SketchINR: A First Look into Sketches as Implicit Neural Representations

Hmrishav Bandyopadhyay[1] Ayan Kumar Bhunia[1] Pinaki Nath Chowdhury[1] Aneeshan Sain[1]
Tao Xiang[1,2] Timothy Hospedales [3] Yi-Zhe Song[1,2]
[1]SketchX, CVSSP, University of Surrey, United Kingdom.
[2]iFlyTek-Surrey Joint Research Centre on Artificial Intelligence.
[3]University of Edinburgh, United Kingdom
{h.bandyopadhyay, a.bhunia, p.chowdhury, a.sain, t.xiang, y.song}@surrey.ac.uk
t.hospedales@ed.ac.uk

(a) Self-Reconstruction (b) Complexity Scaling

Figure 1. SketchINR is an implicit neural representation for sequential vector sketches. It is the first neural representation with sufficient fidelity to be a drop-in replacement for the raw sketch data (a). The representation is significantly higher fidelity than existing learned representations such as SketchRNN [28], especially for more complex sketches (b, left). SketchINR also provides state-of-the-art sketch compression, with a substantially more compact representation than either vector or raster sketches (b, right).

Abstract

We propose SketchINR, to advance the representation of vector sketches with implicit neural models. A variable length vector sketch is compressed into a latent space of fixed dimension that implicitly encodes the underlying shape as a function of time and strokes. The learned function predicts the xy point coordinates in a sketch at each time and stroke. Despite its simplicity, SketchINR outperforms existing representations at multiple tasks: (i) Encoding an entire sketch dataset into a fixed size latent vector, SketchINR gives $60\times$ and $10\times$ data compression over raster and vector sketches, respectively. (ii) SketchINR's auto-decoder provides a much higher-fidelity representation than other learned vector sketch representations, and is uniquely able to scale to complex vector sketches such as FS-COCO. (iii) SketchINR supports parallelisation that can decode/render $\sim 100\times$ faster than other learned vector representations such as SketchRNN. (iv) SketchINR, for the first time, emulates the human ability to reproduce a sketch with varying abstraction in terms of number and complexity of strokes. As a first look at implicit sketches, SketchINR's compact high-fidelity representation will support future work in modelling long and complex sketches.

1. Introduction

The prevalence of touch-screen devices has triggered significant research progress on sketches [4, 6, 8, 51, 57]. Modelling digital sketches has become an important challenge for learning systems that aim to stimulate human creativity [17, 27]. Large-scale datasets [24, 28, 53] motivated several downstream applications like image retrieval [15, 61], image generation [37, 65], image editing [1, 41], and 3D content creation [5, 8, 38], among others [58, 60].

These applications are underpinned by sketches captured in raw form either as raster or vector images, and usually encoded into derived representations by ConvNets [15], splines, and so on. For over a decade, discourse around what is a "good" representation of human sketches has persisted [31]. A now substantial body of work has focused on representations for sequential vector sketches that model both explicit strokes, and their drawing over time — most famously by the auto-regressive SketchRNN [28], but also using representations such as parametric curves [16], Gaussians [2], and Neural ODEs [17]. We introduce a novel continuous-time representation of sequential vector sketches by taking an implicit neural representation perspective on sketches for the first time.

First, we review the limitations of current sketch representations. The two most popular raw sketch representations are (i) raster sketch – a black and white image of a line drawing, and (ii) vector sketch – a temporal sequence of points and strokes. Raster sketch has a large but fixed storage (*e.g.*, 256×256). Its compatibility with ConvNets [64] made raster sketches popular but they lack the tempo-

ral information (i.e., stroke order) necessary for generative modelling [17]. Vector sketches efficiently store the xy-coordinates of points and strokes ($N \times 2$). However, the storage increases with the length of sketch N (Fig. 1b). (iii) Raw vector sketches can be encoded by popular learned sketch representations such as SketchRNN [28] and others [16, 17, 39]. However, for longer ($N \geq 1000$) scene sketches, the quadratic computational cost for transformers becomes intractable [39] and recurrent nets become unstable [28]. Nevertheless, most of these still suffer from slow sequential autoregressive decoding, and they mostly cannot accurately reconstruct complex sketches with many strokes. To summarise, current sketch representations either discard temporal information (raster), scale poorly in storage size (vector), or become slow and inaccurate with complex sketches (learned autoregressive vector).

To overcome the above issues, we propose an implicit neural representation for sketches – SketchINR. We piggyback on the well-established [8, 30, 34, 48] potential of implicit functions [22] to encode an underlying shape as a function of time/location. Different to standard implicits [43, 48], the input of sketches as parametric functions is a hierarchy of stroke sequence and point sequence. We develop SketchINR to generate points (xy-coordinates) given both time and stroke inputs. To represent multiple sketches, SketchINR learns a fixed size latent space and a conditional implicit function that generates xy-coordinates of any sketch instance given its corresponding latent code.

Despite its simplicity, SketchINR is competitive with existing sketch representations. (i) It can encode an entire sketch dataset into a compact latent space (*e.g.*, $\mathbb{R}^{N\times 512}$) and decode using a fixed size function (*e.g.*, an 8 layer MLP). Compared to raster sketches (256×256), SketchINR provides $128\times$ storage compression. While simple object-level vector sketches (length ≤ 300) have comparable storage with SketchINR ($\mathbb{R}^{512}$), for practical scene-level [13] vector sketches (length ≥ 1000), SketchINR provides $10\times$ better compression (Fig. 1b). (ii) For decoding/generation, SketchINR can parallelly predict the xy-coordinates for all time and strokes. In practice, this is $100\times$ faster than an autoregressive vector sketch generator with length ≥ 300. (iii) SketchINR can accurately represent complex sketches (Fig. 1a,b) unlike competitors that suffer from complexity limitations [16, 17] or the length limitations of auto-regression [28]. The encoding fidelity is sufficiently high that SketchINR provides a drop-in replacement for raw sketch data, while being more compact. (iv) SketchINR supports diverse applications spanning smooth latent space interpolation, and sketch generation, sketch completion. Uniquely, it also supports sketch abstraction – the human-like ability to replicate the essence of a sketch with a variable number and complexity of strokes [31, 54].

In summary, our contributions are: (i) We present the first implicit neural representation approach to vector sketch modelling, extending INRs as a function of time and strokes, and defining a training objective for learning them. (ii) SketchINR provides the first learned representation capable of representing complex sketches compactly and with high-fidelity (Fig. 1a). Based on our compact high-fidelity representation we introduce the task of sketch compression, and demonstrate excellent compression results (Fig. 1b)). (iii) We further demonstrate SketchINR's applicability to a variety of sketch related tasks including generation, interpolation, completion and abstraction.

