SketchEmbedNet: Learning Novel Concepts by Imitating Drawings

Alexander Wang* ALEXW@CS.TORONTO.EDU
University of Toronto, Vector Institute
Mengye Ren* MREN@CS.TORONTO.EDU
University of Toronto, Vector Institute, Uber ATG
Richard Zemel ZEMEL@CS.TORONTO.EDU
University of Toronto, Vector Institute, CIFAR

Abstract

Sketch drawings are an intuitive visual domain that appeals to human instinct. Previous work has shown that recurrent neural networks are capable of producing sketch drawings of a single or few classes at a time. In this work we investigate representations developed by training a generative model to produce sketches from pixel images across many classes in a sketch domain. We find that the embeddings learned by this sketching model are informative for visual tasks and capture some forms of visual understanding. We then use them to exceed state-of-the-art performance in unsupervised few-shot classification on the Omniglot and mini-ImageNet benchmarks. We also leverage the generative capacity of our model to produce high quality sketches of novel classes based on just a single example.

1. Introduction

Upon encountering a novel concept, such as a six-legged turtle, humans can quickly generalize this concept by composing a mental picture. The ability to generate drawings greatly facilitates communicating new ideas. This dates back to the advent of writing, as many ancient written languages are based on logograms, such as Chinese hanzi and Egyptian hieroglyphs, where each character is essentially a sketch of the object it represents. We often see complex visual concepts summarized by a few simple strokes.

Inspired by the human ability to draw, recent research has explored the potential to generate sketches using a wide variety of machine learning models, ranging from hierarchical Bayesian models (Lake et al., 2015), to more recent deep autoregressive models (Chen et al., 2017; Gregor et al., 2015; Ha and Eck, 2018) and generative adversarial nets (GANs) (Li et al., 2019). It is a natural question to ask whether we can obtain useful intermediate representations from models that produce sketches. We believe that this output format is more natural and may better encode visual information compared to existing generative models (Doersch et al., 2015; Donahue et al., 2017; Goodfellow et al., 2014; Kingma and Welling, 2014; Ranzato et al., 2006). Of them, hierarchical

* Equal contribution

Bayesian models suffers from prolonged inference time, while other current sketch models mostly focus on producing drawings in a closed set setting with a few classes (Chen et al., 2017; Ha and Eck, 2018), or on improving log likelihood at the pixel level (Rezende et al., 2016). Leveraging the learned representation from these drawing models remains a rather unexplored topic.

In this paper, we pose the following question: *Can we learn a generalized embedding function that captures salient and compositional features by directly imitating human sketches?* The answer is affirmative. In our experiments we develop SketchEmbedNet, an RNN-based sketch model trained to map grayscale and natural image pixels to the sketch domain. It is trained on hundreds of classes without the use of class labels to learn a robust drawing model that can sketch diverse and unseen inputs. We evaluate the latent "SketchEmbedding" on unsupervised few-shot classification, and achieve state-of-the-art performance on the Omniglot (Lake et al., 2015) dataset. We then explore how the embeddings depend on image components and their spatial relations. As well, we investigate the generative variability and recognizability of samples in one-shot generation.

We also push the boundary further by applying our sketch model to natural images—to our knowledge, we are the first to extend stroke-based autoregressive models to produce drawings of open domain natural images. We train our model with adapted SVG images from the Sketchy dataset (Sangkloy et al., 2016) and then evaluate the embedding quality directly on unseen classes in the mini-ImageNet task for few-shot classification (Vinyals et al., 2016). Our approach out-performs existing unsupervised few-shot learning methods (Antoniou and Storkey, 2019; Hsu et al., 2019; Khodadadeh et al., 2019) on this natural image benchmark. In both the sketch and natural image domain, we show that by learning to draw, our methods generalize well even across different datasets and classes.

2. Related Work

In this section we review relevant literature including generating sketch-like images, unsupervised representation learning, unsupervised few-shot classification and sketch-based image retrieval (SBIR).

Autoregressive drawing models Graves (2013) used an LSTM to directly output the pen coordinates to imitate handwriting sequences. SketchRNN (Ha and Eck, 2018) built on this by applying it to general sketches beyond characters. Cao et al. (2019); Ribeiro et al. (2020); Song et al. (2018) all extended SketchRNN through architectural improvements. Chen et al. (2017) changed inputs to be pixel images. This and the previous three approaches considered multi-class sketching, but none handle more than 20 classes. Autoregressive models also generated images directly in the pixel domain. DRAW (Gregor et al., 2015) used recurrent attention to plot pixels; Rezende et al. (2016) extended this to one-shot generation and PixelCNN (van den Oord et al., 2016) generated image pixels sequentially.

