol. 21, no. 1, pp. 221–233, 2019.

[16] Z. Huang, H. Fu, and R. W. H. Lau, "Data-driven segmentation and labeling of freehand sketches," *ACM Trans. Graph.*, vol. 33, no. 6, 2014.

[17] R. G. Schneider and T. Tuytelaars, "Example-based sketch segmentation and labeling using crfs," *ACM Trans. Graph.*, vol. 35, no. 5, Jul. 2016.

[18] R. K. Sarvadevabhatla, I. Dwivedi, A. Biswas, S. Manocha, and V. B. R., "Sketchparse: Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks," in *Proc. ACM Int. Conf. Multimedia*, 2017, p. 10–18.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[21] B. Coüasnon, "Dmos, a generic document recognition method: Application to table structure analysis in a general and in a specific way," *Int. J. Doc. Anal. Recognit.*, vol. 8, no. 2-3, pp. 111–122, 2006.

[22] M. Bresler, D. Průša, and V. Hlavác, "Modeling flowchart structure recognition as a max-sum problem," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2013, pp. 1215–1219.

[23] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "An integrated grammar based approach for mathematical expression recognition," *Pattern Recognit.*, vol. 51, pp. 135–147, 2016.

[24] J. Wu, C. Wang, L. Zhang, and Y. Rui, "Offline sketch parsing via shapeness estimation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1200–1207.

[25] M. Bresler, D. Prusa, and V. Hlavac, "Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques," in *Proc. Int. Conf. Front. Handwrit. Recognit.*, 2016, pp. 48–53.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Proc. Syst*, 2015, pp. 91–99.

[27] F. D. Julca-Aguilar and N. S. Hirata, "Symbol detection in online handwritten graphics using faster r-cnn," in *Proc. IAPR Int. Workshop Doc. Anal. Syst.*, 2018, pp. 151–156.

[28] B. Schäfer and H. Stuckenschmidt, "Arrow r-cnn for flowchart recognition," in *Proc.Workshops Int. Conf. Doc. Anal. Recognit.*, 2019.

[29] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *Int. J. Doc. Anal. Recognit.*, vol. 15, no. 4, pp. 331–337, 2012.

[30] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models," *Pattern Recognit. Lett.*, vol. 35, pp. 58 – 67, 2014.

[31] S. MacLean and G. Labahn, "A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets," *Int. J. Doc. Anal. Recognit.*, vol. 16, pp. 139–163, 2013.

[32] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush, "Image-to-markup generation with coarse-to-fine attention," in *Proc. Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 980–989.

[33] J. Zhang, J. Du, and L. Dai, "A gru-based encoder-decoder approach with attention for online handwritten mathematical expression recognition," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2017, pp. 902–907.

[34] J. Zhang, J. Du, S. Zhang, D. Liu, Y. Hu, J. Hu, S. Wei, and L. Dai, "Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition," *Pattern Recognition*, vol. 71, pp. 196 – 206, 2017.

[35] A. D. Le and M. Nakagawa, "Training an end-to-end system for handwritten mathematical expression recognition by generated patterns," in *Proc. Int. Conf. Doc. Anal. Recognit.*, vol. 28, 2017, pp. 1056–1061.

[36] J.-W. Wu, F. Yin, Y.-M. Zhang, X.-Y. Zhang, and C.-L. Liu, "Handwritten mathematical expression recognition via paired adversarial learning," *Int. J. Comput. Vis.*, vol. 128, pp. 2386–2401, 2020.

[37] X. Zhang, X. Li, Y. Liu, and F. Feng, "A survey on freehand sketch recognition and retrieval," *Image Vision Comput.*, vol. 89, pp. 67–87, 2019.

[38] P. Xu, "Deep learning for free-hand sketch: A survey," *arXiv preprint arXiv:2001.02600v1*, 2020.

[39] X. Zhu, Y. Xiao, and Y. Zheng, "2d freehand sketch labeling using cnn and crf," *Multimedia Tools and Applications*, p. 1–18, 2019.

[40] K. Kaiyrbekov and M. Sezgin, "Deep stroke-based sketched symbol reconstruction and segmentation," *IEEE Comput. Graph. Appl.*, p. 112–126, 2019.

[41] L. Yang, J. Zhuang, H. Fu, K. Zhou, and Y. Zheng, "Sketchgcn: Semantic sketch segmentation with graph convolutional networks," *arXiv preprint arXiv:2003.00678v1*, 2020.