2. Related Work

Sketch Representations: Digital sketches are predominantly captured as a function of 'time' [28], in a *sequence* of coordinates traced by an artist on a canvas. Recent works [24, 28] emphasise this temporal order of coordinates as an indicator of their relative importance in depicting a sketch-concept, as humans draw in a coarse-to-fine fashion. As such, they [2, 7, 18, 28] exploit temporal information in sketches for downstream tasks like sketch-assisted retrieval [7], generation [2, 18, 28] and modelling [38]. Vector sequences are further parameterised, for representation with parametric curves like Béziers [16, 57] and B-Splines [50, 66] for early-handwritten digit representation [50] to recent representations of complex structures like real-world objects [16, 57] and scenes [58]. Discarding temporal significance of strokes, vector-sketches can be converted to raster images by rendering coordinate strokes [9] on a 2D canvas. While raster sketches are less informative than their vector counterparts [28], they offer enhanced applicability by expanding beyond digital sketches to *paper*-sketches [18] where stroke order is already lost. Raster representations are encoded with translation-invariant networks like CNNs [63] and Vision-Transformers [8], making them extremely useful to represent object-level information.

Non-Parametric Modelling: Both vector sequences and raster renderings of these sequences are non-parametric representations [16] of sketches. To preserve temporal stroke order, vector sketches are handled as sequences [28] of coordinates rather than unordered sets. Hence, they are encoded with position-aware networks like RNNs [28], LSTMs [15, 61], and Transformers [10, 51]. Without positional cues, raster sketches are processed by spatially-aware CNNs [63] and Vision Transformers [52]. Semantic encodings of non-parametric representations capture the expressive nature of sketch, allowing its use as a creative input for downstream segmentation [33], retrieval [7], and editing [41], as well as for interactive tasks like object-detection [14] and image-inpainting [62]. These encodings are further used in auto-regressive [28, 59] and VAE-like [10] modelling of vector coordinate sequences for the generation of textual-characters [1] and object sketches [28].

Parametric Representation learning: Parametric Splines [20], such as Beizer curves [42], approximate vector sketches by per-sample fitting on individual sketch samples. Amortised frameworks bypass this per-sample optimisation by inferring curve control points [16] and degree [18] from individual stroke features. Recent works exploit the representative power of parametric curves for generation of simple characters [26] to complex object [16, 18, 57] and scene level sketches[58]. Both non-parametric and curve-fitted Bézier parametric representations capture the *explicit* form of a sketch as spatial and sequential forms.

Beyond explicit representations of sketches, we learn visual implicits to capture spatial dynamics in vector-sketches. Learned implicits can be used for sketch reconstruction with particular control over abstraction through *number* of reconstructed strokes. Somewhat similar to implicit representations for vector sketches, CoSE [2] parameterises Gaussian Mixtures on individual strokes, learning one implicit for one stroke only. New implicits are generated auto-regressively from previous predictions for auto-regressive generation and completion of sketches. Unlike CoSE, we learn a single implicit for the entire sketch, thus having one implicit parameter to sample the sketch in one pass. While SketchODE [17] similarly parameterises Neural Ordinary Differential Equations [12] to capture vector-sketch dynamics, it's optimisation is extremely slow (as much as $120\times$ slower) because of high time complexity in solving higher order differentials. Three dimensional implicits like Signed-Distance [48] and Occupancy [43] functions learned from 3D point-clouds are particularly analogous to our implicit representation.

Sketch Abstraction: The continuity of human cognition in visual perception [36] is directly reflected in how we sketch. This enables versatility [64] in sketch, allowing us to express a huge range of thoughts and visions directly on paper in the form of coarse-grained 'ideas' [32] and fine-grained 'objects' [64]. Sketch abstraction, as a direct inverse of this granularity, has been studied in depth as a function of (i) drawing time: humans draw most representative strokes first [28], and (ii) compactness: in a constrained stroke setting, salient strokes are prioritised [46]. As abstraction gives sketches a human touch, generative modelling of sketches is focused on abstraction towards making generated drawings "more humane" [19] and augmenting limited sketch datasets with simulated sketches from photos [57, 58]. Importantly, recent works [57] model abstraction as a function of *strokes* to explicitly control abstraction levels by restricting the number of generated strokes. Here, we represent sketches as visual implicits which allows us to reconstruct them at varying levels of abstraction and detail by controlling the number of reconstructed strokes. We demonstrate several downstream applications of this representation, particularly sketch compression, generation and completion.

3. Methodology

Overview: Vector sketches are captured as pen movements on a digital canvas, represented by coordinates $(x, y) \in [0, 1]$ with binary pen-states $\delta \in \{0 : \text{pen-down}, 1 : \text{pen-up}\}$ at those coordinates. A sketch $p = \{(x_i, y_i, \delta_i)\}_{i=1}^{N}$ comprises N vector *way-points* (coordinates), and is portrayed (rendered) by retracing these points on a 2D canvas. This process, termed as rasterisation produces a pixelated raster sketch p^s on the canvas. In this work, we leverage scalability [48] in implicit functions to learn a *flexible* representation for vector sketches with raster guidance.

3.1. Learning Visual Implicits for Sketches

Vector sketches can be modelled as implicits, where a sketch is represented as function of time $f_\theta(t_j) : \mathbb{R} \rightarrow \mathbb{R}^{2+1}$ parameterised by θ. Each time-stamp $t_j \in [0, 1)$ is mapped to a pen position in the form of xy-coordinates $(x, y) \in \mathbb{R}^2$ and binary pen-states $\delta \in \mathbb{R}^1$. The entire sketch can be reconstructed as a vector sequence by sampling t from 0 to 1 with a learned θ as $\mathbf{p} = \{f_\theta(0), \ldots, f_\theta(t_j), \ldots f_\theta(\frac{J-1}{J})\}$ for a total of J timestamps. This in itself is a flexible representation for a vector sketch, as it allows for sampling with arbitrary timestamps, thus potentially increasing or decreasing resolution of the resulting sketch by modulating number of strokes with sampling resolution ($t : \{0, \frac{1}{3}, \frac{2}{3}\}$ v/s $t : \{0, \frac{1}{10}, \ldots, \frac{9}{10}\}$). For increased control over reconstruction granularity in the form of abstraction, we model f_θ on an additional variable in the form of strokestamps $\{s_k\}_{k=1}^{K}$, representing the sketch as $\hat{p} = \{f_\theta(0, 0), \ldots, f_\theta(t_j, s_k), \ldots, f_\theta(\frac{J-1}{J}, \frac{K-1}{K})\}$ for K strokes (Fig. 2). These strokestamps provide explicit control over pen-up and pen-down states, allowing us to reconstruct the sketch with re-defined granularity during inference (*e.g.* 7 or 9 strokes). Specifically, strokestamps represent the current stroke number as a fraction of the total number of strokes K which allows us to change strokes by changing s_k. Pen-states δ are thus explicitly determined from strokestamps only by toggling when s_k changes value. Our practical implementation of sketch implicits follows:

$$f_\theta(t_j, \Delta t, s_k, \Delta s) = \mathrm{p}_j \tag{1}$$

where, time instances are computed as $t_j = t_{j-1} + \Delta t$, $t_0 = 0$, and $\Delta t = \frac{1}{J}$. Similarly, K-stroke instances are computed as $s_k = s_{k-1} + \Delta s$, $s_0 = 0$, and $\Delta s = \frac{1}{K}$. To exploit bias in neural networks towards learning low frequency functions [44, 49], we smoothen out high frequency variations $(0\text{to}\frac{1}{K}, 0\text{to}\frac{1}{S})$ for timestamps and strokestamps by mapping them from $\mathbb{R}$ to a higher dimension $\mathbb{R}^{2L}$ using positional embeddings $\mathrm{PE}(\cdot)$ as

$$\begin{aligned}\mathrm{PE}(x) = &[\sin(10^{-4/L}x), \cos(10^{-4/L}x), \\ &\ldots, \sin(10^{-4L/L}x), \cos(10^{-4L/L}x)]\end{aligned} \tag{2}$$

for $x \in \{t_n, \Delta t, s_k, \Delta s\}$. We uniformly distribute J timestamps across K strokes, where each stroke has J/K points. For data augmentation, we train f_θ with varying J and K ($\pm 50\%$ of ground truth value). This helps f_θ to scale to different levels of abstraction, and number of points and strokes during inference.

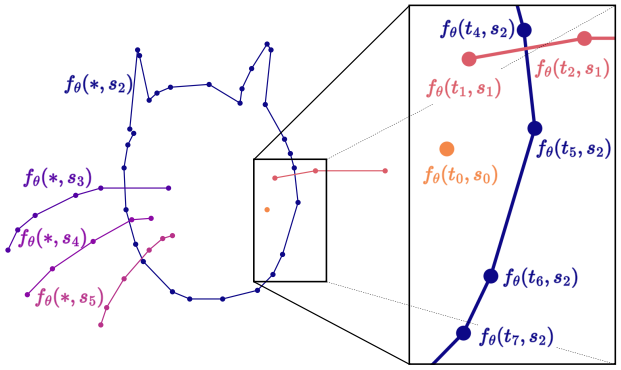

Figure 2. We model sketch as a function $f_\theta(t_j, s_k)$ of J timestamps and K strokes. Timestamps $\{t_a, \ldots, t_b\}$ where $t_j \in [0, 1)$ correspond to *way-points* for a specific stroke s_k. Finally, f_θ with learned weights θ can be sampled with arbitrary number of strokes as $t_j \in \{0, \frac{1}{J}, \ldots, \frac{J-1}{J}\}$ and $s_k \in \{0, \frac{1}{5}, \ldots, \frac{4}{5}\}$ or $s_k \in \{0, \frac{1}{10}, \ldots, \frac{9}{10}\}$, leading to increasing or decreasing abstraction for $K = 5$ or $K = 10$, respectively.

3.2. Loss functions

To fit our implicit function f_θ to a particular sketch instance $p = \{(x_0, y_0, \delta_0), \ldots, (x_{M-1}, y_{M-1}, \delta_{M-1})\}$ with M vector way-points $\{p_m\}_{m=0}^{M-1}$, a naive approach would be to optimise θ with mean squared error against the ground truth vectors as:

$$\mathcal{L}_{\text{MSE}} = \sum_{m=0}^{M} \left|\left| f_\theta\left(\frac{m}{M}, \frac{k}{K}\right) - p_m \right|\right|_2 \tag{3}$$

where each predicted coordinate and pen-state is optimised directly against corresponding vector way-points from the ground truth. However, this rigid point-to-point matching enforces memorisation of particular points and strokes without any spatial understanding. As such, sampling with a different resolution at this stage yields noisy output that does not correspond to the raster sketch. To explicitly optimise for sampling flexibility, we introduce a visual loss, penalising sketch reconstructions that do not match visually. For this, we sample a template grid G of 2D coordinates and compute the region of influence of each vector-waypoint p_m of stroke s_k on coordinate g as an intensity map [3, 59]:

$$\mathrm{I}_k(g, s_k) = \max_{p_m \in s_k, r \in [0,1]} \exp(\gamma ||g - r p_m - (1-r) p_{m+1}||_2) \tag{4}$$

where intensity I_k is smoothed with γ for points p_m in stroke s_k where $m \in \{0, \ldots, \frac{K-1}{K}\}$. The intensity map for the entire sketch is formulated as a summation of maps from all strokes for $k \in (0, 1]$ for both the ground truth sketch vector p with K strokes and the arbitrarily sampled sketch $\hat{p}$ with $\hat{K}$ strokes as $\hat{\mathrm{I}}_k$, composing the loss as:

$$\mathcal{L}_V = \left|\left| \sum_{k=0}^{\frac{K-1}{K}} \mathrm{I}_k - \sum_{k=0}^{\frac{\hat{K}-1}{\hat{K}}} \hat{\mathrm{I}}_k \right|\right|_2 \tag{5}$$

Intensity Smoothing with γ: The sketch intensity map I_k helps us to ground vector sketches to raster definites (Eq. (4)) by a visual rendering of individual strokes where intensity in coordinates $g \in G$ drops exponentially with distance from $p_m \in s_k$. This exponential drop is controlled by a smoothing factor γ as a hyperparameter, where (i) a low γ yields thicker lines on I_k (low-quality sketch reconstructions) but offers better optimisation, while (ii) a high γ yields more refined lines (higher quality reconstructions), making optimisation significantly slower (Fig. 3).

For faster optimisation, we assist the visual penalty with a weighted MSE loss from Eq. (3) and a lower gamma, empirically set to $\gamma = 150$. The loss function weighted with λ_{MSE}, looks can be written as:

$$\mathcal{L}_{\text{implicit}} = \mathcal{L}_V^{\gamma=150} + \lambda_{\text{MSE}} \cdot \mathcal{L}_{\text{MSE}} \tag{6}$$

where we set λ_{MSE} to 0.7. This, however, inherently introduces bias in our optimisation to match the order of strokes with the ground truth order to some extent. Future works could ideally set $\lambda_{\text{MSE}} = 0$ if they obtain faster convergence from a better engineered implicit function f_θ. Fig. 4a summarises our implicit sketch learning framework.

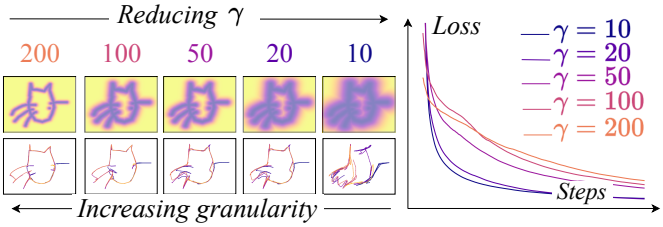

Figure 3. Effect of smoothing factor γ on training: Reducing γ leads to higher intensity in the surrounding region near stroke s_k. This is similar to stroke dilation of intensity map I_k. A lower γ leads to stable training (plot on right) but lacks fine-grained details. A higher γ gives a fine-grained sketch but is harder to train.