Image processing methods & GANs Other works produced sketch-like images based on style transfer or low-level image processing techniques. Classic methods were based on edge detection and image segmentation (Arbelaez et al., 2011; Xie and Tu, 2017). Zhang et al. (2015) used a CNN to directly produce sketch-like pixels for face images. Photo-sketch and pix2pix (Isola et al., 2017; Li et al., 2019) proposed a conditional GAN to generated images across different style domains. Image

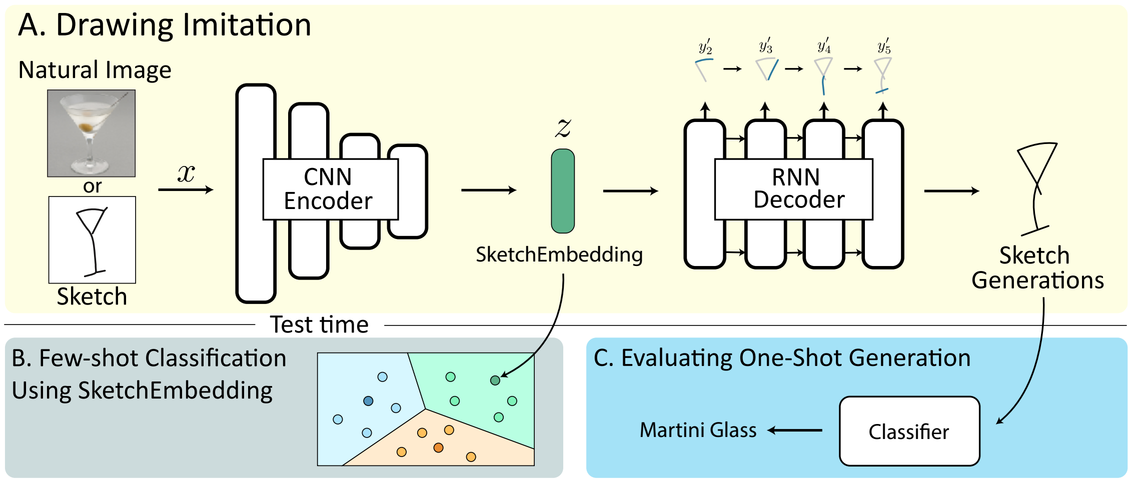


Figure 1: **A:** A natural or sketch pixel image is passed into the CNN encoder to obtain Gaussian SketchEmbedding z, which is concatenated with the previous stroke y_{t-1} as the decoder input at each timestep to generate y_t. **B+C:** Downstream tasks performed after the training is complete.

processing based methods do not acquire high-level image understanding, as all the operations are in terms of low-level filtering.

Unsupervised representation learning In the sketch image domain, our method is similar to the large category of generative models which learn unsupervised representations by the principle of analysis-by-synthesis. Hinton and Nair (2005) proposed to operate in a sketch domain and learn to draw by synthesizing an interpretable motor program. Bayesian Program Learning (Lake et al., 2015) draws through exact inference of common strokes but learning and inference are computationally challenging. As such, a variety of deep generative models aim to perform approximate Bayesian inference by using an encoder structure that directly predicts the embedding, e.g., deep autoencoders (Vincent et al., 2010), Helmholtz Machine (Dayan et al., 1995), variational autoencoder (VAE) (Kingma and Welling, 2014), BiGAN (Donahue et al., 2017), etc. Our method is also related to the literature of self-supervised representation learning (Doersch et al., 2015; Gidaris et al., 2018; Noroozi and Favaro, 2016; Zhang et al., 2016), as sketch strokes are part of the input data itself. In few-shot learning (Finn et al., 2017; Snell et al., 2017; Vinyals et al., 2016), recent work has explored unsupervised meta-training. CACTUs, AAL and UMTRA (Antoniou and Storkey, 2019; Hsu et al., 2019; Khodadadeh et al., 2019) all operate by generating pseudo-labels for training.

Sketch-based image retrieval (SBIR) In SBIR, a model is provided a sketch-drawing and retrieves a photo of the same class. The area is split into fine-grained (FG-SBIR) (Bhunia et al., 2020; Sangkloy et al., 2016; Yu et al., 2016) and a zero-shot setting (ZS-SBIR) (Dey et al., 2019; Dutta and Akata, 2019; Pandey et al., 2020). FG-SBIR considers minute details while ZS-SBIR learns high-level cross-domain semantics and a joint latent space to perform retrieval.

3. Learning to Imitate Drawings

Here we describe learning to draw through sketch imitation. Our architecture is a generative encoder-decoder model with a CNN encoder for pixel images and an RNN decoder to output vector sketches as shown in Figure 1. Unlike other drawing models that only train on a single or few classes (Chen et al., 2017; Ha and Eck, 2018), SketchEmbedNet can sketch images from a wide variety of classes. We also introduce a differentiable rasterization function for computing an additional pixel-based training loss.

Input & output representation Unlike SketchRNN which encodes drawing sequences, we learn an image embedding by mapping pixels to sketches, similar to Chen et al. (2017). Training data for this task (adopted from Ha and Eck (2018)) consists of a tuple $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$ is the input image and $\boldsymbol{y} \in \mathbb{R}^{T \times 5}$ is the stroke target. T is the maximum sequence length of the stroke data $\boldsymbol{y}$, and each stroke $\boldsymbol{y}_t$ consists of 5 elements, $(\Delta_x, \Delta_y, s_1, s_2, s_3)$. The first 2 elements are horizontal and vertical displacements on the drawing canvas from the endpoint of the previous stroke. The latter 3 elements are mutually exclusive pen states: s_1 indicates the pen is on paper for the next stroke, s_2 indicates the pen is lifted, and s_3 indicates the sketch sequence has ended. $\boldsymbol{y}_0$ is initialized with (0, 0, 1, 0, 0) to start the generative process. Note that no class information is available while training.

