Multigraph Transformer for Free-Hand Sketch Recognition

Peng Xu, Chaitanya K. Joshi, and Xavier Bresson

***Abstract*— Learning meaningful representations of free-hand sketches remains a challenging task given the signal sparsity and the high-level abstraction of sketches. Existing techniques have focused on exploiting either the static nature of sketches with convolutional neural networks (CNNs) or the temporal sequential property with recurrent neural networks (RNNs). In this work, we propose a new representation of sketches as multiple sparsely connected graphs. We design a novel graph neural network (GNN), the multigraph transformer (MGT), for learning representations of sketches from multiple graphs, which simultaneously capture global and local geometric stroke structures as well as temporal information. We report extensive numerical experiments on a sketch recognition task to demonstrate the performance of the proposed approach. Particularly, MGT applied on 414k sketches from Google QuickDraw: 1) achieves a small recognition gap to the CNN-based performance upper bound (72.80% versus 74.22%) and infers faster than the CNN competitors and 2) outperforms all RNN-based models by a significant margin. To the best of our knowledge, this is the first work proposing to represent sketches as graphs and apply GNNs for sketch recognition. Code and trained models are available at https://github.com/PengBoXiangShang/multigraph_transformer.**

***Index Terms*— Free-hand sketch, graph neural network (GNN), graph representation of sketch, hand-drawn sketch, multigraph transformer (MGT), neural representation of sketch, sketch, sketch classification, sketch recognition, transformer.**

I. Introduction

FREE-HAND sketches are drawings made without the use of any instruments. Sketches are different from traditional images: they are formed of temporal sequences of strokes [1], [2], while images are static collections of pixels with dense color and texture patterns. Sketches capture high-level abstraction of visual objects with very sparse information compared to regular images, which makes the modeling of sketches unique and challenging.

The modern prevalence of touchscreen devices has led to a flourishing of sketch-related applications in recent years, including sketch recognition [3], [4], sketch scene understanding [5], sketch hashing [2], sketch-based image retrieval [6]–[11], sketch-related generation [1], [3], [12], [13], sketch self-supervised learning [14], [15], and so on.

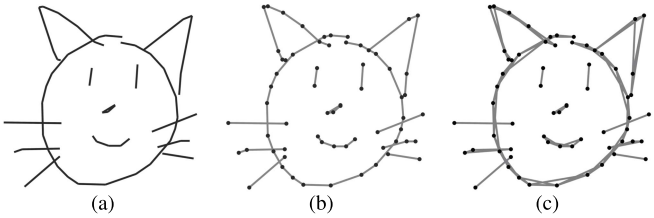


Fig. 1. Sketches can be seen as sets of curves and strokes, which are discretized by graphs. (a) Original sketch. (b) 1-hop connected. (c) 2-hop connected.

If we assume sketches to be 2-D static images, convolutional neural networks (CNNs) can be directly applied to sketches, such as "Sketch-a-Net" [16]. If we now suppose that sketches are ordered sequences of point coordinates, then recurrent neural networks (RNNs) can be used to recursively capture the temporal information, e.g., "SketchRNN" [1].

In this work, we introduce a new representation of sketches with *graphs*. We assume that sketches are sets of curves and strokes, which are discretized by a set of points representing the graph nodes. This view offers high flexibility to encode different sketch geometric properties as we can decide different connectivity structures between the node points. We use two types of graphs to represent sketches: intrastroke and extra-stroke graphs. Intrastroke graphs capture the local geometry of strokes, independently to each other, with for example 1-hop or 2-hop connected graphs, see Fig. 1. Extra-stroke graphs encode the global geometry and temporal information of strokes. Another advantage of using graphs is the freedom to choose the node features. For sketches, spatial, temporal, and semantic information is available with the stroke point coordinates, the ordering of points, and the pen state information, respectively. In summary, representing sketches with graphs offers a universal representation that can make use of global and local spatial sketch structures as well as temporal and semantic information.

To exploit these graph structures, we propose a new transformer [17] architecture that can use multiple sparsely connected graphs. It is worth reporting that a direct application of the original transformer model on the input spatio-temporal features provides poor results. We argue that the issue comes from the graph structure in the original transformer, which is a fully connected graph. Although fully connected word graphs work impressively for natural language processing (NLP), where the underlying word representations

Manuscript received July 14, 2020; revised December 24, 2020; accepted March 24, 2021. The work of Xavier Bresson was supported in part by the National Research Foundation Singapore (NRF) Fellowship under Grant NRFF2017-10. *(Corresponding author: Peng Xu.)*

Peng Xu and Xavier Bresson are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore (e-mail: peng.xu@ntu.edu.sg).

Chaitanya K. Joshi is with the Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3069230.

Digital Object Identifier 10.1109/TNNLS.2021.3069230

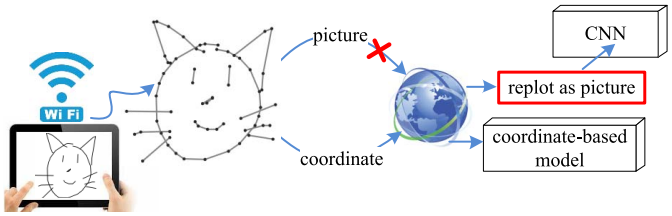


Fig. 2. In sketch-based human-computer interaction scenarios, it is time-consuming to render and transfer pictures of sketches. Solely transferring stroke coordinates leads to real-time applications.

themselves contain rich information, such dense graph structures provide poor innate priors/inductive bias [18] for 2-D sketch tasks. Transformers require sketch-specific design coming from geometric structures. This led us to naturally extend transformers to multiple arbitrary graph structures. Moreover, graphs provide more robustness to handle noisy and style-changing sketches as they focus on the geometry of stokes and not on the specific distribution of points.

Another advantage of using domain-specific graphs is to leverage the sparsity property of discretized sketches. Observe that intrastroke and extra-stroke graphs are *highly sparse* adjacency matrices. In practical sketch-based human-computer interaction scenarios, it is time-consuming to directly transfer the original sketch picture from user touch-screen devices to the back-end servers. To ensure real-time applications, transferring the stroke coordinates as a character string would be more beneficial, see Fig. 2.

Our **main contributions** can be summarized as follows.

1) We propose to model sketches as sparsely connected graphs, which are flexible to encode local and global geometric sketch structures. To the best of our knowledge, it is the first time that graphs are proposed for representing sketches.
2) We introduce a novel transformer architecture that can handle multiple arbitrary graphs. Using intrastroke and extra-stroke graphs, the proposed multigraph transformer (MGT) learns both local and global patterns along with subcomponents of sketches.
3) Numerical experiments demonstrate the performances of our model. MGT significantly outperforms RNN-based models and achieves a small recognition gap to CNN-based architectures. Moreover, our MGT infers faster than the strongest CNN baseline (see details in Section IV-B7). This is promising for real-time sketch-based human-computer interaction systems. Note that for sketch recognition, CNNs are the performance upper bound of coordinate-based models that involve truncating coordinate sequences, e.g., RNN or transformer-based architectures.
4) This MGT model is agnostic to graph domains and can be used beyond sketch applications. We transfer our MGT idea to conduct relation extraction RE) on an NLP benchmark, i.e., SemEval-2010 task 8, outperforming the state-of-the-art CNNs by a clear margin [prediction acc.: ours (89.45%) versus CNN (86.60%)].

From the perspective of representing sketches and the practical sketch-oriented applications, the **main advantages** of our proposed MGT against CNN are:

1) Our model can be used for real-time sketch-based human-computer interaction (HCI) systems because it does not need to render and transfer sketch pictures (see Fig. 2).
2) Our MGT is faster than the strongest CNN baseline, i.e., Inception V3. This will be demonstrated by experiments in Section IV-B7.
3) As demonstrated in [19], coordinate-based models (e.g., our MGT, RNNs) are more suitable than CNNs for the stroke-level tasks, e.g., perceptual grouping, stroke-grained segmentation.
4) Our MGT takes the stroke sequence as input, thus it can handle the tasks that need to understand the logic and timing patterns of the sketching process, e.g., stroke-by-stroke generation, stroke-level abstraction. Specifically, stroke-by-stroke sketch generation needs models to learn the temporal information because the models need to decide the order of stroke generation. However, as discussed in [20], CNNs intrinsically fail to work for these tasks.

The rest of this article is organized as follows: Section II briefly summarizes related work. Section III describes our proposed MGT. Experimental results and discussion are presented in Section IV, followed by a Conclusion Section V. Finally, we discuss our future work in Section VI.