3.3. Generalising to Multiple Sketches

To efficiently represent multiple sketch implicits with a single global and general function, we modify Eq. (1) to include a sketch descriptor $\nu_i \in \mathbb{R}^d$ for each sketch i. The function $f_\theta(\nu_i, t, \Delta t, s_k, \Delta s)$ is approximated with a fully connected auto-decoder network, similar to DeepSDF [48]. Thus, feature descriptors ν_i are hidden vectors, learned

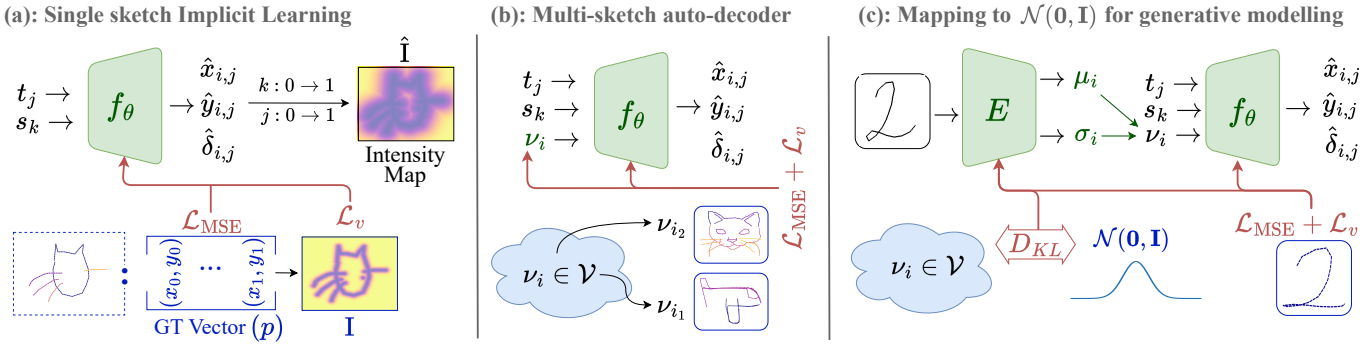

Figure 4. SketchINR model diagram. *(a)* Embedding a single vector sketch as an implicit model F mapping stroke and time $(\mathbf{s}, \mathbf{t})$ to ink coordinate $\mathbf{p}$. *(b)* Embedding a vector sketch dataset as a shared decoder and set of latent vectors $\mathcal{V}$. *(c)* Training a generative model for implicit sketches by generating latent codes ν using an encoder E that inputs raster sketches.

jointly with the function parameters θ on a training dataset ($i \in D$). Similar to single sketch optimisation, we train on multiple samples with a combination of MSE and visual loss (Eq. (6)). Fig. 4b summarises our multi-sketch auto-decoder through block diagrams.

Generative Modelling: The generalised auto-decoder, enables simple generative modelling of sketches by generating their feature descriptors $\nu_i \in \mathcal{V}$, which allows us to create novel sketch implicits. Towards this, we train a Variational Auto-Encoder [35] as our generator, where we replace the decoder with our pre-trained implicit decoder. For variational inference, our encoder $E(\cdot)$ represents each raster sketch instance i as a set of d Gaussians with mean $\mu_i \in \mathbb{R}^d$ and variance $\sigma_i \in \mathbb{R}^d$ which are re-parameterised [35] to a latent encoding as $z_i = \mu_i + \sigma_i \cdot \epsilon$ where $\epsilon \in \mathcal{N}(\mathbf{0}, \mathbf{I})$. This latent is then projected to the d-dimensional implicit latent space $\nu_i \in \mathcal{V}$. We train the encoder as a ResNet-18 network with (i) reconstruction loss from both our visual and vector spaces (Eq. (6)) and (ii) KL-divergence loss to bring the distribution closer to a unit normal $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The loss function for the encoder can be written as:

$$\mathcal{L}_{\text{enc}} = \mathcal{L}_{\text{implicit}} - \beta \cdot D_{KL}(q(z_i|i), \mathcal{N}(\mathbf{0}, \mathbf{I})) \qquad (7)$$

where β is a weighing factor, set to 0.7 and $q(z_i|i)$ represents the underlying probability distribution modelled with encoder E that infers z_i for a given sketch i. Fig. 4c summarises our implicit sketch generative model. The decoder can be used for sketch generation, while the encoder can encode raster sketches for vectorisation.

4. Applications

Datasets: We learn implicits for sketches of various complexities and abstraction levels, ranging from highly complex scene sketches that correspond to a photo scenery [13] to abstract doodles [28] drawn from memory. Based off complexity in downstream applications, we use scene sketches to demonstrate self-reconstruction and sketch compression, while focusing on simpler object level sketches for more complex tasks like sketch mixing (interpolation) and generation for faster optimisation. Specifically, we use the *FS-COCO* [13], *Sketchy* [53], and *Quick-Draw!* [28] datasets. (i) *FS-COCO* [13] contains 10,000 hand-drawn sketches of complex MS-COCO [40] scenes, drawn by amateurs without time constraints. Sketches in *FS-COCO* have on a median of 64 strokes, resulting in the sketch having ∼3000 waypoints. (ii) Comparatively simpler than scene sketches, *Sketchy* [53] consists of ∼75K sketches of 12.5K object photographs from 125 categories. These sketches similar to *FS-COCO* are drawn from reference photographs and have photo-sketch pairs. (iii) *Quick-Draw!* [28] sketches are drawn from memory and in a constrained time settings ($<$ 20s). Hence, these sketches (doodles) are abstract in nature, lacking any pre-defined configuration or orientation from reference photos. In addition to these three datasets, we use a vectorised form of the MNIST dataset, *Vector-MNIST* [17], consisting of ∼10K samples as a simpler dataset to demonstrate complex downstream applications like generation and interpolation.

4.1. SketchINR: A Compact High-Fidelity Codec

We begin by demonstrating that SketchINR provides a high fidelity neural representation for sketches. Specifically, we train SketchINR on complex variable length vector sketch datasets such as *FS-COCO* and *Sketchy*. We represent each sketch in these datasets as a fixed size vector with a dataset-specific decoder f_θ. We then render each implicit sketch $\nu_i \in \mathcal{V}$ with the same number of strokes as the ground-truth sketch $\mathbf{p}^i$ to evaluate reconstruction quality. Following [17, 28], we measure reconstruction quality using (i) Chamfer Distance (CD), and (ii) retrieval accuracy (top-10) with a naive sketch retrieval network (ResNet-18) trained on ground-truth sketch vectors $\mathbf{p}$ and evaluated on rendered $\hat{\mathbf{p}}$. The quantitative results in Tab. 1, show that SketchINR provides substantially higher fidelity encoding than alternative representations like RNN [28], Bézier strokes [16],

	FS-COCO [13]			Sketchy [53]		
	CD	R@10	Time (s)	CD	R@10	Time(s)
SketchRNN [28]	–	–	–	0.16	41.65	0.11
CoSE [2]	–	–	–	0.14	45.82	0.10
SketchODE [17]	0.41	5.17	1.68	0.17	37.20	0.13
BézierSketch [16]	0.57	2.96	1.22	0.23	25.72	0.08
Ours	0.011	15.61	0.3	0.008	57.29	0.0007

Table 1. Evaluating neural sketch representations' reconstruction quality (Chamfer distance ↓; recall@10 ↑) and decoding speed.

