Title	Sketch-R2CNN: An RNN-Rasterization-CNN Architecture for Vector Sketch Recognition
Authors	Li, Lei; Zou, Changqing; Zheng, Youyi; Su, Qingkun; Fu, Hongbo; Tai, Chiew Lan
Source	IEEE Transactions on Visualization and Computer Graphics, v. 27, (9), September 2021, article number 9068451, p. 3745-3754
Version	Accepted Version
DOI	10.1109/tvcg.2020.2987626
Publisher	IEEE
Copyright	© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Sketch-R2CNN: An RNN-Rasterization-CNN Architecture for Vector Sketch Recognition

Lei Li, Changqing Zou, Youyi Zheng, Qingkun Su, Hongbo Fu, and Chiew-Lan Tai

Abstract—Sketches in existing large-scale datasets like the recent QuickDraw collection are often stored in a vector format, with strokes consisting of sequentially sampled points. However, most existing sketch recognition methods rasterize vector sketches as binary images and then adopt image classification techniques. In this article, we propose a novel end-to-end single-branch network architecture *RNN-Rasterization-CNN* (*Sketch-R2CNN* for short) to fully leverage the vector format of sketches for recognition. Sketch-R2CNN takes a vector sketch as input and uses an RNN for extracting per-point features in the vector space. We then develop a neural line rasterization module to convert the vector sketch and the per-point features to multi-channel point feature maps, which are subsequently fed to a CNN for extracting convolutional features in the pixel space. Our neural line rasterization module is designed in a differentiable way for end-to-end learning. We perform experiments on existing large-scale sketch recognition datasets and show that the RNN-Rasterization design brings consistent improvement over CNN baselines and that Sketch-R2CNN substantially outperforms the state-of-the-art methods.

Index Terms—Freehand sketching, RNN, CNN, neural rasterization, object classification, QuickDraw

1 Introduction

Freehand sketching is an easy and quick means of communication because of its simplicity and expressiveness. While we human beings have the innate ability to interpret drawing semantics, it is still a challenging task for machines. Sketch analysis has been an active research topic in the computer vision and graphics fields, spanning a wide spectrum including sketch recognition [1], [2], [3], sketch segmentation [4], [5], [6], [7], sketch-based retrieval [8], [9], [10], [11] and modeling [12], etc. In this paper, we focus on developing a novel learning-based approach for freehand sketch recognition.

The goal of sketch recognition (or classification) is to identify the object category of an input sketch, which is more challenging than natural image classification largely due to the inherent ambiguities, geometric variations and lack of rich texture details in the input. Traditional studies [1], [13], [14] mostly cast sketch recognition as an image classification task by converting sketches to binary images and then extracting hand-crafted local features from the images. With the quantified image features, a typical classifier such as Support Vector Machine (SVM) is trained for object category prediction. Recent years have witnessed the impressive success of deep learning techniques in image classification [15], [16], [17], [18], and convolutional neural networks (CNNs) have also been applied to the recognition of sketch images [2], [10]. Although these deep learning-based methods outperform the traditional ones, the unique properties of sketches, as discussed in the following, are often overlooked, leaving room for performance improvement.

Thanks to the ubiquity of input devices, sketches are often acquired digitally and stored in a vector format [19], [20], represented as a sequence of strokes (polylines) with each stroke consisting of a point sequence in the drawing order (Fig. 1). Such vector sketch data, like those in the QuickDraw dataset [19], includes (1) positional information of points, (2) temporal order (stroke order and point order within each stroke) and (3) grouping of points as strokes (or pen states). The latter two types of information, however, cannot be effectively accessed by existing CNNs [2], [10], [21], which deal with the rasterized version of vector sketches.

The recently proposed *SketchRNN* [19] and its follow-up studies have shown that recurrent neural networks (RNNs) can directly take vector sketches as inputs to learn descriptive feature representations, enabling various tasks like vector sketch synthesis [22], [23] or segmentation [6]. Motivated by this, researchers have also incorporated the vector format, serving as a complement to the pixel format, in sketch-based retrieval [11], [24]. They typically adopt a two-branch network architecture: a CNN branch for the pixel sketch and an RNN branch for the vector sketch; a concatenation layer at last is used to fuse feature representations from the two branches. However, the RNN and CNN barely have learning interactions in such a design, and it demands the networks

- *L. Li and C.-L. Tai are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: {llibb, taicl}@cse.ust.hk.*
- *C. Zou is with Huawei HMI Lab, Huawei Technologies, Shenzhen 518129, China. E-mail: aaronzou1125@gmail.com.*
- *Y. Zheng is with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: youyizheng@zju.edu.cn.*
- *Q. Su is with the A.I. Labs, Alibaba Group, Hangzhou 310052, China. E-mail: suqingkun@gmail.com.*
- *H. Fu is with the School of Creative Media, City University of Hong Kong, Kowloon Tong, Hong Kong. E-mail: hongbofu@cityu.edu.hk.*

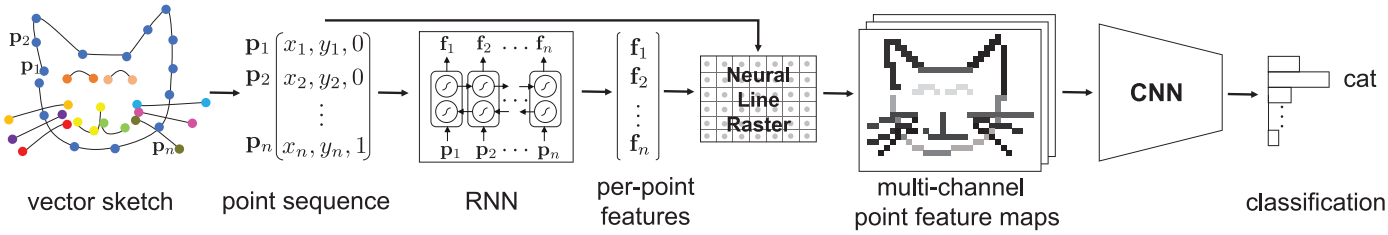

Fig. 1. Illustration of our Sketch-R2CNN architecture for vector sketch recognition. A neural line rasterization (NLR) module is designed to convert per-point features, produced by the RNN with the input vector sketch, to multi-channel point feature maps, which are then consumed by an off-the-shelf CNN for recognition.

to learn to balance contributing weights of the two types of features in the concatenated feature space.

In this work, we seek to boost the synergy between the RNN and CNN with more information flow during end-to-end learning. To this end, a key step is to convert input vector sketches to pixel images within neural networks. Conventional rasterization of sketches is a fixed discretization operation lacking gradient definitions for back propagation, thus it cannot be directly incorporated in neural networks. Inspired by [25], we utilize linear interpolation and propose differentiable line rasterization. Building upon this, we design a novel end-to-end single-branch network architecture *RNN-Rasterization-CNN* (*Sketch-R2CNN* for short) for vector sketch recognition, as illustrated in Fig. 1.