II. Related Work

A. Neural Network Architectures for Sketches

CNNs are a common choice for feature extraction from sketches. "Sketch-a-Net" [16] is a representative CNN-based model having a sketch-specific architecture. It was directly inspired from AlexNet [21] with larger first layer filters, no layer normalization, larger pooling sizes, and high dropout. Song *et al.* [22] further improved Sketch-a-Net by adding spatial-semantic attention layers. "SketchRNN" [1] is a seminal work to model temporal stroke sequences with RNNs, by taking the key point coordinates of stroke as input. "SketchRNN" reminds the researchers that sketching is a dynamic process so that the temporal patterns of sketching strokes should also be considered.

Recently, some CNN-RNN hybrid architectures have been proposed for sketches, e.g., dual-branch networks [2], [23], cascaded networks [4], [24]. For a more detailed summary and comparison for sketch representing networks, please check a comprehensive survey [20].

Essentially, the aforementioned CNN- or RNN-based network architectures model sketch in Euclidean space. How to model sketch in non-Euclidean spaces is an interesting question. In this work, we propose a novel graph neural network (GNN) architecture for learning sketch representations from multiple sparse graphs, combining both stroke geometry and temporal order.

B. Graph Neural Networks

GNNs [25]–[32] aim to generalize neural networks to non-Euclidean domains such as graphs and manifolds. GNNs iteratively build representations of graphs through recursive

neighborhood aggregation (or message passing), where each graph node gathers features from its neighbors to represent local graph structure. Recently, GNNs have been widely applied for various domains, e.g., vision, language. However, sketch-oriented GNNs are still under-studied.

C. Transformers

The transformer architecture [17], originally proposed as a powerful and scalable alternative to RNNs, has been widely adopted in the NLP community for tasks such as machine translation [33], [34], language modeling [35], [36], and question-answering [37], [38].

Transformers for NLP can be regarded as GNNs which use self-attention [39], [40] for neighborhood aggregation on fully connected word graphs [41]. However, GNNs and Transformers perform poorly when sketches are modeled as fully connected graphs. This work advocates for the injection of inductive bias into transformers through domain-specific graph structures.

III. Method

A. Notation

We assume that the training data set D consists of N labeled sketches: $D = \{(\mathbf{X}_n, z_n)\}_{n=1}^{N}$. Each sketch $\mathbf{X}_n$ has a class label z_n, and can be formulated as a S-step sequence $[\mathbf{C}_n, \mathbf{f}_n, \mathbf{p}] \in \mathbb{R}^{S\times 4}$. $\mathbf{C}_n = \{(x_n^s, y_n^s)\}_{s=1}^{S} \in \mathbb{R}^{S\times 2}$ is the coordinate sequence of the sketch points $\mathbf{X}_n$. All sketch point coordinates have been uniformly scaled to $x_n^s, y_n^s \in [0,\ 256]^2$. If the true length of $\mathbf{C}_n$ is shorter than S then the vector $[-1, -1]$ is used for padding. Flag bit vector $\mathbf{f}_n \in \{f_1, f_2, f_3\}^{S\times 1}$ is a ternary integer vector that denotes the pen state sequence corresponding to each point of $\mathbf{X}_n$. It is defined as follows: f_1 if the point (x_n^s, y_n^s) is a starting or ongoing point of a stroke, f_2 if the point is the ending point of a stroke, and f_3 for a padding point. Vector $\mathbf{p} = [0, 1, 2, \ldots, S-1]^T$ is a positional encoding vector that represents the temporal position of the points in each sketch $\mathbf{X}_n$.

Given D, we aim to model $\mathbf{X}_n$ as multiple sparsely connected graphs and learn a deep embedding space, where the high-level semantic tasks can be conducted, e.g., sketch recognition.

B. Multimodal Input Layer

Given a sketch $\mathbf{X}_n$, we model its S stroke points as S nodes of a graph. Each node has three features: 1) $\mathbf{C}_n^s$ is the spatial positional information of the current stroke point s; 2) $\mathbf{f}_n^s$ is the pen state of the current stroke point. This information helps to identify the stroke points belonging to the same stroke; and 3) $\mathbf{p}^s$ is the temporal information of the current stroke point. As sketching is a dynamic process, it is important to use temporal information.

The complete model architecture for our MGT is presented in Fig. 3. Let us start by describing the input layer. The final vector at node s of the multimodal input layer is defined as

$$\left(\mathbf{h}_n^s\right)^{(l=0)} = \mathcal{C}\left(\mathcal{E}_1\left(\mathbf{C}_n^s\right), \mathcal{E}_2\left(\mathbf{f}_n^s\right), \mathcal{E}_2(\mathbf{p}^s)\right) \quad (1)$$

where $\mathcal{E}_1(\mathbf{C}_n^s)$ is the embedding of $\mathbf{C}_n^s$ with a linear layer of size $2 \times \hat{d}$, $\mathcal{E}_2(\mathbf{f}_n^s)$ and $\mathcal{E}_2(\mathbf{p}^s)$ are the embeddings of the flag bit $\mathbf{f}_n^s$ (three discrete values) and the position encoding $\mathbf{p}^s$ (S discrete values) from an embedding dictionary of size $(S+3) \times \hat{d}$, and $\mathcal{C}(\cdot,\cdot)$ is the concatenation operator. The node vector $(\mathbf{h}_n^s)^{(l=0)}$ has dimension $d = 3\hat{d}$. The design of the input layer was selected after extensive ablation studies, which are described in Section IV-B5.

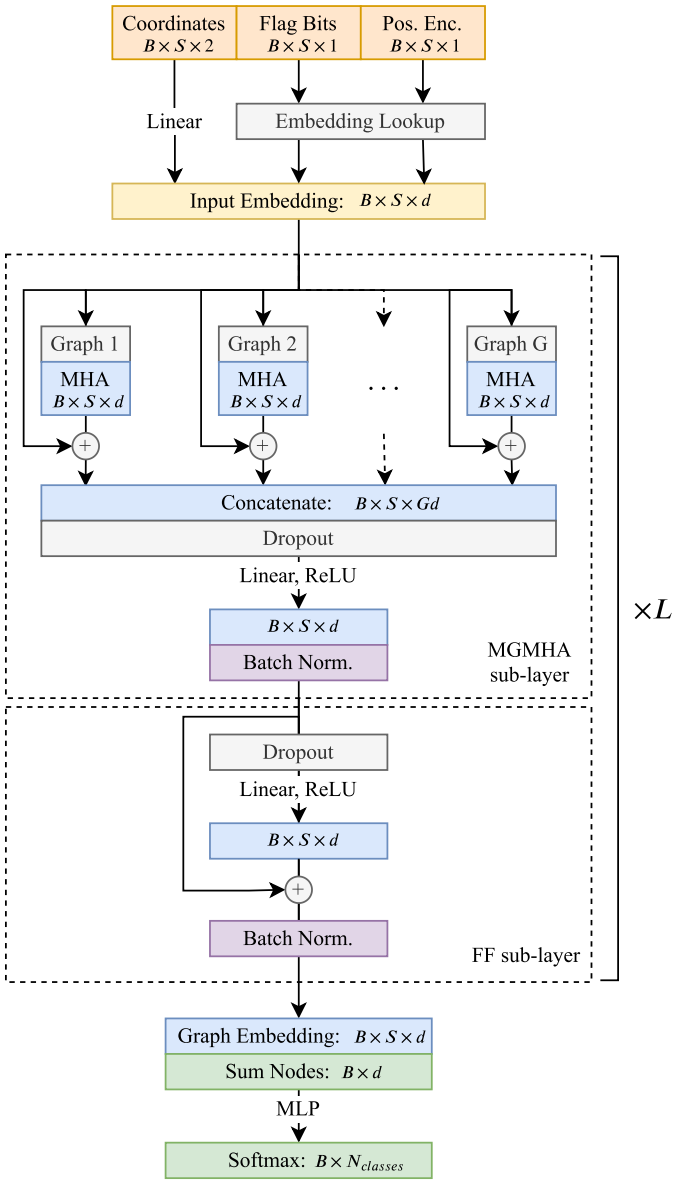


Fig. 3. MGT architecture. Each MGT layer is composed of 1) a MGMHA sublayer and 2) a position-wise fully connected FF sublayer. See details in text. "B" denotes batch size.

C. Multigraph Transformer

The initial node embedding $(\mathbf{h}_n^s)^{(l=0)}$ is updated by stacking L MGT layers (7). Let us describe all layers.