SketchEmbedding as a compositional encoding of images A CNN is used to encode the input image $\boldsymbol{x}$ and obtain the latent space representation $\boldsymbol{z}$, as shown in Figure 1. To model intra-class variance, $\boldsymbol{z}$ is a Gaussian random variable parameterized by CNN outputs, similar to a VAE (Kingma and Welling, 2014). Throughout this paper, we refer to $\boldsymbol{z}$ as the *SketchEmbedding*. In typical image representations the embedding is trained to classify object classes, or to reconstruct the input pixels. Here, since the SketchEmbedding is fed into an RNN decoder to produce a sequence of drawing actions, $\boldsymbol{z}$ is additionally encouraged to have a compositional understanding of the object structure, instead of just an unstructured set of pixel features. For instance, if the input image is a turtle with six legs, a regular CNN embedding will have salient features of legs in multiple places of the feature map. After an average pooling layer, it will only retain information about a leg-like feature but not how many legs are present and where they are positioned with respect to the canvas. The loss of compositional information of regular CNN embeddings is also observed in the GAN literature (Goodfellow, 2017). By contrast, the SketchEmbedNet must preserve the information for drawing six distinct legs in order to succeed in the imitation task.

To accommodate the increased training data complexity by including hundreds of classes, we also upscale the size of our model in comparison to work by Chen et al. (2017); Ha and Eck (2018); Song et al. (2018). The backbone is either a 4-layer CNN (Conv4) (Vinyals et al., 2016) for consistent comparisons in the few-shot setting or a ResNet12 (Oreshkin et al., 2018) which produces better drawing results. In comparison, Chen et al. (2017) only use 2D convolution with a maximum of 8 filters.

RNN decoder The RNN decoder used in SketchEmbedNet is the same as in SketchRNN (Ha and Eck, 2018). The decoder outputs a mixture density which represents the stroke distribution at each timestep. Specifically, the stroke distribution is a mixture of some hyperparameter M bivariate Gaussians denoting spatial offsets as well as the probability of the three pen states s_{1-3}. The spatial offsets $\boldsymbol{\Delta} = (\Delta x, \Delta y)$ are sampled from the mixture of Gaussians, described by: (1) the normalized mixture weight π_j; (2) mixture means $\boldsymbol{\mu}_j = (\mu_x, \mu_y)_j$; and (3) covariance matrices Σ_j. We further reparameterize each Σ_j with its standard deviation $\boldsymbol{\sigma}_j = (\sigma_x, \sigma_y)_j$ and correlation coefficient $\rho_{xy,j}$. Thus, the stroke offset distribution is

$$p(\boldsymbol{\Delta}) = \sum_{j=1}^{M} \pi_j \mathcal{N}(\boldsymbol{\Delta} | \boldsymbol{\mu}_j, \Sigma_j). \tag{1}$$

The RNN is implemented using a HyperLSTM (Ha et al., 2017); LSTM weights are generated at each timestep by a smaller recurrent "hypernetwork" to improve training stability. Generation is autoregressive, using $\boldsymbol{z} \in \mathbb{R}^D$, concatenated with the stroke from the previous timestep $\boldsymbol{y}_{t-1}$, to form the input to the LSTM. Stroke $\boldsymbol{y}_{t-1}$ is the ground truth supervision at train time (teacher forcing), or a sample $\boldsymbol{y}'_{t-1}$, from the mixture distribution output by the model during from timestep $t-1$.

3.1. Learning

We train the drawing model in an end-to-end fashion by jointly optimizing three losses: a pen loss $\mathcal{L}_{\text{pen}}$ for learning pen states, a stroke loss $\mathcal{L}_{\text{stroke}}$ for learning pen offsets, and our proposed pixel loss $\mathcal{L}_{\text{pixel}}$ for matching the visual similarity of the predicted and the target sketch:

$$\mathcal{L} = \mathcal{L}_{\text{pen}} + (1-\alpha)\mathcal{L}_{\text{stroke}} + \alpha\mathcal{L}_{\text{pixel}}, \tag{2}$$

where α is a loss weighting hyperparameter. Both $\mathcal{L}_{\text{pen}}$ and $\mathcal{L}_{\text{stroke}}$ were in SketchRNN, while the $\mathcal{L}_{\text{pixel}}$ is our novel contribution to stroke-based generative models. Unlike SketchRNN, we do not impose a prior using a KL term as it negatively impacted later experiments.

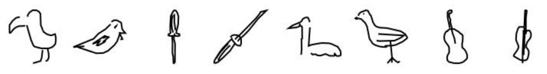
Pen loss The pen-states predictions $\{s'_1, s'_2, s'_3\}$ are optimized as a simple 3-way classification with the softmax cross-entropy loss, $\mathcal{L}_{\text{pen}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{m=1}^{3} s_{m,t}\log(s'_{m,t})$.