Specifically, Sketch-R2CNN takes as input only a vector sketch and employs an RNN to extract feature representations for each point of the sketch. The RNN is similar to the RNN encoder used in [11], [19] for learning latent representations of vector sketches. We then develop a *neural line rasterization* (NLR) module, which converts the vector sketch with the per-point features to multi-channel point feature maps in a differentiable way. Subsequently, an off-the-shelf CNN consumes the resulting point feature maps and predicts the target object category as output. The NLR module allows the CNN to access the features of vector sketches at early stages, bridging the gap between the vector sketch space and pixel sketch space in neural networks. The module is comparatively lightweight and can be easily attached to various CNN backbones with little modification. Experiments on existing crowd-sourced datasets [1], [19] show that by leveraging the vector format of sketches, our RNN-Rasterization-CNN architecture can consistently improve the recognition performance of CNN-only methods. Particularly, on the million-scale testing dataset of QuickDraw [19] (similar to the scale of ImageNet [26]), Sketch-R2CNN outperforms CNN counterparts (including ResNet [17] and DenseNet [18]) by 19K - 31K recognition successes (2.2 - 3.6 percent).

In summary, our contributions in this work are: (1) the first single-branch architecture with sequentially-arranged RNN and CNN for vector sketch recognition, achieving the state-of-the-art accuracy; (2) a differentiable line rasterization module that connects the vector sketch space and pixel sketch space in neural networks, allowing end-to-end learning. We will make our code publicly available.

2 Related Work

To recognize sketched objects, traditional methods generally take preprocessed pixel sketches as inputs. To quantify a sketch image, existing studies have tried to utilize various hand-crafted local features originally intended for photos (e.g., bag-of-features [1], Fisher Vectors with SIFT features [13], or HOG features [14]). With the extracted features, classifiers (e.g., SVMs) are then trained to recognize unseen sketches [1], [13]. Different learning schemes, such as multiple kernel learning [14] or active learning [27], may be employed for performance improvement. Another line of traditional methods has attempted to exploit additional cues for sketch recognition, such as prior knowledge of specific domains [28], [29], [30], [31], [32], [33] or object context of sketched scenes [3], [34]. Although progress has been made in sketch recognition, these methods still cannot robustly handle freehand sketches with large abstraction variations, especially those hastily drawn in dozens of seconds [19], struggling to achieve performance on par with human on the existing TU-Berlin dataset [1].

Recently, deep learning has revolutionized many research fields, including sketch recognition, with state-of-the-art performance. Research efforts [2], [10], [35] have been made to employ deep neural networks (e.g., AlexNet [15] or GoogLeNet [16]) to learn more discriminative image features from pixel sketches to replace hand-engineered features. Yu *et al*. [2] proposed *Sketch-a-Net*, an AlexNet-like architecture specifically adapted for sketch images by using large kernels in convolutions to accommodate the sparsity of stroke pixels. Their method achieved superior classification accuracy (77.95 percent) on the TU-Berlin dataset [1], surpassing human performance (73.1 percent) for the first time. Their method still follows the existing learning paradigm of image classification, i.e., using converted binary sketch images as CNN inputs, and thus cannot end-to-end learn from the additional information contained in vector sketches by design. In contrast, our approach uses an RNN to directly take the vector format of sketches as input for analysis and then produces informative point feature maps for the subsequent CNN.

The vector format of sketches has been considered in several deep learning tasks, such as sketch synthesis [19], [20], [22], [23], [36], [37], sketch abstraction [37] and sketch segmentation [6]. Notably, SketchRNN proposed by Ha and Eck [19], which receives much attention recently, is a Sequence-to-Sequence Variational Autoencoder built upon RNNs for vector sketch synthesis. This work shows that RNN can encode a vector sketch as a descriptive low-dimensional latent vector, from which a sketch of similar shape and drawing order can be reconstructed. Several follow-up studies have extended this idea to other sketch-related problems [6], [22], [23], [37]. For example, the work of Song *et al*. [22] learns an RNN-based translation model with shortcut cycle consistency to generate vector sketches from real

photos. Li *et al.* [6] adopted an RNN encoder to learn feature representations for each single stroke, which are then used to group semantically similar strokes in a sketch.

There exist a few studies that try to combine the vector and pixel formats of sketches to learn more descriptive fused features. The two-branch late-fusion network used in sketch-based retrieval [11], [24] is probably the most relevant to ours. In this design, the pixel format of an input sketch is fed to a CNN branch and the corresponding vector format is fed to a parallel RNN branch. A concatenation layer at last aggregates feature representations from the two branches. Although the retrieval performance benefits from the fused features, the RNN and CNN individually work on two different sketch spaces with little learning interaction, except at the last concatenation layer. In contrast, our single-branch RNN-Rasterization-CNN design brings more information flow between the RNN and CNN, which is enabled by our differentiable neural line rasterization (NLR) module. The evaluation (Section 4.3) shows that our approach outperforms the two-branch late-fusion network.

Our network is also related to CNN with an attention mechanism. Attention has been widely employed in many visual tasks, such as image classification [38], [39], [40], [41] and image captioning [42], [43]. An attention module in CNN generally works by computing soft masks over the spatial image grid [40], [42] or feature channels [41] to obtain weighted combination of features. This technique has also been applied to the sketch domain. For example, Song *et al.* [44] have incorporated a spatial attention module in Sketch-a-Net for fine-grained sketch-based image retrieval. While their work strives to estimate attention from the pixel format that contains limited visual information, our method derives attentive point feature maps from the vector format with in-network rasterization.

3 Methodology

The architecture of our Sketch-R2CNN is illustrated in Fig. 1. Given a vector sketch S as input (Section 3.1), our network seeks to interpret its object category by jointly considering the feature representations learned in the vector sketch space as well as in the pixel sketch space. Existing CNN-based approaches [2], [10] perform recognition only with the pixel version of S, which is a structured but reduced input representation complying with CNNs. To exploit the drawing cues in S, we resort to an RNN for analyzing the points of S sequentially and extracting features for each point (Section 3.2). To inform the CNN with the learned RNN features, we design a neural line rasterization (NLR) module that converts S with the per-point features to multi-channel point feature maps in a differentiable way (Section 3.3). The NLR module is the key enabler for connecting the two sub-networks that operate in completely different spaces. Compared to the pixel sketch input, the point feature maps are capable of delivering more drawing cues to the CNN.

3.1 Input Representation

We consider the input vector sketch S to be a sequence of strokes, each stroke comprising of a sequence of points. This vector format is widely adopted for sketches in many existing crowdsourced datasets [1], [10], [19], [45].