1) Graph Attention Layer: Let $\mathbf{A}$ be a graph adjacency matrix of size $S \times S$ and $\mathbf{Q} \in \mathbb{R}^{S\times d_q}$, $\mathbf{K} \in \mathbb{R}^{S\times d_k}$, $\mathbf{V} \in \mathbb{R}^{S\times d_v}$ be the query, key, and value matrices. We define a graph attention layer as

$$\text{GraphAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{A}) = \mathbf{A} \odot \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (2)$$

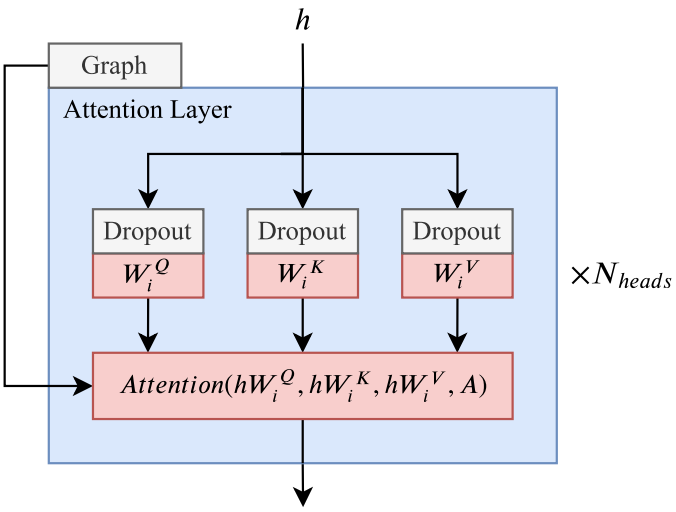


Fig. 4. MHA layer, consisting of several graph attention layers in parallel.

where $\odot$ is the Hadamard product. We simply weight the "Scaled Dot-Product Attention" [17] with the graph edge weights. We set $d_q = d_k = d_v = d/I$, where I is the number of attention heads.

2) Multihead Attention Layer: We aggregate the graph attentions with multiple heads

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{A}) = \mathcal{C}(\text{head}_1, \ldots, \text{head}_I)\mathbf{W}^O \quad (3)$$

where $\mathbf{W}^O \in \mathbb{R}^{Id_v \times d}$ and each attention head is computed with the graph attention layer (2)

$$\text{head}_i = \text{GraphAttention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V, \mathbf{A}\right) \quad (4)$$

where $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_q}$, $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$, and $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$. We add dropout [42] before the linear projections of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. An illustration of the multi-head attention (MHA) layer is presented in Fig. 4.

3) Multigraph Multihead Attention Layer: Given a set of adjacency graph matrices $\{\mathbf{A}_g\}_{g=1}^G$, we can concatenate multi-head attention layers

$$\begin{aligned}\text{MultiGraphMultiHeadAttention}&\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \{\mathbf{A}_g\}_{g=1}^G\right)\\ &= \text{ReLU}(\mathcal{C}(\text{ghead}_1, \ldots, \text{ghead}_G)\mathbf{W}^{\tilde{O}}) \quad (5)\end{aligned}$$

where $\mathbf{W}^{\tilde{O}} \in \mathbb{R}^{Gd \times d}$ and each MHA layer is computed with (3)

$$\text{ghead}_g = \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{A}_g). \quad (6)$$

4) Multigraph Transformer Layer: The MGT at layer l for node s is defined as

$$\begin{aligned}\left(\mathbf{h}_n^s\right)^{(l)} &= \text{MGT}\left(\left(\mathbf{h}_n\right)^{(l-1)}\right)\\ &= \hat{\mathbf{h}}_n^s + \text{FF}^{(l)}\left(\hat{\mathbf{h}}_n^s\right) \quad (7)\end{aligned}$$

where the intermediate feature representation $\hat{\mathbf{h}}_n^s$ is defined as

$$\hat{\mathbf{h}}_n^s = \left(\text{MGMHA}_n^s\right)^{(l)}\left(\left(\mathbf{h}_n^1\right)^{(l-1)}, \ldots, \left(\mathbf{h}_n^S\right)^{(l-1)}\right). \quad (8)$$

The MGT layer is thus composed of 1) a multigraph multihead attention (MGMHA) sublayer (5) and 2) a position-wise fully connected feed-forward (FF) sublayer. Each MHA sublayer (6) and FF (7) has residual-connection [43] and batch normalization [44]. See Fig. 3 for an illustration.

D. Sketch Embedding and Classification Layer

Given a sketch $\mathbf{X}_n$ with t_n key points, its continuous representation $\mathbf{h}_n$ is simply given by the sum over all its node features from the last MGT layer

$$\mathbf{h}_n = \sum_{s=1}^{t_n} \left(\mathbf{h}_n^s\right)^{(L)}. \quad (9)$$

Finally, we use a multilayer perceptron (MLP) to classify the sketch representation $\mathbf{h}_n$, see Fig. 3.

E. Sketch-Specific Graphs

In this section, we discuss the graph structures we used in our graph transformer (GT) layers. We considered two types of graphs, which capture local and global geometric sketch structures.

The first class of graphs focuses on representing the local geometry of individual strokes. We choose K-hop graphs to describe the local geometry of strokes. The intrastroke adjacency matrix is defined as follows:

$$\mathbf{A}_{n,ij}^{K-\text{hop}} = \begin{cases} 1, & \text{if } j \in \mathcal{N}_i^{K-\text{hop}} \text{ and } j \in \text{global}(i) \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

where $\mathcal{N}_i^{K-\text{hop}}$ is the K-hop neighborhood of node i and $\text{global}(i)$ is the stroke of node i.

The second class of graphs captures the global and temporal relationships between the strokes composing the whole sketch. We define the extra-stroke adjacency matrix as follows:

$$\mathbf{A}_{n,ij}^{\text{global}} = \begin{cases} 1, & \text{if } |i-j| = 1 \text{ and } \text{global}(i) \neq \text{global}(j) \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

This graph will force the network to pay attention between two points belonging to two distinct strokes but consecutive in time, thus allowing the model to understand the relative arrangement of strokes.

IV. Experiments

A. Experimental Setting

In this section, we detail our experimental settings.

1) Data Set and Preprocessing: Google QuickDraw [1][1,2] is the largest available sketch data set containing 50 Million sketches as simplified stroke key points in temporal order, sampled using the Ramer–Douglas–Peucker algorithm after uniformly scaling image coordinates within 0 to 256. Unlike smaller crowd-sourced sketch data sets, e.g., TU-Berlin [45], QuickDraw samples were collected via an international online game where users have only 20 s to sketch objects from 345 classes, such as cats, dogs, clocks, and so on. Thus, sketch classification on QuickDraw not only involves a diversity of drawing styles but can also be highly abstract and noisy, making it a challenging and practical testbed for comparing

[1]https://quickdraw.withgoogle.com/data
[2]https://github.com/googlecreativelab/quickdraw-dataset

TABLE I
SUMMARY STATISTICS FOR OUR SUBSET OF QUICKDRAW

Set	# Samples	# Truncated (ratio)	# Key Points			
			max	min	mean	std
Training	345,000	11788 (3.42%)	100	2	43.26	21.85
Validation	34,500	1218 (3.53%)	100	2	43.24	21.89
Test	34,500	1235 (3.58%)	100	2	43.20	21.93

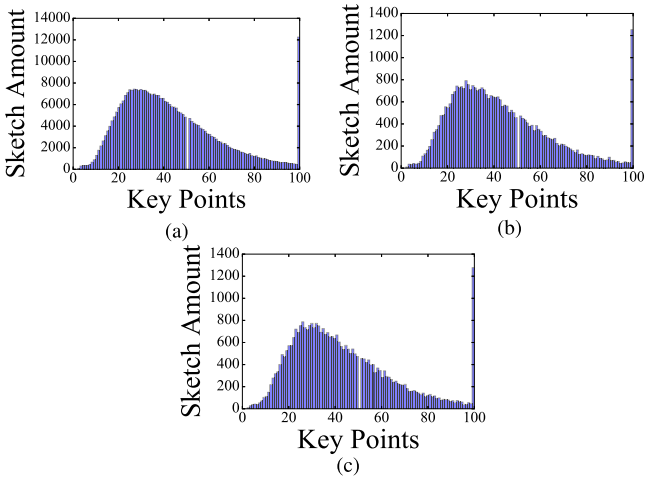


Fig. 5. Histograms of key points per sketch for our subset of QuickDraw. The sharp spike at 100 key points is due to truncation. (a) Training. (b) Validation. (c) Test.

the effectiveness of various neural network architectures. Following recent practices [2], [11], we create random training, validation, and test sets from the full data set by sampling 1000, 100, and 100 sketches, respectively, from each of the 345 categories in QuickDraw. Although the transformer architecture is able to handle sketches with any finite number of key points, following [2], we truncate or pad all samples to a uniform length of 100 key points/steps to facilitate efficient training of RNN and GNN-based models, to save parameters and training time. We provide summary statistics for our training, validation and test sets in Table I, and histograms visualizing the key points per sketch are shown in Fig. 5.