Stroke loss The stroke loss maximizes the log-likelihood of the spatial offsets of each ground truth stroke $\boldsymbol{\Delta}_t$ given the mixture density distribution p_t at each timestep:

$$\mathcal{L}_{\text{stroke}} = -\frac{1}{T}\sum_{t=1}^{T} \log p_t(\boldsymbol{\Delta}_t). \tag{3}$$

Pixel loss While pixel-level reconstruction objectives are common in generative models (Gregor et al., 2015; Kingma and Welling, 2014; Vincent et al., 2010), we introduce a pixel-based objective for vector sketch generation. After decoding, a *differentiable rasterization* function f_{raster} is used to map the sketch into a pixel image. f_{raster} transforms a stroke sequence $\boldsymbol{y}$ into a set of 2D line segments $(l_0, l_1), (l_1, l_2) \dots (l_{T-1}, l_T)$ where $l_t = \sum_{\tau=0}^{t} \boldsymbol{\Delta}_\tau$. It renders by fixing canvas dimensions, and scaling and centering strokes before determining pixel intensity based on the L_2 distance between each pixel to lines in the drawing. Further details on f_{raster} can be found in Appendix A. f_{raster} is applied to both the prediction $\boldsymbol{y}'$ and ground truth $\boldsymbol{y}$, to produce two pixel images. Gaussian blur $g_{\text{blur}}(\cdot)$ is used to reduce strictness before computing the binary cross-entropy loss, $\mathcal{L}_{\text{pixel}}$:

$$I = g_{\text{blur}}(f_{\text{raster}}(\boldsymbol{y})), \quad I' = g_{\text{blur}}(f_{\text{raster}}(\boldsymbol{y}')), \quad \mathcal{L}_{\text{pixel}} = -\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W} I_{ij}\log(I'_{ij}). \tag{4}$$


(*a*) Quickdraw samples

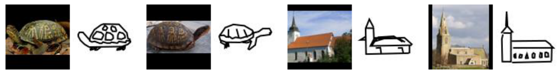

(*b*) Sketchy samples

Figure 2: Examples from Sketchy (Sangkloy et al., 2016) and Quickdraw (Jongejan et al., 2016). Sketchy examples are reshaped and padded to increase image–sketch spatial agreement.

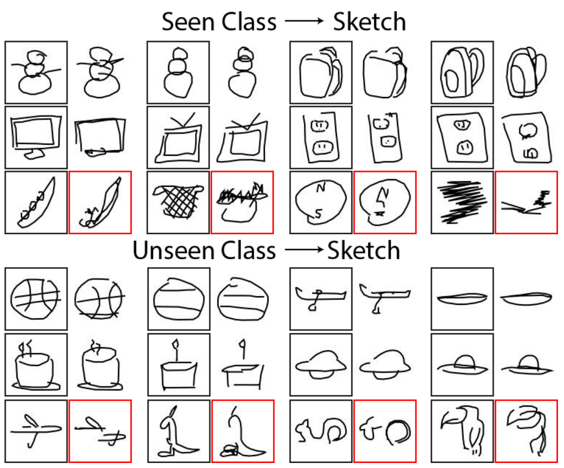


Figure 3: Sampled generated drawings of Quickdraw examples. Weaker drawings boxed in red.

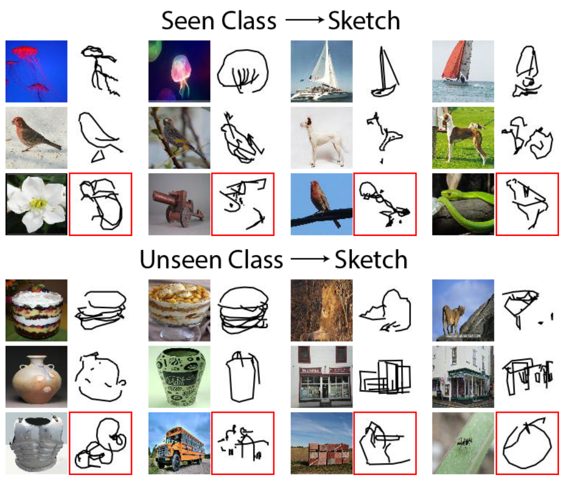


Figure 4: Sampled drawings of mini-ImageNet examples. Weaker drawings boxed in red.

Curriculum training schedule We find that α (in Equation 2) is an important hyperparameter that impacts both the learned embedding space and the generation quality of SketchEmbedNet. A curriculum training schedule is used, increasing α to prioritize $\mathcal{L}_{\text{pixel}}$ relative to $\mathcal{L}_{\text{stroke}}$ as training progresses; this makes intuitive sense as a single drawing can be produced by many different stroke sequences but learning to draw in a fixed manner is easier. While $\mathcal{L}_{\text{pen}}$ promotes reproducing a specific drawing sequence, $\mathcal{L}_{\text{pixel}}$ only requires that the generated drawing visually matches the image. Like a human, the model should learn to follow one drawing style (a la paint-by-numbers) before learning to draw freely.

4. Drawing Imitation Experiments

In this section, we introduce our experiments on training SketchEmbedNet using two sketching datasets. The first is based on pure stroke-based drawings, and the second consists of natural image and drawing pairs.