Following [20], we represent S as an ordered point sequence $S = \{\mathbf{p}_i = (x_i, y_i, s_i)\}_{i=1...n}$, where x_i and y_i are the 2D coordinates of point $\mathbf{p}_i$, s_i is a binary pen state, and n is the total number of points in all strokes. Specifically, state $s_i = 0$ indicates that the current stroke has not ended and that the stroke connects $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$; $s_i = 1$ indicates that $\mathbf{p}_i$ is the last point of the current stroke and $\mathbf{p}_{i+1}$ is the starting point of another stroke.

3.2 Network Architecture

In the initial stage of Sketch-R2CNN, an RNN is adopted to perform analysis on the point sequence of S and then produce a feature vector for each point $\mathbf{p}_i$. At time step i, the recurrent operation of the RNN can be expressed in a general form as

$$\begin{aligned}[\mathbf{h}_i; \mathbf{c}_i] &= \mathcal{G}_r(\mathbf{p}_i, [\mathbf{h}_{i-1}; \mathbf{c}_{i-1}]),\\ \mathbf{f}_i &= \mathcal{G}_f(\mathbf{h}_i),\end{aligned} \tag{1}$$

where $\mathbf{h}$ represents the hidden states of the RNN, $\mathbf{c}$ is optional cell states, and $\mathbf{f}_i \in \mathbb{R}^d$ is a d-dimensional point feature output for $\mathbf{p}_i$. The symbol $\mathcal{G}_r$ denotes a nonlinear mapping for recurrently updating the internal states, and $\mathcal{G}_f$ denotes a nonlinear function that projects the hidden states to the desired outputs. This vector sketch encoding scheme follows the encoder network of SketchRNN [19]. In our implementation, we use a bidirectional Long Short-Term Memory (LSTM) [46] unit with two layers as $\mathcal{G}_r$. We set the sizes of hidden states and cell states to both be 512 and adopt dropout with probability = 0.5. For $\mathcal{G}_f$, we employ a fully-connected layer followed by a sigmoid function. Similar to [19], instead of using absolute coordinates, for each $\mathbf{p}_i$ fed into the RNN, we compute the offsets from its previous point $\mathbf{p}_{i-1}$ as its 2D coordinates.

As described in Eq. (1), the RNN progressively evaluates the point sequence in S along the temporal dimension, and thus it might fall short in discovering correlations of points that are temporally distant but spatially close to each other in 2D. In contrast, CNNs are known to excel at constructing hierarchical representations for 2D inputs [47], where neighboring pixels interact at lower layers and distant pixels interact at high layers. To allow CNNs to gain access to the per-point features learned by the RNN for 2D analysis, we perform in-network rasterization for S with a differentiable NLR module as detailed in Section 3.3.

We pass the point sequence along with the point features, i.e., $\{(\mathbf{p}_i, \mathbf{f}_i)\}_{i=1...n}$, through our differentiable NLR module. Conceptually, the NLR module "draws" the per-point features from RNN onto a multi-channel image, following the rasterization process of a vector sketch to a pixel sketch. The output of NLR is d-channel point feature maps of size $h \times w \times d$, with each channel corresponding to one component of the point features. The symbols h and w are the height and width of the resulting maps respectively. The dimension d is a flexible hyper parameter. For example, attention maps estimated by CNNs in existing studies [44] are similar to a special case of our design (i.e., $d = 1$), but differently, our design exploits a new attention source (i.e., feature representations in the vector sketch space). Apart from attention maps, our design can also deliver non-trivial feature patterns discovered by the RNN to CNNs, as illustrated in Section 4.2.

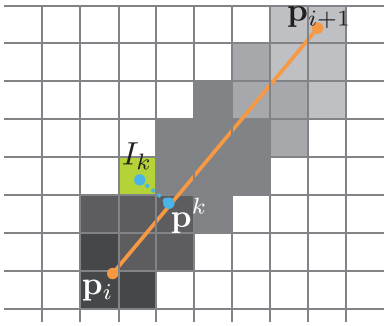

Fig. 2. Rasterization of line segment $\mathbf{p}_i\mathbf{p}_{i+1}$ and linear interpolation of the feature value for stroke pixel I_k.

The subsequent sub-network, a deep CNN, takes the d-channel point feature maps as inputs for hierarchical feature extraction. A wide range of CNNs for image recognition on ImageNet [26] (e.g., ResNet [17] or DenseNet [18]) can be used. At last, the CNN backbone is attached to a fully-connected layer to predict object categories. We use the cross entropy loss for optimizing the whole Sketch-R2CNN.

3.3 Neural Line Rasterization

The goal of our NLR module is to perform in-network vector-to-pixel sketch conversion. The module is designed to be differentiable so that it can be easily attached to existing CNNs for end-to-end learning. The NLR module takes as input the point sequence of S with per-point features $\{(\mathbf{p}_i, \mathbf{f}_i)\}_{i=1...n}$. Let $f_i^c \in \mathbb{R}$ ($c \in [1, d]$) denote the cth component of $\mathbf{f}_i$, and $I^c \in \mathbb{R}^{h \times w}$ be the cth channel of the resulting feature maps. In the following, for ease of explanation, we describe the rasterization process of $\{(\mathbf{p}_i, f_i^c)\}$ to I^c, which can be done independently and similarly for each feature component c. To simplify notations, the symbol c in f_i^c and I^c is omitted in the remainder of this section.

In the forward pass, the basic operation of NLR is to draw each valid line segment $\mathbf{p}_i\mathbf{p}_{i+1}$ (i.e., $s_i = 0$ as defined in Section 3.1) onto the canvas I. Similar to the conventional line rasterization, to determine whether or not a pixel I_k is covered by the line segment $\mathbf{p}_i\mathbf{p}_{i+1}$ (Fig. 2), we simply compute the distance from the pixel's center to the line segment and check whether it is smaller than a predefined threshold ϵ (we set $\epsilon = 1$ in our experiments). If I_k is a stroke pixel, we compute its feature value by linear interpolation; otherwise its feature value is set to zero. More specifically, let $\mathbf{p}^k$ be the projection point of I_k's center onto $\mathbf{p}_i\mathbf{p}_{i+1}$, and the feature value of I_k is defined as

$$I_k = (1 - \alpha_k) \cdot f_i + \alpha_k \cdot f_{i+1}, \tag{2}$$

where $\alpha_k = \|\mathbf{p}^k - \mathbf{p}_i\|_2 / \|\mathbf{p}_{i+1} - \mathbf{p}_i\|_2$, and $\mathbf{p}^k$, $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are in absolute 2D coordinates. If a pixel is covered by multiple line segments, we perform visibility test according to the drawing order.