2) Evaluation Metrics: Our evaluation metric for sketch recognition is "top K accuracy," the proportion of samples whose true class is in the top K model predictions, for values $k = 1, 5, 10$. (Note that acc. at $k = 1.0$ means 100%).

3) Implementation Details: For fair comparison under similar hardware conditions, all experiments were implemented in PyTorch [46][3] and run on one Nvidia 1080Ti GPU. For transformer models, we use the following hyperparameter values: $S = 100$, $L = 4$, $\hat{d} = 128$, $G = 3$ ($\mathbf{A}^{1-\text{hop}}, \mathbf{A}^{2-\text{hop}}, \mathbf{A}^{\text{global}}$), and $I = 8$ (per graph) for our Base model (and $\hat{d} = 256$ for our Large model). Our FF sublayer is a d-dimensional linear layer ($d = 3\hat{d}$) followed by rectified linear unit (ReLU) [47] and dropout. The MLP Classifier consists of two $4\hat{d}$-dimensional linear layers with ReLU and dropout, followed by a 345-dimensional linear projection representing logits over the 345 categories in QuickDraw. We train all models by minimizing the softmax cross-entropy loss using the Adam [48] optimizer for 100 epochs. We use an initial learning rate of $5e{-}5$ and multiply by a factor 0.7 every 10 epochs. We use an early stopping strategy (with the hyperparameter "patience" of 10 epochs) for selecting the final model, and the checkpoint with the highest validation performance is chosen to report test performance.

4) Baselines:

(i) From the perspective of coordinate-based sketch recognition, RNN models are a simple-yet-effective baseline. Following Xu *et al.* [2], we design several bidirectional long short-term memory (LSTM) [49] and gate recurrent unit (GRU) [50] models at increasing parameter budgets comparable with MGT. The final RNN states are concatenated and passed to the MLP classifier described previously. We use batch size 256, initial learning rate $1e-4$ and multiply by 0.9 every 10 epochs. We train models with both our multimodal input (Section III-B) as well as the 4-D input from [2].

(ii) Although converting sketch coordinates to images adds time overhead in practical settings and can be seen as auxiliary information, we compare MGT to various state-of-the-art CNN architectures. It is important to note that sketch sequences were truncated/padded for training both MGT and RNNs, hence, image-based CNNs stand as an upper bound in terms of performance. For Inception V3 [51] and MobileNet V2 [52], initial learning rate is $1e-3$ and multiplied by 0.5 every 10 epochs. For other CNN baselines, the initial learning rate and decay are configured following their original articles. For each model, we use the maximum possible batch size. Following standard practice in computer vision [43], [53], we employ early stopping based on observing overfitting in the validation loss, and select the checkpoint with the highest validation accuracy for evaluation on the test set.

(iii) To evaluate the effectiveness of the proposed GT layer, we compare it with popular GNN variants: the graph convolutional network (GCN) [29] and the graph attention network (GAT) [40].[4] All GNN models follow the same hyperparameter setup as Transformers ($L = 4, \hat{d} = 256$) and are augmented with residual connections and batch normalization for fair comparison, following [54]. Optimal hyperparameters and learning rate schedules are selected based on validation set performance.

B. Results

For a fair comparison with RNN and CNN baselines at various parameter budgets, we implement two configurations of MGT: Base (10M parameters) and Large (40M parameters). In addition, we perform several ablation studies to evaluate the effectiveness of our multigraph architecture and our sketch-specific input design. Our main results are presented in Table II.

1) Comparison With RNN Baselines: We trained RNNs at various parameter budgets, and present results for the best performing bidirectional LSTM and GRU models in Table II.

[3]https://pytorch.org/

[4]For GAT, we use the same scaled dot-product attention mechanism as GT for efficiency.

TABLE II

TEST SET PERFORMANCE OF MGT VERSUS THE STATE-OF-THE-ART RNN AND CNN ARCHITECTURES. THE FIRST/SECOND/THIRD BEST RESULTS PER COLUMN ARE INDICATED IN **RED**/BLUE/MAGENTA

Network	Configurations	Recognition Accuracy			Parameter Amount
		acc.@1	acc.@5	acc.@10	
Bi-directional LSTM #1	4D Input, $\hat{d}=256, L=4, Dropout_{LSTM}=0.5, Dropout_{MLP}=0.15$	0.6665	0.8820	0.9189	5,553,241
Bi-directional LSTM #2	4D Input, $\hat{d}=256, L=5, Dropout_{LSTM}=0.5, Dropout_{MLP}=0.15$	0.6524	0.8697	0.9133	7,130,201
Bi-directional GRU	4D Input, $\hat{d}=256, L=5, Dropout_{GRU}=0.5, Dropout_{MLP}=0.15$	0.6768	0.8854	0.9234	5,419,097
AlexNet [21]	Standard architecture and configurations	0.6808	0.8847	0.9203	58,417,305
VGG-11 [55]		0.6743	0.8814	0.9191	130,179,801
Inception V3 [51]		0.7422	0.9189	0.9437	25,315,474
ResNet-18 [43]		0.7031	0.9030	0.9351	11,353,497
ResNet-34 [43]		0.7009	0.9010	0.9347	21,461,657
ResNet-152 [43]		0.6924	0.8973	0.9312	58,850,713
DenseNet-201 [53]		0.7050	0.9013	0.9331	18,755,673
MobileNet V2 [52]		0.7310	0.9161	0.9429	2,665,817
SCNet [56]		0.7123	0.9026	0.9351	24,222,489
ResNet-102+BSConv-U [57]		0.7172	0.9037	0.9334	7,029,791
Vanilla Transformer [17]	$\hat{d}=256, L=4, I=8, Dropout=0.1$, Fully-connected graph	0.5249	0.7802	0.8486	14,029,401
MGT (Base)	$\hat{d}=128, L=4, I=24, Dropout=0.1$, $\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{2-hop}}, \mathbf{A}^{\text{global}}$ graphs	0.7070	0.9030	0.9351	10,096,601
MGT (Large)	$\hat{d}=256, L=4, I=24, Dropout=0.25$, $\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{2-hop}}, \mathbf{A}^{\text{global}}$ graphs	0.7280	0.9106	0.9387	39,984,729

TABLE III

ABLATION STUDY FOR MULTIGRAPH ARCHITECTURE OF MGT. GT DENOTES SINGLE-GRAPH VARIANTS OF MGT. THE FIRST/SECOND BEST RESULTS PER COLUMN ARE INDICATED IN **RED**/BLUE. || DENOTES THE LOGICAL UNION OPERATION

Network	Configurations						Recognition Accuracy			Parameter Amount
	G	Graph Structure	I_{total}	$\hat{d}$	L	Dropout	acc.@1	acc.@5	acc.@10	
GT #1	1	Fully-connected (*vanilla*)	8	256	4	0.10	0.5249	0.7802	0.8486	14,029,401
GT #2	1	Intra-stroke Fully-connected	8	256	4	0.10	0.6487	0.8697	0.9151	14,029,401
GT #3	1	Random (10%)	8	256	4	0.10	0.5271	0.7890	0.8589	14,029,401
GT #4	1	Random (20%)	8	256	4	0.10	0.5352	0.7945	0.8617	14,029,401
GT #5	1	Random (30%)	8	256	4	0.10	0.5322	0.7917	0.8588	14,029,401
GT #6	1	$\mathbf{A}^{\text{1-hop}}$	8	256	4	0.10	0.7023	0.8974	0.9303	14,029,401
GT #7	1	$\mathbf{A}^{\text{2-hop}}$	8	256	4	0.10	0.7082	0.8999	0.9336	14,029,401
GT #8	1	$\mathbf{A}^{\text{3-hop}}$	8	256	4	0.10	0.7028	0.8991	0.9327	14,029,401
GT #9	1	$\mathbf{A}^{\text{global}}$	8	256	4	0.10	0.5488	0.8009	0.8659	14,029,401
GT #10	1	$\mathbf{A}^{\text{1-hop}}\|\|\mathbf{A}^{\text{2-hop}}\|\|\mathbf{A}^{\text{global}}$	8	256	4	0.10	0.7057	0.9021	0.9346	14,029,401
MGT #11	2	$\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{2-hop}}$	16	256	4	0.25	0.7149	0.9049	0.9361	28,188,249
MGT #12	2	$\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{global}}$	16	256	4	0.25	0.7111	0.9041	0.9355	28,188,249
MGT #13	2	$\mathbf{A}^{\text{2-hop}}, \mathbf{A}^{\text{global}}$	16	256	4	0.25	0.7237	0.9102	**0.9400**	28,188,249
MGT #14	3	$\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{1-hop}}$	24	256	4	0.25	0.7077	0.9020	0.9340	39,984,729
MGT #15	3	$\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{2-hop}}, \mathbf{A}^{\text{3-hop}}$	24	256	4	0.25	0.7156	0.9066	0.9365	39,984,729
MGT #16	3	$\mathbf{A}^{\text{1-hop}}\|\|\mathbf{A}^{\text{2-hop}}\|\|\mathbf{A}^{\text{global}}$	24	256	4	0.25	0.7126	0.9051	0.9372	39,984,729
MGT #17	3	$\mathbf{A}^{\text{1-hop}}, \mathbf{A}^{\text{2-hop}}, \mathbf{A}^{\text{global}}$	24	256	4	0.25	**0.7280**	**0.9106**	0.9387	39,984,729

(i) MGT outperforms both LSTM and GRU baselines by a significant margin (by 3% acc. at 1 for Base, 5% for Large), indicating that both geometry and temporal order of strokes are important for sketch representation learning.