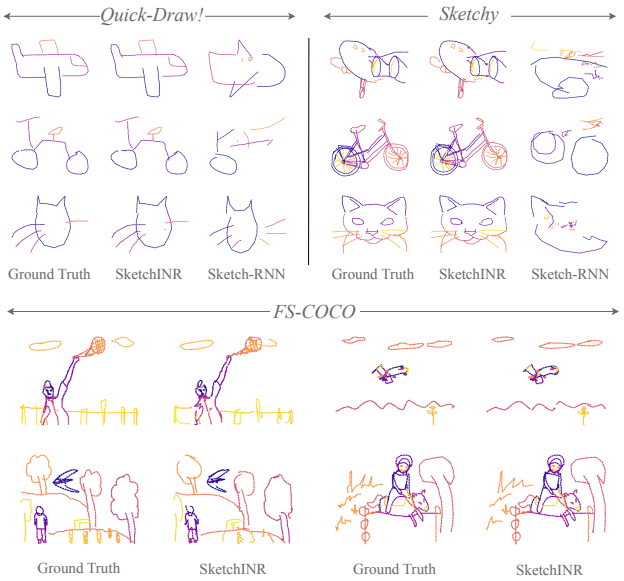

Figure 5. Qualitative reconstruction results for neural sketch representations. SketchINR is uniquely able to scale to sketches of *Sketchy* and *FS-COCO* complexity.

ODE dynamics [17], and stroke embeddings [2] particularly for extremely complex sketches in *FS-COCO*, where SketchRNN and CoSE fail entirely. Qualitative results in Fig. 5 show SketchINR to reconstruct simple and complex sketches with high fidelity while SketchRNN performs poorly for *Sketchy* and fails completely on *FS-COCO*.

Sketch Compression: We next show that SketchINR's high-fidelity encoding is extremely compact, thus providing an efficient sketch codec. INRs store information implicitly [55] in the network weights (θ) and latent codes (ν_i). To store or transmit a sketch dataset, we only stream weights θ once (fixed cost) and a compressed latent vector $\nu_i \in \mathbb{R}^D$ for each sketch (variable cost). Hence, the space complexity for N_D sketches with SketchINR is $|\theta| + N_D \times |\nu_i|$ bytes, where ν_i is 64 dimensional latent. For streaming a large number of sketches, we discount θ as negligible streaming overhead. Comparing to alternative representations, streaming N_D (i) binary raster (BR) sketches of size $(N_x \times N_y)$ take $N_D \times N_x \times N_y$ bits , and (ii) vector scene sketches with N coordinates and pen states ($\mathbb{R}^{2+1}$) take $N_D \times N \times 2 \times 3$ bytes in 16-bit floating precision ($N \approx 3000$ in *FS-COCO* [13]). Raster representations can be further stored by only storing the coordinates ($\mathbb{R}^2$) of pixels containing the sketch (*i.e.*, inked pixels) as sparse binary rasters (SBR). Fig. 6 shows rate-distortion for storing $10,000$ scene sketches [13] with ν_i varying from 128 to 8 dimensional latent, and quality for raster and vector forms dropped with downsampling and RDP simplification [23] respectively. Quality metrics like PSNR and SSIM are more suited to photos. We use Chamfer Distance as a quality metric for vector sketch and observe $\sim$10$\times$ and $\sim$60$\times$ more efficiency than vector and raster sketches, respectively. Future works can further optimise our models and thus reduce storage requirements via model compression [21, 29, 56] and quantisation [11].

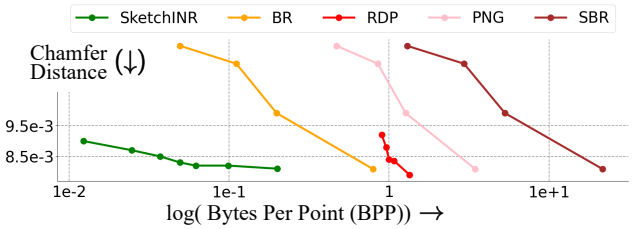

Figure 6. Rate Distortion: SketchINR can encode complex scene sketches from *FS-COCO* [13] in highly compact ($\mathbb{R}^{64}$) latent codes. Specifically, despite nearly identical sketch quality (low CD represents higher fidelity), SketchINR has $\sim$60$\times$ lower BPP than PNG raster sketches and $\sim$10$\times$ lower than vectors sketches.

Intra-Sketch Variation: Unlike prior representations (RNN [28], ODE dynamics [17]), humans do not draw exact stroke-level replica of a raster sketch. Instead, we choose the level of detail, the number of points, and the strokes used to convey the underlying shape. Given a latent code ν_i, *SketchINR* allows users to explicitly choose N and K, where the number of points $p = \{p_1, \ldots, p_N\}$ and strokes $s = \{s_1, \ldots, s_K\}$. Choosing a low N and K leads to abstract self-reconstructions with fewer points and strokes. Fig. 7(top to bottom) shows that we can control K to decrease or increase the level detail in the sketch.

4.2. Latent Space Interpolation (Creative Mixing)

Interpolation of learned latents allows us to explore latent space continuity. Auto-regressive modelling of sketches builds poor latents as explored by Das *et. al.* [19] due to the lack of a full visibility of the sketch at any decoding timestep. Specifically, auto-regressive models use the partially predicted sketch along with the sketch's latent representation to predict new way-points with consistency. This, however, introduces noise in the latent → sketch decoding so that *small changes in these latents lead to big changes in the final sketch*. Deterministic latent to sketch decoding solves this problem as noise is not added at any point, leading to meaningful sketch interpolation from corresponding latent space interpolations. We visualise in

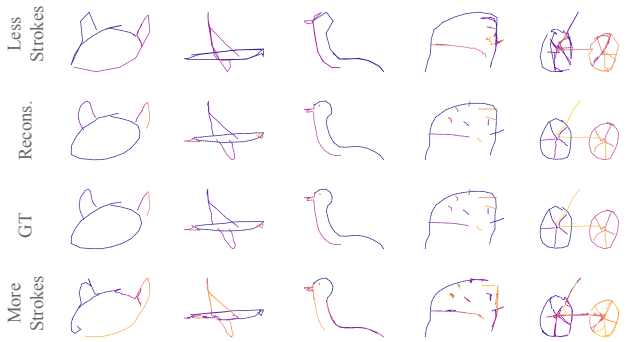

Figure 7. Given a latent code ν_i, we can reconstruct a sketch using learned f_θ at multiple resolutions by controlling the number of strokes K during sampling.