Through the above process, a vector sketch can be easily converted into a pixel image (or point feature maps) in the forward pass. In order to propagate the gradients (w.r.t the loss function) from CNN to RNN in the backward optimization pass, we need to derive gradients for the above rasterization process. Owing to the simplicity of linear interpolation in Eq. (2), the gradients for the rasterization of $\mathbf{p}_i\mathbf{p}_{i+1}$ with f_i and f_{i+1} can be computed as follows:

$$\frac{\partial I_k}{\partial f_i} = 1 - \alpha_k, \quad \frac{\partial I_k}{\partial f_{i+1}} = \alpha_k. \tag{3}$$

Let L be the loss function and δ_k^I be the gradient w.r.t L back-propagated into I_k through CNN. By the chain rule, we have

$$\frac{\partial L}{\partial f_i} = \sum_k \delta_k^I \cdot (1 - \alpha_k), \ \frac{\partial L}{\partial f_{i+1}} = \sum_k \delta_k^I \cdot \alpha_k, \tag{4}$$

where k iterates over all the stroke pixels covered by the line segment $\mathbf{p}_i\mathbf{p}_{i+1}$. If segment $\mathbf{p}_{i-1}\mathbf{p}_i$ exists (i.e., $s_{i-1} = 0$), we accumulate the gradients for $\mathbf{p}_i$. With the computation in Eq. (4), the gradients (w.r.t L) can continue to flow into RNN for optimizing the learning of point features in the vector sketch space. Note that f_i and f_{i+1} are scalar components of the point features $\mathbf{f}_i$ and $\mathbf{f}_{i+1}$ respectively, which are independently derived from the hidden states $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ according to Eq. (1). The recurrent relations between $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ are already modeled by the RNN (i.e., $\mathcal{G}_r$). Therefore, no gradient computation is needed between f_i and f_{i+1} in NLR.

The NLR module is non-parametric as it emulates the conventional line rasterization. NLR enables the unification of the two sketch spaces in a single neural network, and its differentiability allows learning interactions between the RNN and CNN. Unlike a two-branch design [11], the RNN and CNN arranged sequentially in Sketch-R2CNN can cooperate more effectively and work towards the same goal. On one hand, the CNN, serving as an abstract visual concept extractor, is the main workhorse for recognition, and the RNN complements the CNN with feature representations extracted from a sequential data format. On the other hand, the CNN informs the RNN with 2D spatial relationships of the points, which aid the RNN in learning correlations of temporally-distant but spatially-close points.

4 Experiments

4.1 Datasets and Settings

Datasets. We evaluated the performance of Sketch-R2CNN on two existing crowd-sourced sketch datasets. The first one is the TU-Berlin dataset [1], which contains 250 object categories with only 80 sketches per category (i.e., 20K sketches in total). Each sketch in TU-Berlin was created within 30 minutes by a participant from Amazon Mechanical Turk. While most previous recognition methods have been evaluated on TU-Berlin, since it is a relatively small-scale dataset, CNNs with millions or tens of millions of parameters (e.g., ResNet [17] and DenseNet [18]) tend to overfit the data (Section 4.2). Thus, for more reliable evaluations with deep CNNs, we also performed experiments on a recently introduced million-scale dataset - the QuickDraw dataset [19], which contains 345 categories with 75K sketches per category (25.8 million sketches in total). Since during acquisition the participants were given only 20 seconds to draw an object, sketches in QuickDraw are more abstract and contain fewer strokes than those in TU-Berlin. Detailed statistics of the number of strokes per sketch of the two datasets are listed in Table 1.

Sketches in QuickDraw have already been preprocessed with the Ramer-Douglas-Peucker (RDP) simplification algorithm [19], and the maximum number of points of a sketch is 321. For sketches in TU-Berlin, we performed

TABLE 1
Statistics of the TU-Berlin and QuickDraw Datasets After Preprocessing: The Number of Strokes and the Number of Points per Sketch

Dataset	#strokes per sketch			#points per sketch		
	Median	Mean	Stdev	Median	Mean	Stdev
TU-Berlin	13.0	17.5	16.4	179.0	203.6	113.3
QuickDraw	4.0	5.1	3.8	47.0	52.9	29.0

similar simplification with the RDP algorithm, and the maximum number of points of a sketch is 448. Table 1 lists more detailed statistics of the number of points per sketch of the two datasets.

Implementation. We implemented our Sketch-R2CNN with PyTorch. The dimension d of point features produced by the RNN is set to 8 (Section 4.3). We tested Sketch-R2CNN with various CNN backbones to show consistent improvements brought by our single-branch design. Specifically, Sketch-a-Net v2 [2] (SN v2 for short) achieved the state-of-the-art performance on TU-Berlin, but its original implementation based on Caffe is not compatible with our NLR implementation. Thus we reproduced and re-trained SN v2 with PyTorch for evaluation (Section 4.2). Furthermore, we also performed experiments with several off-the-shelf CNNs pre-trained on ImageNet, including ResNet50 [17], ResNet101 [17], and DenseNet161 [18]. Compared to SN v2, these CNNs are significantly larger in terms of network size and thus require longer training time.

Training. Comparable to the scale of ImageNet [26], the QuickDraw dataset has already been divided into training, validation and testing sets with sizes of 24.1 million, 862K and 862K, respectively. Due to the relatively small scale of TU-Berlin, following [2] we used data augmentation (including horizontal reflection, stroke removal and sketch deformation) during training, and adopted three-fold cross validation on this dataset (i.e., two folds for training and one fold for testing, 6.6K sketches per fold).

For training on TU-Berlin, due to the limited data, Yu *et al.* [2] used edge maps extracted from the photos of ImageNet as the pre-training data for SN v2. However, the synthesized drawings might contain various noise (e.g., edges from cluttered image backgrounds) and lack the artistic styles from human [45]. Instead, we used QuickDraw as the pre-training data for its fidelity to human drawing styles and ease of preparation to train the CNNs (the reproduced SN v2, ResNet50, ResNet101, and DenseNet161) on TU-Berlin. We observe that SN v2 pre-trained on QuickDraw achieves similar performance to [2] without additional network ensemble [48]. For Sketch-R2CNN, a similar training schedule was used: our network was first trained on QuickDraw (the RNN initialized with uniformly sampled weights and the CNN backbone with the pre-trained weights), and then fine-tuned on TU-Berlin. Note that the RNN and CNN in Sketch-R2CNN were jointly trained in an end-to-end manner on the two datasets. We adopted Adam [49] ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with a learning rate of 0.0001 for stochastic gradient descent update. For training with the reproduced SN v2, ResNet50 and ResNet101, a batch size of 48 was used; for training with DenseNet161, a batch size of 24 was used. The network training and evaluation were performed with an NVIDIA GTX 1080Ti GPU.

Metrics. Similar to [2], evaluation results are reported with top-1 recognition accuracy (Table 2). We also report accuracies of partial sketch recognition (Table 3), since in real applications users complete a drawing stroke by stroke, and recognition can be done on partially drawn sketches iteratively (like the Google Quick, Draw! Experiment).[1] For each testing sketch from TU-Berlin or QuickDraw, recognition of its 25, 50, or 75 percent strokes (at least one stroke in a partial sketch) in the drawing order was performed.