Quickdraw: Stroke-based image sketching The Quickdraw dataset (Jongejan et al., 2016) consists of 345 classes of each with 70,000 examples, produced by human players participating in the game "Quick, Draw!". Examples from the Quickdraw dataset are shown in Figure 2(*a*). The input image $\boldsymbol{x}$ is a direct rasterization of the drawing data $\boldsymbol{y}$. Images from 300 of the 345 classes are randomly selected for training; $\boldsymbol{x}$ is rasterized to a resolution of 28×28 and stroke labels $\boldsymbol{y}$ padded up to length $T = 64$. Any drawing samples exceeding this length were discarded. Note that this an unsupervised representation learning approach, as no class information is used by the system.

Sketchy: Open domain natural image sketching We further extend our stroke-based generation model on open domain natural images. Here, the input is an RGB photo, and the output is a human drawing which does not align with the photo precisely and also does not match with the low-level image details. This is a novel setting, as prior efforts have only applied their sketch RNN models on the Quickdraw dataset or natural images with only two object classes (shoe/chair) and scrubbed backgrounds (Chen et al., 2017; Ha and Eck, 2018; Song et al., 2018; Yu et al., 2016). Learning to sketch open domain natural images is very challenging as it requires the model to identify the subject and filter unnecessary details not present in the sketch. At test time, we further challenge our method by evaluating on unseen data distributions necessitating generalization over natural images.

For this task we use the Sketchy dataset (Sangkloy et al., 2016) which consists of ImageNet images paired with vector sketches for a total of 56k examples after processing. Sketches are stored as SVGs that we adapt to our format by sampling path elements. Images are also centered, padded and resized to resolution 84×84 (see Figure 2(b)). We fix the maximum sequence length to $T = 100$, and use all 125 categories but remove classes that have overlapping child synsets with the test classes of mini-ImageNet (Vinyals et al., 2016). This enables testing on mini-ImageNet without any alterations to the benchmark. Once again this is an unsupervised learning formulation.

Results and visualizations Figure 3 shows drawings conditioned on sketch image inputs. There is little noticeable drop in quality when we sample sketches from unseen classes compared to those it has seen before. Figure 4 shows examples of sketches generated from natural images. While they are not fine-grained copies, these sketches clearly demonstrate SketchEmbedNet's ability to capture key components of seen and unseen classes. Unlike pixel-based auto-encoder models, our sketches do not follow the exact pose of the original strokes, but rather capture a general notion of component groupings. As shown in the basketball example of Figure 3, the lines are not a good pixel-wise match to those in the original image yet they are placed in sensible relative positions. Weaker examples are presented in the last row of Figure 3 and 4; regardless, even poorer examples still capture some structural aspects of the original image. We use the models trained in this section for later downstream tasks. Implementation details can be found in Appendix B.

5. Few-shot Classification using SketchEmbedding

We would like to assess the benefits of learning to draw by performing few-shot classification with our learned embedding space. Examining performance on discriminative tasks reveals that learning to imitate sketches allows the embeddings to capture salient and compositional information of a novel object class. Below we describe our few-shot classification procedure and summarize results on the Omniglot (Lake et al., 2015) and mini-ImageNet benchmarks (Vinyals et al., 2016).

Comparison to unsupervised few-shot classification In unsupervised few-shot classification, a model is not provided with any class labels during meta-training, until it is given a few labeled examples (”shots”) of the novel classes at meta-test time. While our model is provided a ”target”—a

Table 1: Few-shot classification results on Omniglot

Omniglot			**(way, shot)**			
Algorithm	**Backbone**	**Train Data**	**(5,1)**	**(5,5)**	**(20,1)**	**(20,5)**
Training from Scratch (Hsu et al., 2019)	N/A	Omniglot	52.50 ± 0.84	74.78 ± 0.69	24.91 ± 0.33	47.62 ± 0.44
CACTUs-MAML (Hsu et al., 2019)	Conv4	Omniglot	68.84 ± 0.80	87.78 ± 0.50	48.09 ± 0.41	73.36 ± 0.34
CACTUs-ProtoNet (Hsu et al., 2019)	Conv4	Omniglot	68.12 ± 0.84	83.58 ± 0.61	47.75 ± 0.43	66.27 ± 0.37
AAL-ProtoNet (Antoniou and Storkey, 2019)	Conv4	Omniglot	84.66 ± 0.70	88.41 ± 0.27	68.79 ± 1.03	74.05 ± 0.46
AAL-MAML (Antoniou and Storkey, 2019)	Conv4	Omniglot	88.40 ± 0.75	98.00 ± 0.32	70.20 ± 0.86	88.30 ± 1.22
UMTRA (Khodadadeh et al., 2019)	Conv4	Omniglot	83.80	95.43	74.25	92.12
Random CNN	Conv4	N/A	67.96 ± 0.44	83.85 ± 0.31	44.39 ± 0.23	60.87 ± 0.22
Conv-VAE	Conv4	Omniglot	77.83 ± 0.41	92.91 ± 0.19	62.59 ± 0.24	84.01 ± 0.15
Conv-VAE	Conv4	Quickdraw	81.49 ± 0.39	94.09 ± 0.17	66.24 ± 0.23	86.02 ± 0.14
SketchEmbedding *(Ours)*	Conv4	Omniglot	94.88 ± 0.22	99.01 ± 0.08	86.18 ± 0.18	96.69 ± 0.07
SketchEmbedding-avg *(Ours)*	Conv4	Quickdraw	96.37	99.43	90.69	98.07
SketchEmbedding-best *(Ours)*	Conv4	Quickdraw	**96.96** ± 0.17	**99.50** ± 0.06	**91.67** ± 0.14	**98.30** ± 0.05
MAML *(Supervised)* (Finn et al., 2017)	Conv4	Omniglot	94.46 ± 0.35	98.83 ± 0.12	84.60 ± 0.32	96.29 ± 0.13
ProtoNet *(Supervised)* (Snell et al., 2017)	Conv4	Omniglot	98.35 ± 0.22	99.58 ± 0.09	95.31 ± 0.18	98.81 ± 0.07