(ii) Training larger RNNs is harder to converge, leading to degrading performance, e.g., GRUs outperform deeper LSTMs by 2%.

These results are not surprising: RNNs are notoriously hard to train at scale [58], while Transformer performance is known to improve with scale, even with billions of model parameters [59].

2) Comparison With CNN Baselines: Table II also presents performance of several state-of-the-art CNN architectures for computer vision.

(i) Inception V3 [51] and MobileNet V2 [52] are the best performing CNN architectures. Our MGT Base has competitive or better recognition accuracy than all other baselines: AlexNet [21], VGG-11 [55], ResNet models [43], and DenseNet-201 [53].

(ii) MGT Large has a small performance gap to Inception V3 and MobileNet V2 (i.e., 72.80% acc. at 1 versus 74.22%, 72.80% acc. at 1 versus 73.10%) and outperforms all other CNN architectures by almost 2%.

(iii) Somewhat counter-intuitively, shallow networks (Inception V3, MobileNet V2) outperform deeper networks (ResNet-152, Densenet-201) by almost 2%. This result highlights that CNNs designed for images with dense colors and textures are un-suitable for sparse sketches.

Note that MobileNet V2 is specifically designed for fast inference on mobile phones and is not directly comparable in terms of model parameters.

TABLE IV

TEST SET PERFORMANCE OF GT VERSUS OTHER GNN VARIANTS. THE FIRST/SECOND BEST RESULTS PER COLUMN ARE INDICATED IN **RED**/BLUE

Network	Graph Structure	Recognition Accuracy			Parameter Amount
		acc.@1	acc.@5	acc.@10	
Graph Convolutional Networks (GCN) [29]	fully-connected	0.4098	0.7384	0.8213	6,948,441
	$\mathbf{A}^{1\text{-hop}}$	0.6800	0.8869	0.9224	
Graph Attention Networks (GAT) [40]	fully-connected	0.4098	0.6960	0.7897	11,660,889
	$\mathbf{A}^{1\text{-hop}}$	0.6977	0.8952	0.9298	
Graph Transformer (GT)	fully-connected	0.5242	0.7796	0.8465	14,029,401
	$\mathbf{A}^{1\text{-hop}}$	**0.7057**	**0.8992**	**0.9311**	
position-wise feed-forward	None	0.5296	0.7901	0.8576	4,586,073

TABLE V

ABLATION STUDY FOR MULTIMODAL INPUT FOR MGT (LARGE). NOTATIONS: "+" AND "$\mathcal{C}(\cdots)$" DENOTE "SUM" AND "CONCATENATE", RESPECTIVELY. THE FIRST/SECOND BEST RESULTS PER COLUMN ARE INDICATED IN **RED**/BLUE

Input Permutation	Recognition Accuracy		
	acc.@1	acc.@5	acc.@10
coordinate	0.6512	0.8735	0.9162
coordinate + flag bit	0.6568	0.8762	0.9176
coordinate + flag bit + position encoding	0.6600	0.8766	0.9182
$\mathcal{C}$(coordinate, flag bit)	0.7017	0.8996	0.9321
$\mathcal{C}$(coordinate, flag bit, position encoding)	**0.7280**	**0.9106**	**0.9387**
4D Input	0.6559	0.8758	0.9175
4D Input + position encoding	0.6606	0.8781	0.9190
$\mathcal{C}$(4D Input, position encoding)	0.7117	0.9048	0.9366

3) Ablations for Multigraph Architecture: We design several ablation studies to evaluate our sketch-specific multigraph architecture in Table III.

(i) We evaluate GTs trained on fully connected graphs, i.e., *vanilla* Transformers (GT #1), fully connected graphs within strokes (GT #2), as well as random graphs with 10%, 20%, and 30% connectivity (GT #3, #4, and #5, respectively). We compare their performance with GTs trained on sketch-specific graphs $\mathbf{A}^{1-\text{hop}}$ (GT #6), $\mathbf{A}^{2-\text{hop}}$ (GT #7), $\mathbf{A}^{3-\text{hop}}$ (GT #8), and $\mathbf{A}^{\text{global}}$ (GT #9). We find that *vanilla* transformers on fully connected (52.49% acc. at 1) and random graphs (52.71%, 53.52%, 53.22%) perform poorly compared to sketch-specific graph structures determined by domain expertise, such as fully connected stroke graphs (64.87%) and $\mathbf{A}^{1-\text{hop}}$ (70.23%). The superior performance of K-hop graphs suggests that transformers benefit from sparse graphs representing local sketch geometry. We also evaluate a combined sketch-specific graph structure, i.e., $\mathbf{A}^{1-\text{hop}}||\mathbf{A}^{2-\text{hop}}||\mathbf{A}^{\text{global}}$ (GT #10), where the graph connectivity is the logical union set of $\mathbf{A}^{1-\text{hop}}$, $\mathbf{A}^{2-\text{hop}}$, and $\mathbf{A}^{\text{global}}$. However, this structure fails to gain performance improvement over $\mathbf{A}^{1-\text{hop}}$, $\mathbf{A}^{2-\text{hop}}$, and $\mathbf{A}^{\text{global}}$, despite involving more domain knowledge.

(ii) We experiment with various permutations of graphs for multigraph models (MGT #11–#17). We find that using a 3-graph architecture (MGT #17) combining local sketch geometry ($\mathbf{A}^{1-\text{hop}}, \mathbf{A}^{2-\text{hop}}$) and global temporal relationships ($\mathbf{A}^{\text{global}}$) significantly boosts performance over 2-graph and 1-graph models (72.80% versus 72.37% for 2-graph and 70.82% for 1-graph). This result is interesting because using global graphs independently (GT #9) leads to comparatively poor performance (54.88%). In addition, we found that using diverse graphs (MGT #15, #17) is better than using the same graph (MGT #14). Comparing MGT #14 and MGT #6 further shows that performance gains are due to the multigraph architecture as opposed to more model parameters.

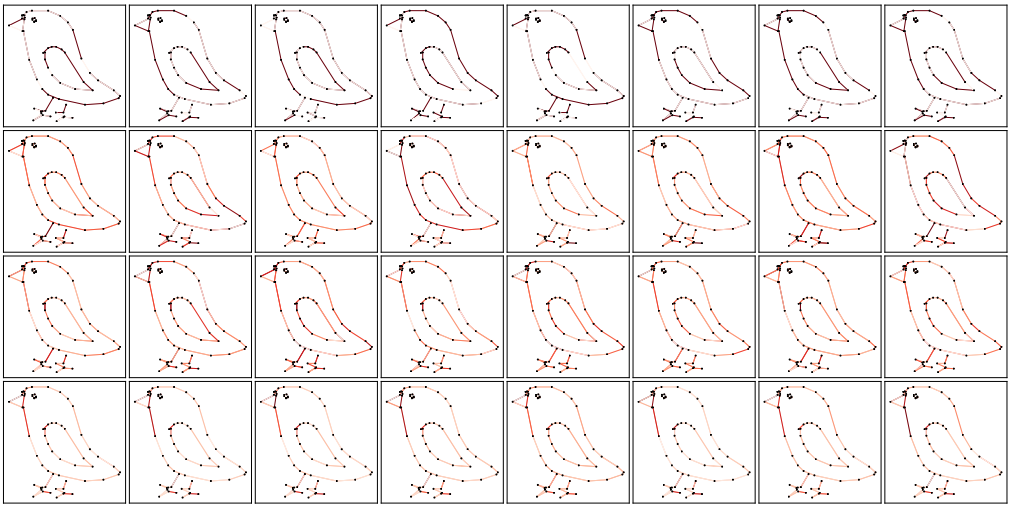
(iii) We also repeatedly input the adjacency matrix of GT #10 (i.e., $\mathbf{A}^{1-\text{hop}}||\mathbf{A}^{2-\text{hop}}||\mathbf{A}^{\text{global}}$) three times as the multiple graph structures to train our MGT (see MGT #16 in Table III). Compared with MGT #17, there is a clear performance gap (71.26% versus 72.80%). This further validates our idea of learning sketch representations through multiple separate graphs.