TABLE 2
Recognition Accuracy (%) of Different Methods on the TU-Berlin and QuickDraw Datasets

Model	Accuracy	
	TU-Berlin	QuickDraw
Humans [1]	73.1	-
HOG-SVM [1]	56.0	-
Ensemble [51]	61.5	-
MKL-SVM [14]	65.8	-
FV-SP [13]	68.9	-
LeNet [52]	55.2	-
AlexNet-SVM [15]	67.1	-
AlexNet-Sketch [15]	68.6	-
SN v1 [21]	74.9	-
SN v2 [2]	77.95	-
SN v2 (reproduced) [2]	77.5	74.8
ResNet50 [17]	83.4	82.5
ResNet101 [17]	83.7	83.1
DenseNet161 [18]	84.2	83.0
Ours (w/ SN v2 reproduced)	79.4	78.4
Ours (w/ ResNet50)	84.5	84.8
Ours (w/ ResNet101)	85.0	**85.3**
Ours (w/ DenseNet161)	**85.4**	85.2

The middle part lists the performance of the models reported in [2], among which only SN v2 adopts pre-training with ImageNet edge maps. Differently, CNNs in the bottom part are pre-trained with QuickDraw. Please refer to the main text for training details.

TABLE 3
Partial Sketch Recognition Accuracy (%) of Our Sketch-R2CNN and its CNN-Only Counterparts on the TU-Berlin and QuickDraw Datasets

Model	TU-Berlin			QuickDraw		
	25%	50%	75%	25%	50%	75%
SN v2 (reproduced) [2]	37.5	61.5	73.5	23.8	43.4	64.4
ResNet50 [17]	40.1	66.2	78.3	25.1	47.5	71.1
ResNet101 [17]	41.5	66.9	78.9	25.0	47.4	71.4
DenseNet161 [18]	41.8	67.6	79.5	**25.4**	48.1	71.7
Ours (w/ SN v2 reproduced)	38.6	62.8	74.7	23.7	44.8	67.4
Ours (w/ ResNet50)	42.5	68.5	80.2	24.8	48.4	73.1
Ours (w/ ResNet101)	43.2	69.1	80.3	24.8	48.2	73.2
Ours (w/ DenseNet161)	**44.1**	**69.7**	**81.2**	25.1	**48.7**	**73.5**

For a testing sketch, its 25, 50 or 75 percent strokes in the drawing order (as partial sketches) were used for recognition.

4.2 Comparison Results

TU-Berlin. Even though pre-training and data augmentation were used, we found that the compared networks easily

1. https://quickdraw.withgoogle.com

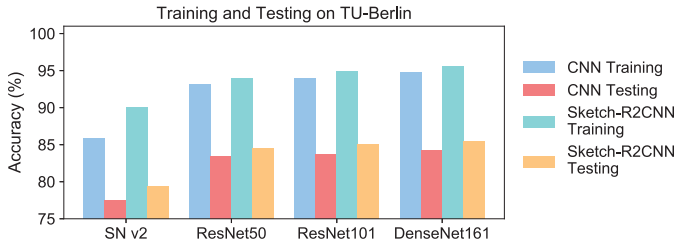

Fig. 3. A large gap between training and testing accuracy of CNNs and Sketch-R2CNN on TU-Berlin due to overfitting.

overfit the training data, as shown in Fig. 3, and thus hardly gain any useful gradients for further optimization. Nevertheless, for completeness of comparisons, we still report the performance of Sketch-R2CNN and its CNN counterparts in Table 2 (the bottom part) and Table 3.

Tables 2 & 3 show that Sketch-R2CNN consistently improves the recognition performance across different CNN backbones on both complete and partial sketch inputs (6.6K testing sketches in total). Specifically, in Table 2, Sketch-R2CNN (79.4 percent, 5.24K successes) outperforms the reproduced SN v2 (77.5 percent, 5.12K successes), which has similar performance to the original SN v2 model [2]. Sketch-R2CNN with DenseNet161 achieves the best performance (85.4 percent, 5.64K successes) on TU-Berlin. It is also observed that ResNet50 alone surpasses SN v2. This indicates the effectiveness of skip connections [17], which deliver lower-level features to higher layers, thus alleviating the loss of stroke details in convolutions [50]. However, increasing CNN depth (ResNet101 - 83.7 percent, DenseNet161 - 84.2 percent) leads to slightly better performance (ResNet50 - 83.4 percent). Comparatively, Sketch-R2CNN (84.5 percent) can improve ResNet50 without adding more convolutional layers. For partial sketches (25 and 50 percent), more than 2 percent improvement brought by Sketch-R2CNN is observed for most CNN backbones in Table 3.

QuickDraw. To further validate the performance, we conducted experiments on the million-scale QuickDraw dataset. Its voluminous data can help to address the overfitting issue (24.1 million training samples) and offer more statistically significant results (862K testing samples). The results in Tables 2 & 3 show the consistent improvement of Sketch-R2CNN over corresponding CNNs on QuickDraw.

In particular, Sketch-R2CNN (78.4 percent) outperforms the reproduced SN v2 (74.8 percent) by 3.6 percent (31K successes) on complete sketch inputs. Note that the original SN v1 and v2 were not tested on QuickDraw in [2]. Results with about 2.2 percent (19K successes) improvement brought by Sketch-R2CNN are obtained using the deeper networks (i.e., ResNet and DenseNet). Sketch-R2CNN with ResNet101 obtains the best performance (85.3 percent) on QuickDraw. Notably, for CNN-only methods, even with sufficient training data, increasing network depth (ResNet101—83.1 percent, DenseNet161—83.0 percent) only brings about 0.5 percent (4K successes) improvement (ResNet50—82.5 percent), which is $1/4$ of the improvement of Sketch-R2CNN (84.8 percent) over ResNet50. Since the number of strokes in each sketch of QuickDraw is 5.1 on average (Table 1), partial sketch inputs (25 and 50 percent) contain fewer strokes than the ones from TU-Berlin and are thus difficult to recognize for both CNNs and Sketch-R2CNN. Even so, for all the CNN backbones, the improvement of Sketch-R2CNN increases consistently with more strokes in the partial sketch inputs.

The above results of various CNN-only methods suggest that with the increased network depth and complexity, the performance of CNNs tends to saturate, and mining discriminative cues from pixel sketches for recognition becomes more difficult. By making the drawing cues in vector sketches accessible to CNNs, Sketch-R2CNN effectively boosts the performance even with smaller CNN backbones, which require less training time and smaller memory footprint. Sketch-R2CNN barely introduces modification to the CNN backbone, and the RNN-Rasterization design is relatively lightweight. With the active development of CNNs, we foresee that Sketch-R2CNN can achieve even better performance with more advanced emerging CNN architectures.