sequence of strokes—during training, it is not given any class information. Therefore, we propose that the presented sketch imitation training is comparable to other class-label-free representation learning approaches (Berthelot et al., 2019; Caron et al., 2018; Donahue et al., 2017) and the learned SketchEmbeddings can be applied to unsupervised few-shot classification methods.

In our experiments, we compare to previous unsupervised few-shot learning approaches: CACTUs (Hsu et al., 2019), AAL (Antoniou and Storkey, 2019), and UMTRA (Khodadadeh et al., 2019). These methods create pseudo-labels during meta-training using either clustering or data augmentation. As additional baselines, a Conv-VAE (Kingma and Welling, 2014) and a random CNN are also included, both using the same Conv4 backbone.

Few-shot experimental setup The CNN encoder of SketchEmbedNet is used as an embedding function combined with a linear classification head to perform few-shot classification. The embedding is made deterministic by taking the mean of the random normal latent space z and discarding the variance parameter from the encoder. Otherwise, the conventional episodic setup for few-shot classification is used; each episode consists of a labeled "support" set of $N \times K$ (N-way K-shot) embeddings and an unlabeled "query" set. The linear classification head is trained on the labeled support set and evaluated on the query set.

Few-shot classification on Omniglot The Omniglot dataset (Lake et al., 2015) contains 50 alphabets, 1623 unique character types, each with 20 examples and is presented as both a greyscale image and a stroke drawing. We use the same train-test split as Vinyals et al. (2016) along with randomly sampled episodes. Experiments using the more challenging Lake split (Lake et al., 2015) are in Appendix E.

To ensure a fair comparison with other few-shot classification models, we use the same convolutional encoder (Conv4) as Vinyals et al. (2016). Results from training only on Omniglot are also presented to demonstrate effectiveness without the larger Quickdraw dataset. No significant improvements were observed using the deeper ResNet12 (Oreshkin et al., 2018) architecture; additional results are in Appendix J.

Table 2: Few-shot classification results on mini-ImageNet

mini-ImageNet			**(way, shot)**			
Algorithm	**Backbone**	**Train Data**	**(5,1)**	**(5,5)**	**(5,20)**	**(5,50)**
Training from Scratch (Hsu et al., 2019)	N/A	mini-ImageNet	27.59 ± 0.59	38.48 ± 0.66	51.53 ± 0.72	59.63 ± 0.74
CACTUs-MAML (Hsu et al., 2019)	Conv4	mini-ImageNet	39.90 ± 0.74	53.97 ± 0.70	63.84 ± 0.70	69.64 ± 0.63
CACTUs-ProtoNet (Hsu et al., 2019)	Conv4	mini-ImageNet	39.18 ± 0.71	53.36 ± 0.70	61.54 ± 0.68	63.55 ± 0.64
AAL-ProtoNet (Antoniou and Storkey, 2019)	Conv4	mini-ImageNet	37.67 ± 0.39	40.29 ± 0.68	-	-
AAL-MAML (Antoniou and Storkey, 2019)	Conv4	mini-ImageNet	34.57 ± 0.74	49.18 ± 0.47	-	-
UMTRA (Khodadadeh et al., 2019)	Conv4	mini-ImageNet	39.93	50.73	61.11	67.15
Random CNN	Conv4	N/A	26.85 ± 0.31	33.37 ± 0.32	38.51 ± 0.28	41.41 ± 0.28
Conv-VAE	Conv4	mini-ImageNet	28.68 ± 0.36	35.30 ± 0.35	39.45 ± 0.30	41.46 ± 0.29
Conv-VAE	Conv4	Sketchy	28.87 ± 0.37	34.91 ± 0.36	39.09 ± 0.31	41.28 ± 0.31
SketchEmbedding-avg *(ours)*	ResNet12	Sketchy	38.55	54.39	65.14	69.70
SketchEmbedding-best *(ours)*	ResNet12	Sketchy	**40.39** ± 0.44	**57.15** ± 0.38	**67.60** ± 0.33	**71.99** ± 0.3
MAML *(supervised)* (Finn et al., 2017)	Conv4	mini-ImageNet	46.81 ± 0.77	62.13 ± 0.72	71.03 ± 0.69	75.54 ± 0.62
ProtoNet *(supervised)* (Snell et al., 2017)	Conv4	mini-ImageNet	46.56 ± 0.76	62.29 ± 0.71	70.05 ± 0.65	72.04 ± 0.60

All of our methods out-perform the previous state-of-the-art on the unsupervised Omniglot benchmark (Table 1). The Quickdraw trained model surpasses supervised MAML (Finn et al., 2017), and is on par with a supervised Prototypical Network (ProtoNet) (Snell et al., 2017) model despite training on an entirely different distribution of sketches rather than handwritten characters. We report both the best seed and an average of 15 random seeds. The demonstrated approach excels in higher-shot settings and also manages to match the supervised ProtoNet model in the 5-shot regime. Both baselines, a Conv-VAE and a random CNN, perform well compared to other unsupervised methods.