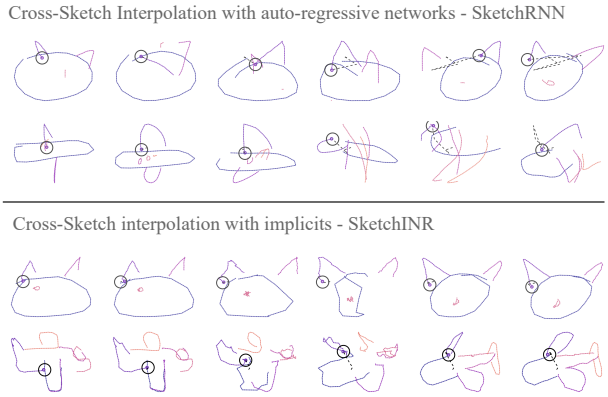

Figure 8. Latent space interpolation shows continuity and translation from one sketch concept to another.

Fig. 8 how one sketch morphs into another by reconstructing from $f_\theta(\nu', t_n, \Delta t, s_k, \Delta s)$, where ν' is sampled along a *latent walk*: $\nu' = (1-\delta)\nu_1 + \delta\nu_2$, for $\delta \in [0, 1]$. Interpolating between two sketches in the continuous latent space gives creative mixing – modifying the sketch of airplane-1 (facing right) into airplane-2 (facing left) by varying δ. Fig. 8 shows that small changes in the latent (columns) leads to smoother changes in sketch space for SketchINR. Meanwhile SketchRNN can exhibit big jumps that can zig-zag over successive latent increments, as indicated by an example tracked point in the figure.

4.3. INR inversion for sketch completion

Neural networks used to approximate implicit representations have been inverted in parallel literature [30] to appropriate samples not seen during training. This inversion, to arrive at a latent ν that describes the new sample, is usually made possible by learned priors in f_θ. Specifically, an implicit decoder f_θ is frozen with latent ν optimised for $f_\theta(\nu, \dots)$ against the ground truth. This behaviour is easily replicated in SketchINR, where we optimise the latent ν to describe sketch samples (Fig. 5). To implement

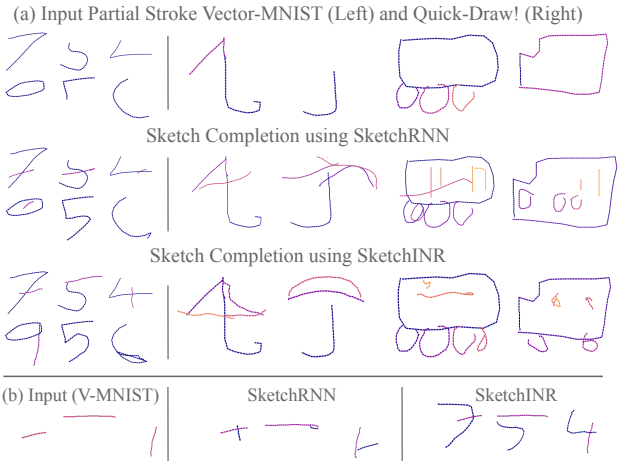

Figure 9. We perform sketch completion by inverting learned implicit representations and decoding for occluded strokes or waypoints. (a) Temporal completion (b) Unlike auto-regressive methods like SketchRNN [28], we can perform a-temporal (or inverse-temporal) completion, where given the second half of a partial sketch, we can successfully reconstruct the first part.

sketch completion, we obtain a sketch descriptor ν from a partial sketch by just optimising for specific time-steps (*e.g.* 1/4th of a sketch; $t \in [0, \frac{1}{4})$) available in a partial doodle. Then, by sampling for rest of the timesteps we can complete the incomplete sketch. We summarise some qualitative sketch completion results in Fig. 9(a). Completed sketches are varied because of randomness in optimisation, as partial sketches are often ambiguous. INR inversion thus provides a novel approach to sketch completion, previously dominated by auto-regressive and models. Unlike auto-regressive models SketchINR can also perform *a-temporal* completion, inferring the first half of a sketch given the second half of the sketch (Fig. 9b). We remark that popular INR applications such as super-resolution [47] and point-cloud densification [48] perform interpolation after inversion on sparse inputs (*e.g.*: $t \in [0, 1)$ given $t \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$), while sketch completion requires extrapolation (*e.g.*: $t \in [0, 1)$ given $t \in [0, \frac{1}{4})$), making it more analogous to visual outpainting.

4.4. Sketch Generation

We can perform conditional and unconditional sequential vector sketch generation after learning a generative model for sketch latents as described in Sec. 3.3.

Unconditional Generation: We unconditionally generate sketches by random sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ with the learnt VAE latent space, obtaining novel sketch descriptors $\nu_i \in \mathcal{V}$. We further sample $f_\theta(\nu_i, \dots)$ with varying timestamps $T \in [100, 300]$ and strokestamps $K \in [10, 30]$ obtaining sketches at varying levels of abstraction. Qualitative results in Fig. 10 show diverse and plausible samples from *Vector-MNIST* [17] and *Quick-Draw!* [28] datasets.

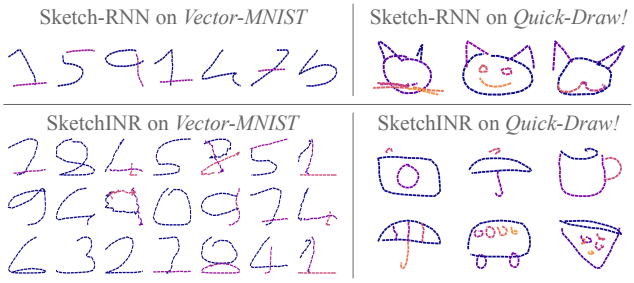

Figure 10. Unconditional sketch generation. Sampling sketch latents ν as well as stroke and point complexity (K, T).

Conditional Generation: To condition sketch generation, we use our pre-trained VAE encoder E to embed raster sketches, which can then be vectorised by the decoder f_θ. Fig. 11 shows qualitative results for sequential vector sketch generation from raster images, where we obtain comparable results with auto-regressive generative modelling [28]. We observe that treating vectorisation as a generative problem rather than a deterministic raster-to-vector mapping [7] leads us to have variations in generated vectors particularly in the form of stroke order resembling real copies of sketches made by humans.