[42] F. Wang, S. Lin, H. Wu, H. Li, R. Wang, X. Luo, and X. He, "Spfusionnet: Sketch segmentation using multi-modal data fusion," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2019, pp. 1654–1659.

[43] D. Ha and D. Eck, "A neural representation of sketch drawings," in *Proc. Int. Conf. Learn. Represent*, 2018.

[44] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[45] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," in *Proc. Adv. Neural Inf. Proc. Syst*, 2016, pp. 4509–4517.

[46] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," *arXiv preprint arXiv:1809.05679*, 2018.

[47] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[48] D. Xu, Y. Zhu, C. B. Choy, and F.-F. Li, "Scene graph generation by iterative message passing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3097–3106.

[49] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. Mckee, J. B. Tenenbaum, and P. W. Battaglia, "Relational inductive bias for physical construction in humans and machines," in *The Annual Meeting of the Cognitive Science Society*, 2018.

[50] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9203–9211.

[51] J.-Y. Ye, Y.-M. Zhang, Q. Yang, and C.-L. Liu, "Contextual stroke classification in online handwritten documents with graph attention networks," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2019, pp. 993–998.

[52] J.-Y. Ye, Y.-M. Zhang, and C.-L. Liu, "Joint training of conditional random fields and neural networks for stroke classification in online handwritten documents," in *Proc. Int. Conf. Pattern Recognit.*, 2016, pp. 3264–3269.

[53] T. Van Phan and M. Nakagawa, "Combination of global and local contexts for text/non-text classification in heterogeneous online handwritten documents," *Pattern Recognit.*, vol. 51, pp. 112–124, 2016.

[54] A. Delaye and C.-L. Liu, "Contextual text/non-text stroke classification in online handwritten notes with conditional random fields," *Pattern Recognit.*, vol. 47, no. 3, pp. 959–968, 2014.

[55] X.-L. Yun, Y.-M. Zhang, J.-Y. Ye, and C.-L. Liu, "Online handwritten diagram recognition with graph attention networks," in *Proc. Int. Conf. Image Graphics*, 2019, pp. 232–244.

[56] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. Int. Conf. Mach. Learn.*, vol. 28, 2013.

[57] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 2650–2658.

[58] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshop*, 2017.

[59] A. Fathi, Z. Wojna, V. Rathod, P. Wang, H. O. Song, S. Guadarrama, and K. Murphy, "Semantic instance segmentation via deep metric learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[60] C. Payer, D. Stern, T. Neff, H. Bischof, and M. Urschler, "Instance segmentation and tracking with cosine embeddings and recurrent hourglass networks," in *Proc. Med. Image Comput. Comput.-Assisted Intervention*, 2018, pp. 3–11.

[61] J. Lahoud, B. Ghanem, M. R. Oswald, and M. Pollefeys, "3d instance segmentation via multi-task metric learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 9255–9265.

[62] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[64] M. Wang, L. Yu, D. Zheng *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[65] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Advances Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[66] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Trans. Inf. Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[67] Q.-F. Wang, F. Yin, and C.-L. Liu, "Handwritten chinese text recognition by integrating multiple contexts," *IEEE Trans on Pattern Anal. Mach. Intell.*, vol. 34, no. 8, pp. 1469–1481, 2012.

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[69] P. Gervais, T. Deselaers, E. Aksan, and O. Hilliges, "The didi dataset: Digital ink diagram data," *arXiv preprint arXiv:2002.09303*, 2020.

[70] L. V. Der Maaten and G. E. Hinton, "Visualizing data using t-sne," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

Xiao-Long Yun received a BS in Automation from Sichuan University, Sichuan, China, in 2016. He is currently pursuing his master's degree in computer technology at the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include online handwritten diagram recognition, graph neural networks, and computer vision.

Yan-Ming Zhang received a bachelor's degree from Beijing University of Posts and Telecommunications and a PhD in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, in 2011. He is an associate professor at the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences. His research interests include machine learning, pattern recognition and their application in document analysis.

Fei Yin received a BS in computer science from Xi'an University of Posts and Telecommunications, Xi'an, China, in 1999, an ME in pattern recognition and intelligent systems from Huazhong University of Science and Technology, Wuhan, China, in 2002, and a PhD degree in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2010. He is an associate professor at the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences. His research interests include document image analysis, handwritten character recognition, and image processing. He has published more than 50 papers in international journals and conferences.