Qualitative Results. Fig. 4 shows some sample sketches that cause confusion to ResNet101 but are successfully recognized by Sketch-R2CNN (ResNet101). We also visualize the multi-channel point feature maps (Section 3.3) produced by the RNN-Rasterization module. It is observed that channels including I^3 - I^5 tend to have higher feature values at pixels covered by long curved strokes, which are important for depicting rough shapes, for example, the circle strokes in the cake or the arc stroke for the nose of the elephant. Channels like I^1, I^2, I^6 and I^8 tend to have higher feature values at pixels covered by short lines or endpoints of strokes, which are mostly for depicting details, for example, the strokes for the face of the lion or the sprinkles on the cake. There is also a certain channel (I^7) that has higher values for all stroke pixels except endpoints of strokes, and it may help to deliver the overall visual appearance information, as contained in binary pixel sketches, to CNNs. The above encoding scheme of vector sketches was learned by the RNN and CNN jointly owing to our NLR module through stochastic gradient descent. By analyzing in the vector space and in-network rasterization with NLR, the RNN constructs a nontrivial representation that differentiates levels of details in sketches. As input to CNNs, such a representation is obviously more informative than the binary pixel format of sketches used in existing studies [2] and can help CNNs to develop the awareness of hierarchical representations even at early stages for feature extraction.

4.3 Alternatives and Ablation Study

In this section, we performed experiments on network design alternatives and ablations of contributing factors for point feature extraction. We mainly used ResNet50 as the backbone here for its competitive performance and fast training, and trained the networks on QuickDraw for its sufficient data.

Point Feature Dimension. The dimension d of point features introduced in Section 3.2 is a hyper-parameter of Sketch-R2CNN. We tested a range of values for d on QuickDraw, and the results are shown in Table 4. We found that using $d = 8$ offers a good tradeoff between accuracy and running time. For a smaller CNN backbone SN v2, setting $d = 8$ increases the accuracy of Sketch-R2CNN from 77.3 percent ($d = 1$) to 78.4 percent, while for a larger CNN backbone ResNet50, the configuration slightly improves the performance from 84.4 to 84.8 percent. We adopted $d = 8$ for all the experiments. Setting $d = 3$ can be an alternative choice,

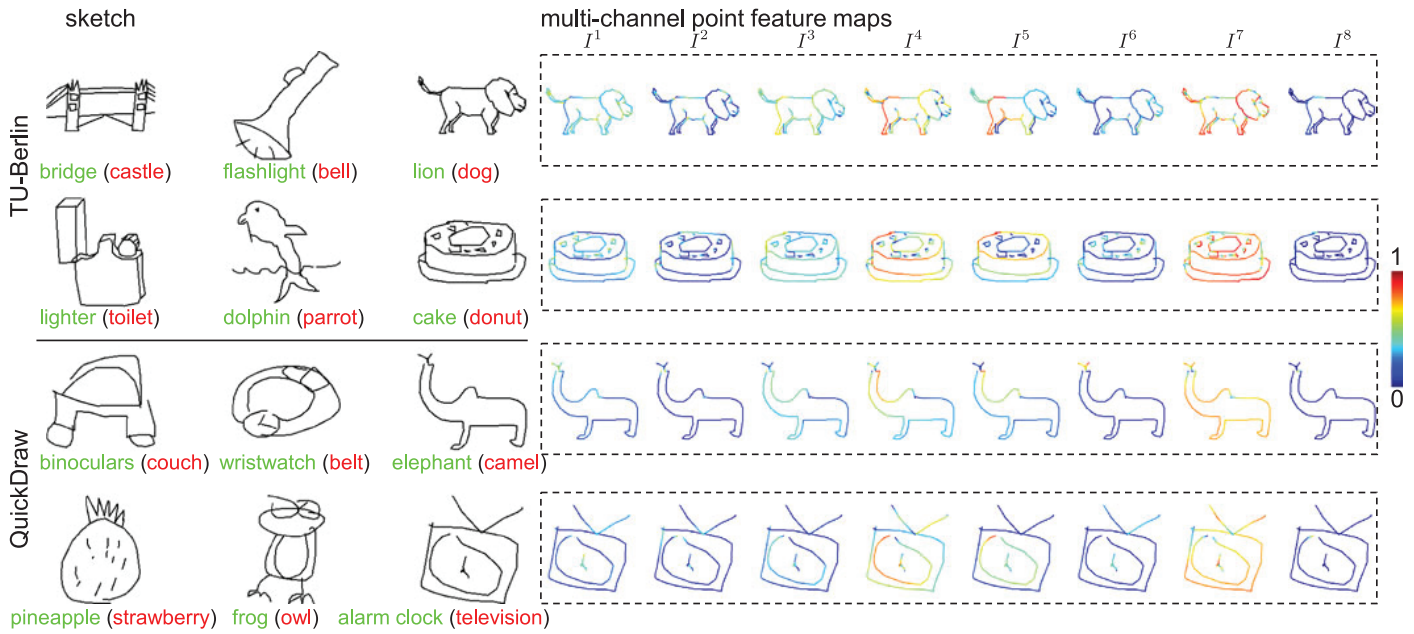

Fig. 4. Recognition samples and visualizations of point feature maps. The green labels are correct predictions by our Sketch-R2CNN (ResNet101), and the red labels in parentheses are wrong predictions by ResNet101. The multi-channel point feature maps (I^1 - I^8) produced by our RNN-Rasterization module are color-coded for visualization. Note that only stroke pixels have valid point features; non-stroke pixels are automatically set to have a feature value of zero by NLR and are not color-coded.

since it brings no modification to the existing pre-trained CNNs. What is more, for $d = 3$, if the pre-trained weights of the first CNN layer are reused, the performance of Sketch-R2CNN is 78.1 percent with SN v2 and 84.7 percent with ResNet50, respectively (Table 4); otherwise, the performance is 77.9 and 84.7 percent, respectively. Thus, reusing the first CNN layer in training for $d = 3$ only has slight influence on SN v2.

Running Time. Table 5 lists the running time of the forward (inference) and backward (optimization) passes of Sketch-R2CNN and its CNN counterparts. It is observed that the additional time incurred by our RNN-Rasterization module is 2∼3 ms for the forward pass and 5∼6 ms for the backward pass, mainly due to recurrent processing of each stroke point in the bidirectional RNN. Nevertheless, Sketch-R2CNN can run in real time, allowing integration to interactive drawing systems.