4) Comparison With GNN Baselines: In Table IV, we present performance of our GT model compared to GCNs [29] and GATs [40], two popular GNN variants.

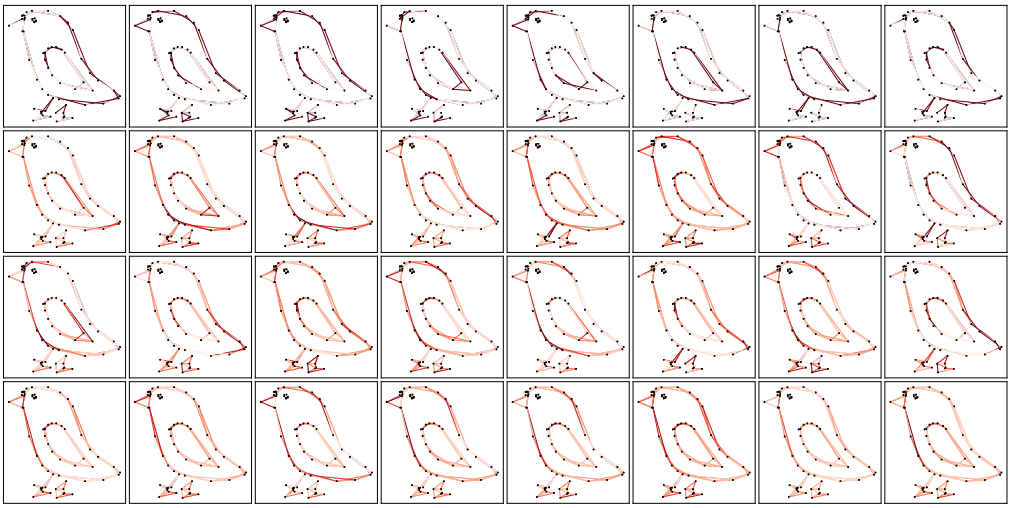
(i) We find that all models perform similarly on fully connected graphs. Using 1-hop graphs results in significant gains for all models, with transformer performing the best.

(ii) Interestingly, both GNNs on fully connected graphs are outperformed by a simple position-wise embedding method without any graph structure: each node undergoes four FF layers followed by summation and the MLP classifier.

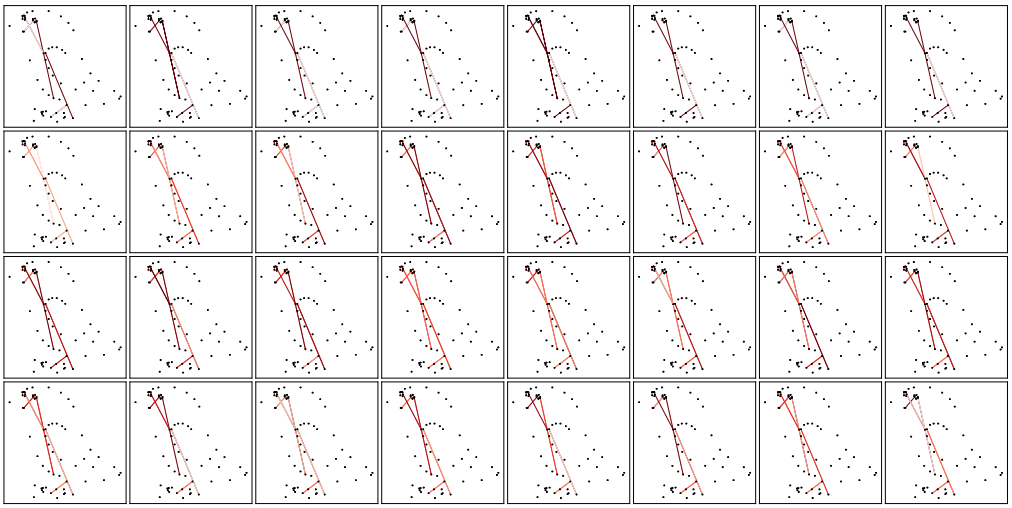
These results further highlight the importance of sketch-specific graph structures for the success of transformers.



(a)

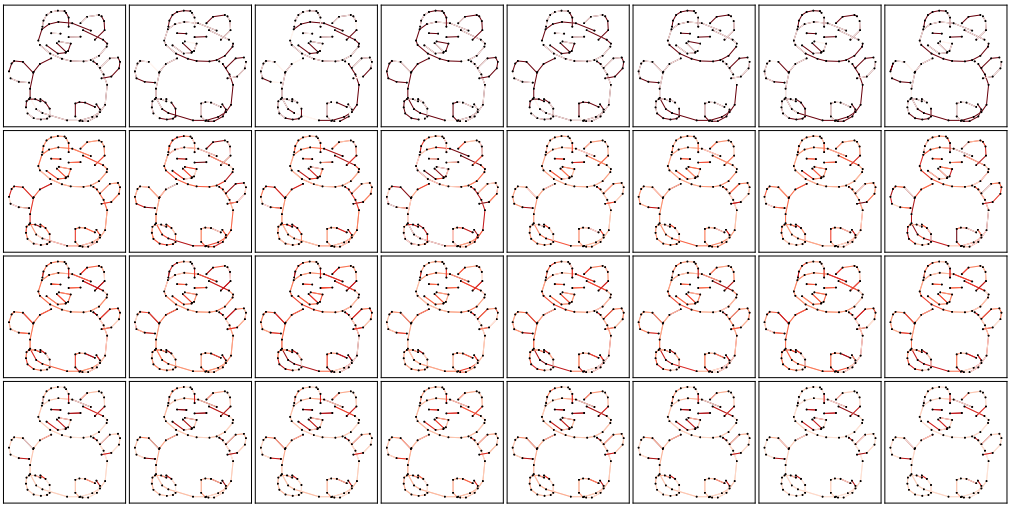


(b)

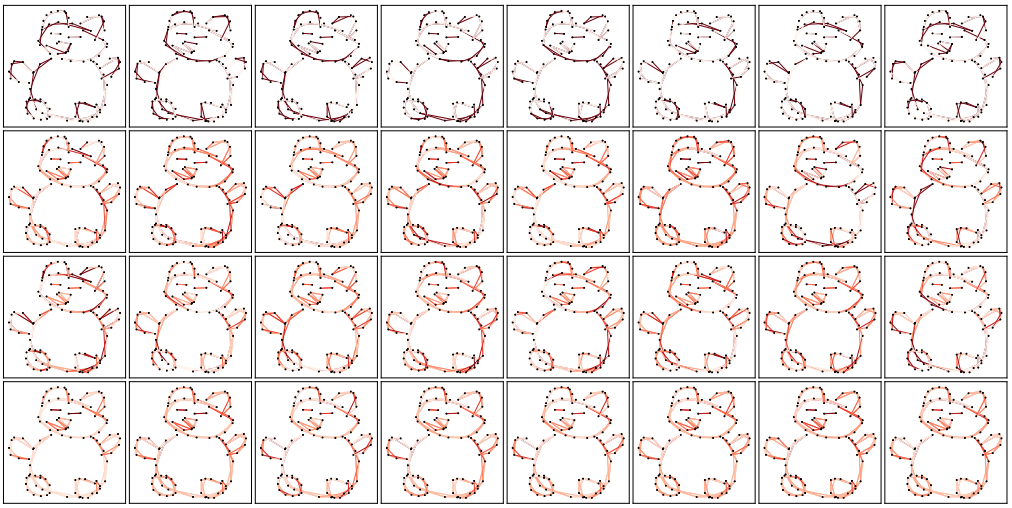


(c)

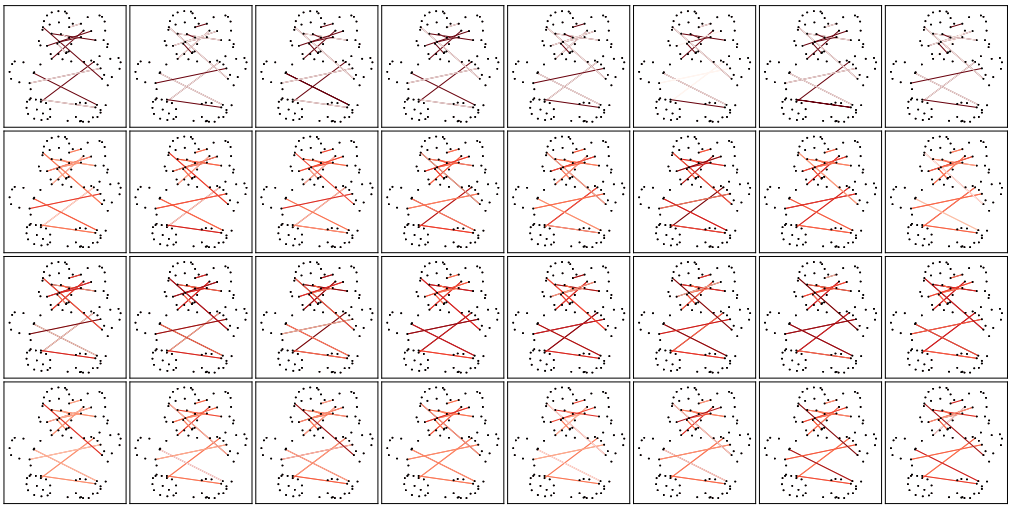
Fig. 6. Attention heads at each layer of MGT for a test set sample labeled *bird*. Each layer has $I = 8$ attention heads per graph in total. Darker reds indicate higher attention values. Best viewed in color. (a) $\mathbf{A}^{1-\text{hop}}$. (b) $\mathbf{A}^{2-\text{hop}}$. (c) $\mathbf{A}^{\text{global}}$.