Few-shot classification on mini-ImageNet We extend our investigation and assess SketchEmbeddings for the classification of natural images in the mini-ImageNet benchmark (Vinyals et al., 2016). The same CNN encoder model from the natural image sketching task is used to match the visual domain of the examples we hope to classify.

The mini-ImageNet dataset consists of 100 classes each with 600 examples. Classes are split 64-16-20 for training, validation and test (Ravi and Larochelle, 2017). As noted earlier, any examples in the Sketchy dataset that are also present in the mini-ImageNet test were removed by filtering the synset (and children synset) IDs ensuring train and test classes are disjoint.

Classification results on the mini-ImageNet task are shown in Table 2. Our best model outperforms previous state-of-the-art unsupervised methods on few-shot settings and stays competitive on average. Additional results investigating the impact of random seeding are in Appendix G.

Varying weighting of pixel-loss We sweep the pixel loss coefficient α_{max} to quantify its impact on model performance on the Omniglot task (Table 7). There is a substantial improvement in few-shot classification when α_{max} is non-zero. α_{max}= 0.50 achieves the best results, and the trend goes downward when α_{max} approaches 1.0, i.e. the weighting for $\mathcal{L}_{\text{stroke}}$ goes to 0.0. This is reasonable as the training of SketchEmbedNet is more stable under the guidance of ground truth strokes.

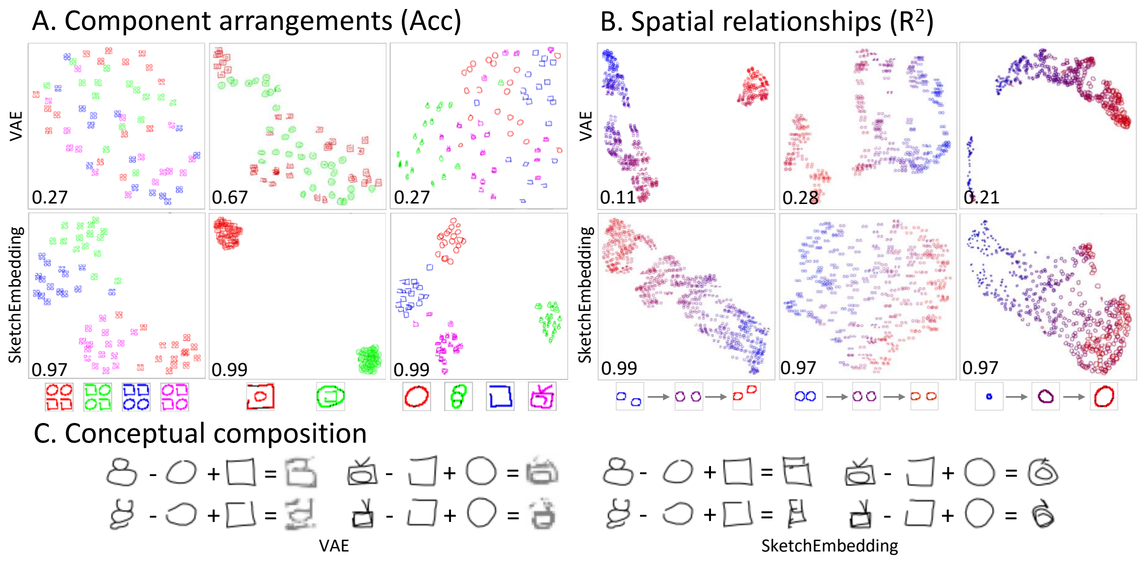


Figure 5: Experiments exploring properties of SketchEmbeddings. Numbers in **A, B** are linear model classification accuracy and regression R^2 respectively. Images colored for understandability.

6. Properties of SketchEmbeddings

We hypothesize that reproducing a sketch drawing rather than a pixel-based approach requires the preservation of more structural information in the image. We examine this through several comparisons of embeddings generated by a Conv-VAE to those of SketchEmbedNet.

Component arrangements We construct images that contain the same set of objects but in different arrangements to test sensitivity to this property in image space. We embed these examples with both generative models then project into 2D space using UMAP (McInnes et al., 2018) to visualize how the representations vary with arrangement. We also apply a simple linear classifier trained on 100 examples and tested on 200 to predict the class of each embedding. In the first 2 panels of Figure 5-A, we see that the SketchEmbeddings are easily separated in unsupervised clustering and are classified with higher accuracy shown in the bottom left in a supervised task. The rightmost panel of Figure 5-A exhibits non-synthetic classes with duplicated shapes; snowmen with circles and televisions with squares. This shows that learning to draw facilitates an understanding of pen actions that vary for different, but visually similar shapes.