Two-Branch Late-Fusion. Different from the single branch design of Sketch-R2CNN, a two-branch late-fusion network [11] incorporates feature representations, learned by the CNN and RNN in parallel, through simple concatenation. For comparison, we followed [11] to construct a similar two-branch late-fusion network, which uses the same RNN cell and CNN backbone as Sketch-R2CNN (ResNet50). The network was trained on QuickDraw as well, and the softmax cross entropy loss was used for optimization. As shown in Table 6, the accuracy of the two-branch late-fusion network is 82.1 percent, significantly lower than the accuracy of Sketch-R2CNN (84.8 percent) by 2.7 percent (23.3K sketches). This shows that our proposed single-branch architecture allows the CNN, which works as a visual concept extractor, and the RNN, which models point features in vector sketches, to complement each other better than the two-branch architecture. Surprisingly, it is also observed that the two-branch late-fusion network achieves slightly lower accuracy than the CNN-only method (i.e., ResNet50 in Table 2). There is a gap between the reported effectiveness of the two-branch architecture in [11] and the subpar performance of our reimplementation in the experiments. Due to the lack of implementation details of [11], we postulate that some differences in data preparation and training procedure may affect the learning of feature fusion and lead to the performance degradation.

Spatial Attention. Since the output of our RNN-Rasterization module can be viewed as a form of attention (Section 2), we also performed a comparison with the spatial attention module proposed by Song *et al.* [44]. The inputs to their network (SN v2) are pixel sketches, and the soft spatial attention is computed on the feature maps of the fifth convolutional layer. We implemented this attention mechanism (DSSA in Table 6) but found that it offers limited improvement

TABLE 4
Recognition Accuracy (%) and Forward Time (ms, Batch Size = 1) of Sketch-R2CNN With Different Point Feature Dimensions (d) on QuickDraw

	w/ SN v2				w/ ResNet50			
d	1	3	8	16	1	3	8	16
acc.	77.3	78.1	78.4	78.3	84.4	84.7	84.8	84.8
time	5.1	5.2	5.4	6.1	9.4	9.5	9.6	10.5

TABLE 5
Running Time Comparisons (ms, Batch Size = 1) of the Forward (Inference) and Backward (Optimization) Passes on QuickDraw

			Ours	
	SN v2	ResNet50	w/ SN v2	w/ ResNet50
forward	2.0	7.1	5.4	9.6
backward	2.4	10.4	7.5	17.0

TABLE 6
Recognition Accuracy (%) of Alternative Design Choices and Ablation Studies on QuickDraw

Model	Accuracy
SN v2 (reproduced) [2]	74.8
ResNet50 [17]	82.5
Ours (w/ SN v2 reproduced)	78.4
Ours (w/ ResNet50)	84.8
Two-Branch Late-Fusion [11]	82.1
DSSA (SN v2) [44]	75.2
w/o Temporal Order	83.2
w/o Pen State	84.6
w/o Temporal Order + Pen State	83.0

(0.4 percent or 3.6K successes) to SN v2, which is lower than the improvement of Sketch-R2CNN (3.6 percent or 31K successes). This result further shows the difficulty of extracting additional information (attention) only in the pixel sketch space and the usefulness of vector sketches.

Contributing Factors in Vector Sketch. As confirmed by existing studies [6], [11], [19], [24] and our experiments, RNNs are capable of learning descriptive features from vector sketches. We further investigated the contributing factors in vector sketches to the point feature extraction. A vector sketch $S = \{(x_i, y_i, s_i)\}_{i=1\cdots n}$ (Section 3.1) includes positional information, temporal order and pen states. Point coordinates $\{(x_i, y_i)\}$ are clearly the most informative part for RNNs to work. In the following, we only performed ablation experiments on the temporal order and pen states.

To study the contribution of temporal order, we processed S with the following randomization scheme that disrupts order information in the vector format while preserving visual appearance in the pixel format. We consider every successive constituent points of a stroke as a tiny line segment and randomly reorder all the resulting line segments (note that NLR requires valid line segments as input (Section 3.3)). We trained Sketch-R2CNN (ResNet50) with the data, and the experiment result in Table 6 (w/o Temporal Order, 83.2 percent) shows that this scheme degrades the accuracy (84.8 percent) of Sketch-R2CNN (ResNet50) by 1.6 percent. To study the contribution of pen states, we trained the network on vector sketch inputs without $\{s_i\}$. The evaluation result (w/o Pen State in Table 6) shows that removing pen states results in minor influence (0.2 percent) on Sketch-R2CNN (ResNet50). Finally, we combined the above two ablations (w/o Temporal Order + Pen State in Table 6), and the accuracy of Sketch-R2CNN (ResNet50) drops by 1.8 percent.

We reiterate that even with the above perturbations in the vector format, our NLR does not change the appearance of the rasterized sketch, of which the subsequent CNN takes advantage for recognition. In other words, the outputs of the RNN-Rasterization module, as visualized in Fig. 4, are at least as good as the binary sketch images, and this ensures no performance degradation of the CNN (the main workhorse for recognition). It is also worth noting that we studied the contributing factors in the context of sketch recognition. In other sketch-related tasks, for example, the sketch synthesis task [19], [22], [23], pen states, together with positional information and temporal order, are all indispensable in generating realistic human drawings in the vector format.

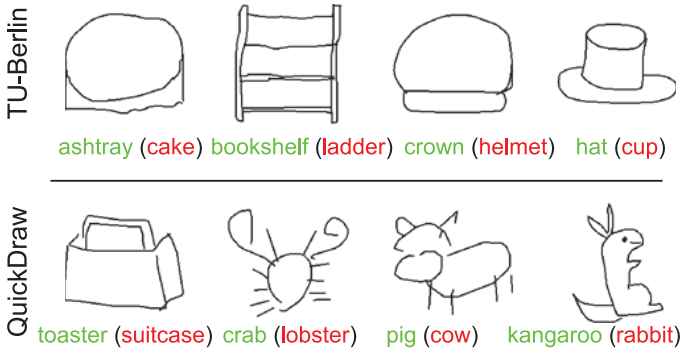

Fig. 5. Recognition failures of our method. The green labels are correct predictions by ResNet101, and the red labels are wrong predictions by Sketch-R2CNN (ResNet101).

4.4 Limitations

Fig. 5 shows some failure cases of Sketch-R2CNN. Due to the abstract and textureless nature of sketches, RNNs may fail to extract descriptive point features to guide CNNs, leading to recognition failures (e.g., the crab). Sketches with seemingly ambiguous categories (e.g., the toaster or the pig) may also pose challenges to our method. It is expected that human would make similar mistakes on such cases. One possible solution to address such ambiguity is to put the sketched objects in context (i.e., scenes), and integrate our method with context-based recognition methods [3], [34].

5 Conclusion

In this work, we have proposed a novel single-branch network architecture named Sketch-R2CNN for vector sketch recognition. Our RNN-Rasterization-CNN design allows CNNs to leverage the per-point features in vector sketches at early stages, which is enabled by a differentiable NLR module. Experiments show that Sketch-R2CNN brings consistent improvement over CNN baselines, especially on the million-scale QuickDraw dataset.