(a)



(b)



(c)

Fig. 7. Attention heads at each layer of MGT for a test set sample labeled *teddy*. Each layer has $I = 8$ attention heads per graph in total. Darker reds indicate higher attention values. Best viewed in color. (a) $\mathbf{A}^{1-\text{hop}}$. (b) $\mathbf{A}^{2-\text{hop}}$. (c) $\mathbf{A}^{\text{global}}$.

TABLE VI

EVALUATION TIMING COMPARISON FOR MGT AND CNNS, AVERAGED OVER THREE RUNS. INFERENCE TIME IS THE TOTAL WALL CLOCK TIME FOR PERFORMING INFERENCE OVER 34 500 SKETCHES FROM THE TEST SET

Network	Inference Time	Parameter Amount
Inception V3	2.682±0.154s	25,315,474
MobileNet V2	1.423±0.070s	2,665,817
MGT (Base)	0.849±0.044s	10,096,601
MGT (Large)	0.843±0.048s	39,984,729

TABLE VII

PERFORMANCE COMPARISON ON THE RE TASK ON SEMEVAL-2010 TASK 8. THE FIRST/SECOND BEST RESULTS PER COLUMN ARE INDICATED IN **RED**/BLUE

Network	Graph Structure	Prediction Accuracy
textual CNN [60]	-	0.8660
Transformer [61]	fully-connected	0.8900
Transformer	fully- and partially-connected	**0.8945**

Our final models use the transformer layer, which implicitly includes the FF sublayer (7).

5) Ablations for Multimodal Input: In Table V, we experiment with various permutations of our sketch-specific multimodal input design. We aggregate information from spatial (coordinates), semantic (flag bits), and temporal (position encodings) modalities via summation (as in transformers for NLP) or concatenation.

(i) Effectively using all modalities is important for performance (e.g., "$\mathcal{C}$(coordinate, flag bit, position encoding)" outperforms "coordinate" and "$\mathcal{C}$(coordinate, flag bit)": 72.80% acc. at 1 versus 65.12%, 70.17%).

(ii) Concatenation works better than 4-D input as well as summation (e.g., "$\mathcal{C}$(coordinate, flag bit, position encoding)" outperforms "$\mathcal{C}$(4-D Input, position encoding)" and "coordinate + flag bit + position encoding": 72.80% versus 71.17%, 66.06%).

6) Qualitative Results: In Figs. 6 and 7, we visualize attention heads at each layer of MGT for various test set samples. Each subfigure contains attention heads for each of the three graphs ($\mathbf{A}^{1-\text{hop}}$, $\mathbf{A}^{2-\text{hop}}$, $\mathbf{A}^{\text{global}}$), and each of the rows #1– #4 in each subfigure correspond to layers #1– #4. Darker reds indicate higher attention values. All figures are best viewed in color.

Attention heads in the initial layers attend very strongly to certain neighbors and very weakly to others, i.e., the model builds local patterns for sketch subcomponents (strokes) through message passing along their contours. In the penultimate layers, the intensity of neighborhood attention is significantly lower and evenly distributed, indicating that the model is aggregating information from various strokes at each node.

In addition, we believe A^{global} graphs are critical for message passing between strokes, enabling the model to understand their relative arrangement. For example, in Fig. 6, both the head and feet of the bird are attached to the bottom of its body. In Fig. 7, the feet of the teddy bear are associated.

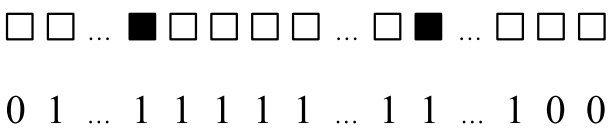


Fig. 8. Partially-connected graph illustration for a long sentence. The entities and the other words are denoted by the solid and hollow squares, respectively. The corresponding attention mask is also provided.

7) Comparison for Time Cost: To demonstrate the speed advantage of our model, we also compare our MGT with the strongest CNN baselines, i.e., Inception V3, MobileNet V2. We report the total time taken by models to perform inference over 34 500 sketches from the test set in Table VI. All models were implemented in PyTorch and run on an Intel Xeon CPU E5-2690 v4 server using a single Nvidia 1080Ti GPU, with a batch size of 256 and 16 workers per run.

In Table VI, we observe: our MGT infers obviously faster than both Inception V3 and MobileNet V2.

Note that, unlike CNNs and RNNs, GNNs, and GTs for sparse graph data formats are not natively supported by PyTorch. We believe that our MGT can infer faster in our future work, if we further speed up MGT via using the tailor-made GNN libraries such as Deep Graph Library[5] and PyTorch Geometric.[6]

8) Evaluation on RE: The main idea of our MGT is injecting domain knowledge into transformers through domain-specific graphs. We also try to evaluate this idea in other modalities beyond sketch, e.g., NLP tasks. Thus, we transfer our MGT idea to conduct RE on a RE benchmark (SemEval-2010 task 8 [62]), outperforming the state-of-the-art CNNs by a clear margin [as reported in Table VII, prediction accuracy: ours (89.45%) versus CNN [60] (86.60%)]. In particular, when we use bidirectional encoder representations from transformers (BERT) [37] as a transformer-based encoder, each sentence will be encoded as a fully connected graph. Given a long sentence, if its two entities are overly far from each other, the RE model should give more attention on the words between and around the two entities. Therefore, we also use multiple graph structures to inject this domain knowledge into the transformer-based encoder.

(i) Fully connected: In the early and middle stages of training, we allow the transformer to encode each sentence as a fully connected graph, regardless of its length.

(ii) Partially-connected: In the last few epochs, if in a given long sentence its two entities are far apart, we force the transformer to encode the long sentence as a partially-connected graph, which consists of the words between and around the two entities. We use a binary attention mask to encode this. See Fig. 8 for an illustration.

This experimental phenomenon further encourages us to explore the applications of using domain-specific graph structures to inject the domain-knowledge into transformers in other modalities and tasks.

[5]https://www.dgl.ai/

[6]https://github.com/rusty1s/pytorch_geometric

V. Conclusion

This article introduces a novel representation of free-hand sketches as multiple sparsely connected graphs. We design a MGT for capturing both geometric structure and temporal information from sketch graphs. The intrinsic traits of the MGT architecture include: 1) using graphs as universal representations of sketch geometry, as well as temporal and semantic information; 2) injecting domain knowledge into transformers through sketch-specific graphs; and 3) making full use of multiple intrastroke and extra-stroke graphs.

For sketch community: We, for the first time, propose to model sketches as sparsely connected graphs. For the GNN community: We design a novel GT model that considers prior/domain knowledge via multiple graph structures. This multigraph modeling idea works well for both sketch and other modalities.

We hope MGT can serve as a foundation for future work in sketch applications and network architectures, motivating the community toward sketch representation learning using graphs. In addition, for the GNN community, we hope that MGT helps free-hand sketch become a new test-bed for GNNs.

VI. Future Work

Our future work will focus on applying our MGT as a general neural representation in various sketch tasks, e.g., sketch generation. More code and results will be updated continuously on our project page.[7]

References

[1] D. Ha and D. Eck, "A neural representation of sketch drawings," 2017, *arXiv:1704.03477*. [Online]. Available: http://arxiv.org/abs/1704.03477

[2] P. Xu *et al.*, "SketchMate: Deep hashing for million-scale human sketch retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8090–8098.

[3] F. Liu, X. Deng, Y.-K. Lai, Y.-J. Liu, C. Ma, and H. Wang, "SketchGAN: Joint sketch completion and recognition with generative adversarial network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5830–5839.

[4] R. K. Sarvadevabhatla, J. Kundu, and V. R. Babu, "Enabling my robot to play pictionary: Recurrent neural networks for sketch recognition," in *Proc. 24th ACM Int. Conf. Multimedia*, 2016, pp. 247–251.