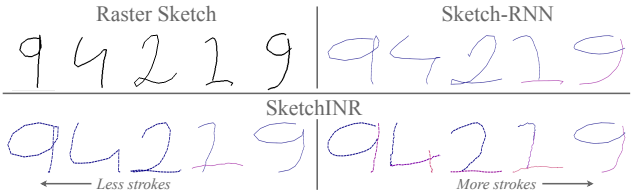

Figure 11. Conditional sketch generation (vectorisation) on *Vector-MNIST*. Raster sketches (top left) can have multiple vector forms, *e.g.*, sketched with different numbers of strokes (bottom).

4.5. Ablations and Discussion

Global vs Local Time Modelling: We model J timestamps across K strokes, by distributing timestamps uniformly over each stroke such that each stroke has $\frac{J}{K}$ timestamps. In other words, time $(t, \Delta t)$ is modelled *globally* for the entire sketch $\mathbf{p} = \{\mathrm{p}_1, \ldots, \mathrm{p}_N\}$. An alternative is to model $(t, \Delta t)$ as 0to1 for each stroke separately (local time-modelling) instead of the entire sketch. Particularly, each of the K strokes are modelled using N points, representing the entire sketch using $N \cdot K$ points as $\mathbf{p} = \{\mathrm{p}_1, \ldots, \mathrm{p}_{N \cdot K}\}$. Visually inspecting global vs local time embedding shows that local modelling leads to *jittery* sketches (see Fig. 12), whereas global modelling gives *visually smooth* sketches. We hypothesise that time is a global property of a sketch and not an independent property localised to each stroke. Hence, modelling time locally puts an additional optimisation burden on our implicit decoder to ensure smoothness and continuity, resulting in poorly reconstructed sketches.

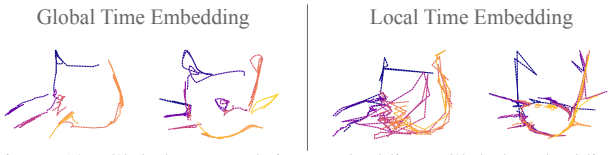

Figure 12. Global vs Local time embedding: Global embedding gives *smooth* sketches compared to local, which gives jittering.

Fixed vs Variable Smoothing Factor (γ): The intensity map I_k in Eq. (4) guides our raster-based visual loss $\mathcal{L}_V$. A key component in I_k is the smoothing parameter γ – low γ (thicker I_k) gives less accurate but smoother values on the entire visual region (easier training); high γ (thinner I_k) gives more accurate but sharper (steeper) intensity-gradients near ground-truth strokes (harder training). While we use a fixed γ with MSE Loss in Eq. (6) to balance out the optimisation complexity, we note that an alternative is to vary γ every fixed number of steps with a scheduler to allow faster optimisation. Specifically, we vary γ as 20to200 on a linear scale by incremental $\Delta\gamma$ every few iterations. Despite its theoretical advantage [25, 45], our initial experiments suggest only minor improvement that reduces training time by 6.1%. A detailed study of bundle-adjustment [45] for visual loss is an interesting future work.

Limitations and Future Work Despite it's efficiency as a representation for hand-drawn sketches, SketchINR suffers from a number of limitations. We naturally share limitations with other implicit representations including optimisation-based encoding and poor cross category generalisation. While unlike Sketch-RNN our generation network can be *trained on 2-3 categories at once*, it fails to generalise on more. In addition, a core limitation lies in slow convergence due to pixel space optimisation. While vector-space loss in the form of mean squared error mitigates this to some extent (Fig. 3), convergence is still slow and can possibly be improved by better engineering the implicit function. Strokes could also be smoothed by explicit inclusion of stroke gradient (slope) based regularisation in future work. Finally, vectorisation for conditional generation performs poorly on high complexity sketches.

5. Conclusion

We introduce visual implicits to represent vector sketches with compressed latent descriptors. This neural representation provides a high-fidelity and compact representation that raises the possibility of a sketch-specific codec for compactly representing large sketch datasets. Our SketchINR can decode sketches at varying levels of detail with controllable number of strokes, and provides superior cross-sketch interpolation between implicits, demonstrating a smoother latent space than auto-regressive models. Applications such as sketch-completion are also supported, including a-temporal completion not available with auto-regressive models. Decoding is inherently parallel and can be over $100\times$ faster than autoregressive models in practice.

References

[1] Emre Aksan, Fabrizio Pece, and Otmar Hilliges. Deepwriting: Making digital ink editable via deep generative modeling. In *CHI*, 2018. 1, 2

[2] Emre Aksan, Thomas Deselaers, Andrea Tagliasacchi, and Otmar Hilliges. Cose: Compositional stroke embeddings. In *NeurIPS*, 2020. 1, 2, 3, 6

[3] Stephan Alaniz, Massimiliano Mancini, Anjan Dutta, Diego Marcos, and Zeynep Akata. Abstracting sketches through simple primitives. In *ECCV*, 2022. 4

[4] Hmrishav Bandyopadhyay, Pinaki Nath Chowdhury, Ayan Kumar Bhunia, Aneeshan Sain, Tao Xiang, and Yi-Zhe Song. What sketch explainability really means for downstream tasks. In *CVPR*, 2024. 1

[5] Hmrishav Bandyopadhyay, Subhadeep Koley, Ayan Das, Ayan Kumar Bhunia, Aneeshan Sain, Pinaki Nath Chowdhury, Tao Xiang, and Yi-Zhe Song. Doodle your 3d: From abstract freehand sketches to precise 3d shapes. In *CVPR*, 2024. 1

[6] Ayan Kumar Bhunia, Ayan Das, Umar Riaz Muhammad, Yongxin Yang, Timothy M. Hospedales, Tao Xiang, Yulia Gryaditskaya, and Yi-Zhe Song. Pixelor: A competitive sketching ai agent. so you think you can beat me? In *SIGGRAPH Asia*, 2020. 1

[7] Ayan Kumar Bhunia, Pinaki Nath Chowdhury, Yongxin Yang, Timothy M Hospedales, Tao Xiang, and Yi-Zhe Song. Vectorization and rasterization: Self-supervised learning for sketch and handwriting. In *CVPR*, 2021. 2, 8

[8] Alexandre Binninger, Amir Hertz, Olga Sorkine-Hornung, Daniel Cohen-Or, and Raja Giryes. Sens: Part-aware sketch-based implicit neural shape modeling. *arXiv preprint arXiv:2306.06088*, 2024. 1, 2

[9] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 1965. 2

[10] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. In *NeurIPS*, 2020. 2

[11] Qualcomm Innovation Center. Ai model efficiency toolkit (aimet). https://github.com/quic/aimet, 2023. 6

[12] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018. 3

[13] Pinaki Nath Chowdhury, Aneeshan Sain, Ayan Kumar Bhunia, Tao Xiang, Yulia Gryaditskaya, and Yi-Zhe Song. Fs-coco: Towards understanding of freehand sketches of common objects in context. In *ECCV*, 2022. 2, 5, 6

[14] Pinaki Nath Chowdhury, Ayan Kumar Bhunia, Aneeshan Sain, Subhadeep Koley, Tao Xiang, and Yi-Zhe Song. What can human sketches do for object detection? In *CVPR*, 2023. 2