Despite the encouraging improvements on TU-Berlin, addressing the overfitting issue for fully optimizing large networks will be an important future task. Besides, our idea of in-network vector-to-pixel sketch conversion with NLR can be beneficial to other sketch-related tasks like sketch retrieval [11], sketch synthesis [22], [23] or sketch simplification [37]. For example, in the photo-to-sketch synthesis task, to generate vector sketches with more plausible spatial arrangement of strokes, it would be easier for generative networks to evaluate the plausibility in the image domain by performing the in-network vector-to-pixel conversion. The NLR module developed in this work handles differentiability with respect to point features but not positions, thus the module needs to be improved with dedicated gradient formulations for the above application. We will investigate these extensions in future work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments as well as Aleksey Nozdryn-Plotnicki for his valuable suggestions on the preliminary version of the manuscript. This work was supported

by Grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (No. HKUST 16210718, CityU 11212119), City University of Hong Kong (No. 7005176), and the Centre for Applied Computing and Interactive Media (ACIM) of School of Creative Media, CityU. Youyi Zheng was supported in part by the Fundamental Research Funds for the Central Universities and the China Young 1000 talent program.

REFERENCES

[1] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Trans. Graph.*, vol. 31, no. 4, pp. 44:1–44:10, Jul. 2012.
[2] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Sketch-a-Net: A deep neural network that beats humans," *Int. J. Comput. Vis.*, vol. 122, no. 3, pp. 411–425, May 2017.
[3] J. Zhang, Y. Chen, L. Li, H. Fu, and C.-L. Tai, "Context-based sketch classification," in *Proc. Joint Symp. Comput. Aesthetics Sketch-Based Interfaces Model. Non-Photorealistic Animation Rendering*, 2018, pp. 3:1–3:10.
[4] Z. Sun, C. Wang, L. Zhang, and L. Zhang, "Free hand-drawn sketch segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 626–639.
[5] Z. Huang, H. Fu, and R. W. H. Lau, "Data-driven segmentation and labeling of freehand sketches," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 175:1–175:10, Nov. 2014.
[6] K. Li *et al.*, "Universal sketch perceptual grouping," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 593–609.
[7] L. Li, H. Fu, and C.-L. Tai, "Fast sketch segmentation and labeling with deep learning," *IEEE Comput. Graphics Appl.*, vol. 39, no. 2, pp. 38–51, Mar./Apr. 2019.
[8] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa, "Sketch-based shape retrieval," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 31:1–31:10, Jul. 2012.
[9] F. Wang, L. Kang, and Y. Li, "Sketch-based 3D shape retrieval using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1875–1883.
[10] 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. 119:1–119:12, Jul. 2016.
[11] P. Xu *et al.*, "SketchMate: Deep hashing for million-scale human sketch retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8090–8098.
[12] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge, "Sketch-based modeling: A survey," *Comput. Graph.*, vol. 33, no. 1, pp. 85–103, 2009.
[13] R. G. Schneider and T. Tuytelaars, "Sketch classification and classification-driven analysis using Fisher Vectors," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 174:1–174:9, Nov. 2014.
[14] Y. Li, T. M. Hospedales, Y.-Z. Song, and S. Gong, "Free-hand sketch recognition by multi-kernel feature learning," *Comput. Vis. Image Understanding*, vol. 137, pp. 1–11, 2015.
[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[16] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[17] 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.
[18] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.
[19] D. Ha and D. Eck, "A neural representation of sketch drawings," in *Proc. Int. Conf. Learn. Representations*, 2018.
[20] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850
[21] Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Sketch-a-Net that beats humans," in *Proc. Brit. Mach. Vis. Conf.*, 2015, pp. 7.1–7.12.
[22] J. Song, K. Pang, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Learning to sketch with shortcut cycle consistency," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 801–810.
[23] C. Nan, Y. Xin, S. Yang, and C. Chaoran, "AI-Sketcher: A deep generative model for producing high quality sketches," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 2564–2571.
[24] J. Collomosse, T. Bui, and H. Jin, "LiveSketch: Query perturbations for guided sketch-based visual search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2874–2882.
[25] H. Kato, Y. Ushiku, and T. Harada, "Neural 3D mesh renderer," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3907–3916.
[26] O. Russakovsky *et al.*"ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[27] E. Yanık and T. M. Sezgin, "Active learning for sketch recognition," *Comput. Graph.*, vol. 52, pp. 93–105, 2015.
[28] C. Alvarado and R. Davis, "SketchREAD: A multi-domain sketch recognition engine," in *Proc. 17th Annu. ACM Symp. User Interface Softw. Technol.*, 2004, pp. 23–32.
[29] J. J. LaViola, Jr, and R. C. Zeleznik, "MathPad2: A system for the creation and exploration of mathematical sketches," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 432–440, Aug. 2004.
[30] T. Y. Ouyang and R. Davis, "ChemInk: A natural real-time recognition system for chemical drawings," in *Proc. 16th Int. Conf. Intell. User Interfaces*, 2011, pp. 267–276.
[31] T. Lu, C.-L. Tai, F. Su, and S. Cai, "A new recognition model for electronic architectural drawings," *Comput.-Aided Des.*, vol. 37, no. 10, pp. 1053–1069, 2005.
[32] T. M. Sezgin and R. Davis, "Sketch recognition in interspersed drawings using time-based graphical models," *Comput. Graph.*, vol. 32, no. 5, pp. 500–510, 2008.
[33] R. Arandjelović and T. M. Sezgin, "Sketch recognition by fusion of temporal and image-based features," *Pattern Recognit.*, vol. 44, no. 6, pp. 1225–1234, 2011.
[34] C. Zou *et al.*, "SketchyScene: Richly-annotated scene sketches," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 438–454.
[35] H. Zhang, S. Liu, C. Zhang, W. Ren, R. Wang, and X. Cao, "SketchNet: Sketch classification with web images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1105–1113.
[36] Y. Chen, S. Tu, Y. Yi, and L. Xu, "Sketch-pix2seq: A model to generate sketches of multiple categories," *CoRR*, vol. abs/1709.04121, 2017. [Online]. Available: http://arxiv.org/abs/1709.04121
[37] U. Riaz Muhammad, Y. Yang, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Learning deep sketch abstraction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8014–8023.
[38] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2204–2212.
[39] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, "The application of two-level attention models in deep convolutional neural network for fine-grained image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 842–850.
[40] F. Wang *et al.*, "Residual attention network for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 6450–6458.
[41] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7132–7141.
[42] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2048–2057.
[43] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3242–3250.
[44] 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.*, 2017, pp. 5552–5561.
[45] M. Li, Z. Lin, R. Mech, E. Yumer, and D. Ramanan, "Photo-Sketching: Inferring contour drawings from images," *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2019, pp. 1403–1412.
[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[47] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.
[48] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun, "Bayesian face revisited: A joint formulation," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 566–579.
[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980