[5] Y. Ye, Y. Lu, and H. Jiang, "Human's scene sketch understanding," in *Proc. ACM Int. Conf. Multimedia Retr.*, Jun. 2016, pp. 355–358.

[6] P. Sangkloy, N. Burnell, C. Ham, and J. Hays, "The sketchy database: Learning to retrieve badly drawn bunnies," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–12, 2016.

[7] L. Liu, F. Shen, Y. Shen, X. Liu, and L. Shao, "Deep sketch hashing: Fast free-hand sketch-based image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2862–2871.

[8] Y. Shen, L. Liu, F. Shen, and L. Shao, "Zero-shot sketch-image hashing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3598–3607.

[9] J. Collomosse, T. Bui, and H. Jin, "LiveSketch: Query perturbations for guided sketch-based visual search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2879–2887.

[10] A. Dutta and Z. Akata, "Semantically tied paired cycle consistency for zero-shot sketch-based image retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5089–5098.

[11] S. Dey, P. Riba, A. Dutta, J. L. Llados, and Y.-Z. Song, "Doodle to search: Practical zero-shot sketch-based image retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2179–2188.

[7] https://github.com/PengBoXiangShang/multigraph_transformer

[12] W. Chen and J. Hays, "SketchyGAN: Towards diverse and realistic sketch to image synthesis," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9416–9425.

[13] Y. Lu, S. Wu, Y.-W. Tai, and C.-K. Tang, "Image generation from sketch constraint using contextual GAN," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 205–220.

[14] P. Xu, Z. Song, Q. Yin, Y.-Z. Song, and L. Wang, "Deep self-supervised representation learning for free-hand sketch," *IEEE Trans. Circuits Syst. Video Technol.*, early access, Jun. 17, 2020, doi: 10.1109/TCSVT.2020.3003048.

[15] H. Lin, Y. Fu, X. Xue, and Y.-G. Jiang, "Sketch-BERT: Learning sketch bidirectional encoder representation from transformers by self-supervised learning of sketch gestalt," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6758–6767.

[16] Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. Hospedales, "Sketch-a-Net that beats humans," 2015, *arXiv:1501.07873*. [Online]. Available: http://arxiv.org/abs/1501.07873

[17] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 5998–6008.

[18] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*. [Online]. Available: http://arxiv.org/abs/1806.01261

[19] K. Li, K. Pang, Y.-Z. Song, T. Xiang, T. M. Hospedales, and H. Zhang, "Toward deep universal sketch perceptual grouper," *IEEE Trans. Image Process.*, vol. 28, no. 7, pp. 3219–3231, Jul. 2019.

[20] P. Xu, T. M. Hospedales, Q. Yin, Y.-Z. Song, T. Xiang, and L. Wang, "Deep learning for free-hand sketch: A survey and a toolbox," 2020, *arXiv:2001.02600*. [Online]. Available: http://arxiv.org/abs/2001.02600

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[22] J. Song, Q. Yu, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Deep spatial-semantic attention for fine-grained sketch-based image retrieval," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5551–5560.

[23] Q. Jia, M. Yu, X. Fan, and H. Li, "Sequential dual deep learning with shape and texture features for sketch recognition," 2017, *arXiv:1708.02716*. [Online]. Available: http://arxiv.org/abs/1708.02716

[24] L. Li, C. Zou, Y. Zheng, Q. Su, H. Fu, and C.-L. Tai, "Sketch-R2CNN: An RNN-rasterization-CNN architecture for vector sketch recognition," *IEEE Trans. Vis. Comput. Graphics*, early access, Apr. 15, 2020, doi: 10.1109/TVCG.2020.2987626.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[26] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Represent.*, 2014.

[27] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 3844–3852.

[28] S. Sukhbaatar *et al.*, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 2244–2252.

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

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[31] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5115–5124.

[32] U. S. Shanthamallu, J. J. Thiagarajan, H. Song, and A. Spanias, "GrAMME: Semisupervised learning using multilayered graph attention models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 3977–3988, Oct. 2020.

[33] S. Edunov, M. Ott, M. Auli, and D. Grangier, "Understanding back-translation at scale," 2018, *arXiv:1808.09381*. [Online]. Available: http://arxiv.org/abs/1808.09381

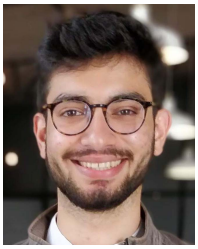
[34] Q. Wang *et al.*, "Learning deep transformer models for machine translation," 2019, *arXiv:1906.01787*. [Online]. Available: http://arxiv.org/abs/1906.01787

[35] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. (2018). Improving Language Understanding By Generative Pre-Training. OpenAI Blog. [Online]. Available: https://openai.com/blog/language-unsupervised/

[36] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," 2019, *arXiv:1901.02860*. [Online]. Available: http://arxiv.org/abs/1901.02860
[37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805
[38] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," 2019, *arXiv:1906.08237*. [Online]. Available: http://arxiv.org/abs/1906.08237
[39] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: http://arxiv.org/abs/1409.0473
[40] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018.
[41] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang, "BP-transformer: Modelling long-range context via binary partitioning," 2019, *arXiv:1911.04070*. [Online]. Available: http://arxiv.org/abs/1911.04070
[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167
[45] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–10, 2012.
[46] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[47] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[49] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[50] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, *arXiv:1409.1259*. [Online]. Available: http://arxiv.org/abs/1409.1259
[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.
[52] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[53] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[54] X. Bresson and T. Laurent, "An experimental study of neural networks for variable graphs," in *Proc. Int. Conf. Learn. Represent. Workshop*, 2018.
[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556
[56] J.-J. Liu, Q. Hou, M.-M. Cheng, C. Wang, and J. Feng, "Improving convolutional networks with self-calibrated convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10096–10105.
[57] D. Haase and M. Amthor, "Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved MobileNets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14600–14609.
[58] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.
[59] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*. [Online]. Available: http://arxiv.org/abs/1909.08053
[60] X. Guo, H. Zhang, H. Yang, L. Xu, and Z. Ye, "A single attention-based combination of CNN and RNN for relation classification," *IEEE Access*, vol. 7, pp. 12467–12475, 2019.
[61] H. Wang *et al.*, "Extracting multiple-relations in one-pass with pre-trained transformers," 2019, *arXiv:1902.01030*. [Online]. Available: http://arxiv.org/abs/1902.01030
[62] I. Hendrickx *et al.*, "SemEval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals," 2019, *arXiv:1911.10422*. [Online]. Available: http://arxiv.org/abs/1911.10422

Peng Xu received the Ph.D. degree from the Pattern Recognition and Intelligent System Laboratory, Beijing University of Posts and Telecommunications, Beijing, China, in 2019.

He is currently a Post-Doctoral Research Fellow with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include fine-grained free-hand sketch analysis, graph neural network, multimodal computing, deep learning, and computer vision.



Chaitanya K. Joshi received the B.Eng. degree in computer science from Nanyang Technological University, Singapore, in 2019, as the Valedictorian.

He was a Research Assistant with Prof. Xavier Bresson. He is currently a Research Engineer with the Institute for Infocomm Research, A*STAR, Singapore. His research interests include graph neural networks, combinatorial optimization, and natural language processing.

Xavier Bresson received the Ph.D. degree from EPFL, Lausanne, Switzerland, in 2005. He is an Associate Professor in computer science with Nanyang Technological University (NTU), Singapore. He is a Leading Researcher in the field of graph deep learning, a new framework that combines graph theory and deep learning techniques to tackle complex data domains in natural language processing (NLP), computer vision, combinatorial optimization, quantum chemistry, physics, neuroscience, genetics, and social networks. In 2016, he received the highly competitive Singaporean NRF Fellowship of 2.5M to develop these deep learning techniques. As a Leading Researcher in the field, he has published more than 60 peer-reviewed papers in the leading journals and conference proceedings in machine learning, including articles in Conference on Neural Information Processing Systems (NeurIPS), International Conference on Machine Learning (ICML), International Conference on Learning Representations (ICLR), IEEE Conference on Computer Vision and Pattern Recognition (CVPR), *Journal of Machine Learning Research (JMLR)*. He has organized several international workshops and tutorials on AI and deep learning in collaboration with Facebook, NYU, and Imperial such as the 2019 and 2018 UCLA, Los Angeles, CA, USA, workshops, the 2017 CVPR tutorial, and the 2017 NeurIPS tutorial. He has been teaching undergraduate, graduate, and industrial courses in AI and deep learning since 2014 at EPFL, NTU, and UCLA.

Dr. Bresson was a recipient of several research grants in the United States and Hong Kong.