Cheng-Lin Liu is a Professor at the National Laboratory of Pattern Recognition (NLPR), Institute of Automation of Chinese Academy of Sciences, Beijing, China, and is now the director of the laboratory. He received a BS in electronic engineering from Wuhan University, Wuhan, China, an ME in electronic engineering from Beijing Polytechnic University, Beijing, China, and a PhD in pattern recognition and intelligent control from the Institute of Automation of Chinese Academy of Sciences, Beijing, China, in 1989, 1992 and 1995, respectively. He was a postdoctoral fellow at Korea Advanced Institute of Science and Technology (KAIST) and later at Tokyo University of Agriculture and Technology from March 1996 to March 1999. From 1999 to 2004, he was a research staff member and later a senior researcher at the Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan. His research interests include pattern recognition, image processing, neural networks, machine learning, and especially their applications to character recognition and document analysis. He has published over 300 technical papers in prestigious international journals and conferences. He is an associate editor-in-chief of Pattern Recognition and is on the editorial board of Image and Vision Computing, International Journal on Document Analysis and Recognition, and Cognitive Computation. He is a fellow of CAAI, IEEE and IAPR.

APPENDIX A

TABLE XI
NODE FEATURES EXTRACTED FROM STROKE s_i.

#	Feature description
1	Trajectory length of s_i
2	Area of the convex hull of s_i
3	Duration of the stroke
4	Ratio of the principal axis of s_i
5	Rectangularity of the minimum area bounding rectangle of s_i
6	Circular variance of points of s_i around its centroid
7	Normalized centroid offset along the principal axis
8	Ratio between first-to-last point distance and trajectory length
9	Accumulated curvature
10	Accumulated squared perpendicularity
11	Accumulated signed perpendicularity
12	Width of s_i, normalized by the median stroke height in the document
13	Height of s_i, normalized by the median stroke height in the document
14	Average of the distances from s_i to time neighbors
15	Standard deviation of the distances from s_i to time neighbors
16	Average of lengths of time neighbors
17	Standard deviation of lengths of time neighbors
18	Average of the distances from s_i to space neighbors
19	Standard deviation of the distances from s_i to space neighbors
20	Average of lengths of space neighbors
21	Standard deviation of lengths of space neighbors
22-25	Normalized coordinates of the bounding box
26-27	Normalized centroid coordinates of s_i

TABLE XII
EDGE FEATURES EXTRACTED FROM PAIRWISE STROKE s_i, s_j.

#	Feature description
1	Minimum distance between 2 strokes
2	Minimum distance between the endpoints of 2 strokes
3	Maximum distance between the endpoints of 2 strokes
4	Distance between the centers of the 2 bounding boxes of 2 strokes
5	Horizontal distances between the centroids of 2 strokes
6	Vertical distances between the centroids of 2 strokes
7	Off-stroke distance between 2 strokes
8	Off-stroke distance projected on X and Y axes
9	Temporal distance between 2 strokes
10	Ratio of off-stroke distance to temporal distance
11	Ratio of off-stroke distance to projected on X; Y axes to temporal distance
12	Ratio of area of the largest bounding box of 2 strokes to that of their union
13	Ratio of widths of the bounding boxes of 2 strokes
14	Ratio of heights of the bounding boxes of 2 strokes
15	Ratio of diagonals of the bounding boxes of 2 strokes
16	Ratio of areas of the bounding boxes of 2 strokes
17	Ratio of lengths of 2 strokes
18	Ratio of durations of 2 strokes
19	Ratio of curvatures of 2 strokes

APPENDIX B

TABLE XIII
NETWORK HYPERPARAMETERS.

Hyperparameters	FC_A	FC_B	FA	CASIA-OHFC
Temporal neighbor K_T	1	1	1	1
Spatial neighbor K_S	5	5	4	7
Backbone node layers	4	4	4	4
Node layers in NC branch	3	3	3	3
Edge layers	4	4	3	3
Embedding layers in NE branch	3	4	4	4
Input dims for node layer	27	27	27	27
Input dims for edge layer	19	19	19	19
Hidden units in node layer	48	32	32	32
Hidden units in edge layer	60	35	35	35
Node attention heads	8	8	6	8
Node attention heads in output	2	2	2	2
Initial learning rate	0.005	0.005	0.003	0.01
Decay rate	0.1	0.1	0.1	0.1
Batch size	8	8	4	32
Patience	20	20	10	20
Training epochs	200	200	200	400
Dropout rate	0.1	0.1	0.1	0.1
Positive edge threshold T_+	0.99	0.99	0.90	0.90