[15] John Collomosse, Tu Bui, and Hailin Jin. Livesketch: Query perturbations for guided sketch-based visual search. In *CVPR*, 2019. 1, 2

[16] Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. Béziersketch: A generative model for scalable vector sketches. In *ECCV*, 2020. 1, 2, 3, 5, 6

[17] Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. Sketchode: Learning neural sketch representation in continuous time. In *ICLR*, 2021. 1, 2, 3, 5, 6, 7

[18] Ayan Das, Yongxin Yang, Timothy M Hospedales, Tao Xiang, and Yi-Zhe Song. Cloud2curve: Generation and vectorization of parametric sketches. In *CVPR*, 2021. 2, 3

[19] Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. Chirodiff: Modelling chirographic data with diffusion models. In *ICLR*, 2023. 3, 6

[20] Carl De Boor and Carl De Boor. *A practical guide to splines*. Applied Mathematical Sciences, 1978. 3

[21] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *ICCV*, 2019. 6

[22] Asen L. Dontchev and R. Tyrrell Rockafellar. *Implicit Functions and Solution Mappings*. Springer-Verlag, 2014. 2

[23] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 1973. 6

[24] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM TOG*, 2012. 1, 2

[25] F. Dan Foresee and Martin T. Hagan. Gauss-newton approximation to bayesian learning. In *ICNN*, 1997. 8

[26] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *ICML*, 2018. 3

[27] Songwei Ge, Vedanuj Goswami, C. Lawrence Zitnick, and Devi Parikh. Creative sketch generation. In *ICLR*, 2021. 1

[28] David Ha and Douglas Eck. A neural representation of sketch drawings. In *ICLR*, 2018. 1, 2, 3, 5, 6, 7, 8

[29] Hai Victor Habi, Roy H. Jennings, and Arnon Netzer. Hmq: Hardware friendly mixed precision quantization block for cnns. In *ECCV*, 2020. 6

[30] Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. Spaghetti: Editing implicit shapes through part aware generation. *ACM TOG*, 2022. 2, 7

[31] Aaron Hertzmann. Why do line drawings work? a realism hypothesis. *Perception*, 2020. 1, 2

[32] Conghui Hu, Da Li, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-classifier: Sketch-based photo classifier generation. In *CVPR*, 2018. 3

[33] Conghui Hu, Da Li, Yongxin Yang, Timothy M Hospedales, and Yi-Zhe Song. Sketch-a-segmenter: Sketch-based photo segmenter generation. *IEEE TIP*, 2020. 2

[34] Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions. *arXiv preprint arXiv:2305.02463*, 2023. 2

[35] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 5

[36] Kurt Koffka. *Principles of Gestalt psychology*. Routledge, 2013. 3

[37] Subhadeep Koley, Ayan Kumar Bhunia, Aneeshan Sain, Pinaki Nath Chowdhury, Tao Xiang, and Yi-Zhe Song. Picture that sketch: Photorealistic image generation from abstract sketches. In *CVPR*, 2023. 1

[38] Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. Free2cad: Parsing freehand drawings into cad commands. *ACM TOG*, 2022. 1, 2

[39] Hangyu Lin, Yanwei Fu, Yu-Gang Jiang, and Xiangyang Xue. Sketch-bert: Learning sketch bidirectional encoder representation from transformers by self-supervised learning of sketch gestalt. In *CVPR*, 2020. 2

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 5

[41] Hongyu Liu, Ziyu Wan, Wei Huang, Yibing Song, Xintong Han, Jing Liao, Bin Jiang, and Wei Liu. Deflocnet: Deep image editing via flexible low-level controls. In *CVPR*, 2021. 1, 2

[42] Asif Masood and Sidra Ejaz. An efficient algorithm for robust curve fitting using cubic bezier curves. In *ICIC*, 2010. 3

[43] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019. 2, 3

[44] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 2021. 3

[45] Jorge J. Moŕe. The levenberg-marquardt algorithm. In *Numerical Analysis*, 1976. 8

[46] Umar Riaz Muhammad, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Learning deep sketch abstraction. In *CVPR*, 2018. 3

[47] Quan H Nguyen and William J Beksi. Single image super-resolution via a dual interactive implicit neural network. In *WACV*, 2023. 7

[48] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 2, 3, 4, 7

[49] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *ICML*, 2019. 3

[50] Michael Revow, Christopher KI Williams, and Geoffrey E Hinton. Using generative models for handwritten digit recognition. *IEEE TPAMI*, 1996. 2

[51] Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *CVPR*, 2020. 1, 2

[52] Aneeshan Sain, Ayan Kumar Bhunia, Pinaki Nath Chowdhury, Subhadeep Koley, Tao Xiang, and Yi-Zhe Song. Clip for all things zero-shot sketch-based image retrieval, fine-grained or not. In *CVPR*, 2023. 2

[53] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: learning to retrieve badly drawn bunnies. *ACM TOG*, 2016. 1, 5, 6

[54] Heping Sheng, John Wilder, and Dirk B. Walther. Where to draw the line? *PLOS One*, 2021. 2

[55] Yannick Str umpler, Janis Postels, Ren Yang, Luc van Gool, and Federico Tombari. Implicit neural representations for image compression. In *ECCV*, 2022. 6

[56] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. In *ICLR*, 2020. 6

[57] Yael Vinker, Ehsan Pajouheshgar, Jessica Y Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. Clipasso: Semantically-aware object sketching. *ACM TOG*, 2022. 1, 2, 3

[58] Yael Vinker, Yuval Alaluf, Daniel Cohen-Or, and Ariel Shamir. Clipascene: Scene sketching with different types and levels of abstraction. In *ICCV*, 2023. 1, 2, 3

[59] Alexander Wang, Mengye Ren, and Richard Zemel. Sketchembednet: Learning novel concepts by imitating drawings. In *ICML*, 2021. 2, 4

[60] Ximing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. In *NeurIPS*, 2023. 1

[61] Peng Xu, Yongye Huang, Tongtong Yuan, Kaiyue Pang, Yi-Zhe Song, Tao Xiang, Timothy M Hospedales, Zhanyu Ma, and Jun Guo. Sketchmate: Deep hashing for million-scale human sketch retrieval. In *CVPR*, 2018. 1, 2

[62] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. 2

[63] Qian Yu, Yongxin Yang, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-net that beats humans. In *BMVC*, 2015. 2

[64] Qian Yu, Feng Liu, Yi-Zhe Song, Tao Xiang, Timothy M Hospedales, and Chen-Change Loy. Sketch me that shoe. In *CVPR*, 2016. 1, 3

[65] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *ICCV*, 2021. 1

[66] Wenni Zheng, Pengbo Bo, Yang Liu, and Wenping Wang. Fast b-spline curve fitting by l-bfgs. *Computer Aided Geometric Design*, 2012. 2