Spatial relationships Drawing also builds awareness of relevant underlying variables, such as the spatial relationships between components of the image. We examine the degree to which the underlying variables of angle, distance or size are captured by the embedding, by constructing images that vary along each dimension respectively. The embeddings are again grouped by a 2D projection in Figure 5-B and used for a supervised linear regression task with R^2 values in the bottom left. When given supervision, SketchEmbeddings are very identifiable and a simple linear model learns the relationship effectively while it struggles with Conv-VAE embeddings. When unsupervised, the 2D projection of SketchEmbeddings arranges the examples in each task along an axis corresponding to the latent variable compared to the Conv-VAE embeddings which is visibly non-linear and arranges

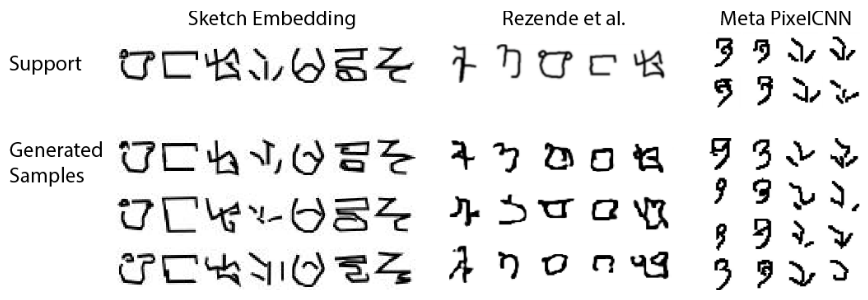


Figure 6: One-shot Omniglot generation compared to Rezende et al. (2016) and Reed et al. (2017).

in clusters. We used 1000 training examples for the regression tasks as the Conv-VAE embeddings fail completely when using 100 examples; additional results are in Appendix F.

Conceptual composition Finally, we explore conceptual composition using SketchEmbeddings (Figure 5-C) by embedding different Quickdraw examples then performing arithmetic with the latent vectors. By subtracting a circle embedding and adding a square embedding from a snowman composed of stacked circles, we produce stacked boxes. This property of vector arithmetic is reminiscent of language representations, as evidenced in analogies like King - Man + Woman = Queen (Ethayarajh et al., 2019). Our results indicate that this property is captured to a greater degree in SketchEmbedding than in the pixel-based VAE embeddings. Composing SketchEmbeddings produces decoded examples that appeal to our intuitive conceptual understanding while the VAE degrades to blurry, fragmented images.

7. One-Shot Generation

To evaluate the sketches generated by our model, we make qualitative comparisons to other one-shot generative models and quantitatively assess our model through visual classification via a ResNet-101 (He et al., 2016).

Qualitative comparisons We compare SketchEmbedNet one-shot generations of Omniglot characters with examples from other few-shot (Reed et al., 2017) and one-shot (Rezende et al., 2016) approaches (Figure 6). In the settings shown, none of the models have seen any examples from the character class, or the parent alphabet. Furthermore, the drawer has seen no written characters during training and is trained only on the Quickdraw dataset. Visually, our generated images more closely resemble the support examples and the variations by stretching and shifting strokes better preserves the semantics of each character. Generations in pixel space may disrupt strokes and alter the character to human perception. This is especially true for written characters as they are frequently defined by a specific set of strokes instead of blurry clusters of pixels.

Quantitative evaluation of generation quality Evaluating generative models is often challenging. Per-pixel metrics like in Reed et al. (2017); Rezende et al. (2016) may penalize generative variability that still preserves meaning. We propose an Inception Score (Salimans et al., 2016) inspired metric to quantify class-recognizability and generalization of generated examples. We train two separate

Table 3: Effect of pixel loss coefficient α on Omniglot few-shot classification

α_{max}	20-way 1-shot Acc.
0.00	87.17 ± 0.36
0.25	87.82 ± 0.36
0.50	**91.39** ± 0.31
0.75	90.59 ± 0.32
0.95	89.77 ± 0.32

Table 4: ResNet-101 45-way classification score on 1-shot generated sketches of seen and unseen classes.

Generation Method	Seen	Unseen
Original Data	97.66	96.09
Conv-VAE	76.28 ± 0.93	75.07 ± 0.84
SketchEmbedNet	**81.44** ± 0.95	**77.94** ± 1.07

ResNet-101 classifiers, each on a different set of 45 Quickdraw classes. One set was part of the training set of SketchEmbedNet (referred to as "seen") and the other set was held out during training (referred to as "unseen"). We then have SketchEmbedNet generate one-shot sketches from each set and have the corresponding classifier predict a class. The accuracy of the classifier on generated examples is compared with its training accuracy in Table 4. For a ResNet classifier, SketchEmbedNet generations are more recognizable for both classes seen and unseen classes.

8. Conclusion

Learning to draw is not only an artistic pursuit but also a cognitive abstraction of real-world visual concepts. We present a generalized drawing model capable of producing accurate sketches and visual summaries of open-domain natural images and demonstrate that the associated embeddings are visually informative through our state-of-the-art performance on unsupervised few-shot tasks. While sketch data may be challenging to source, we show that training to draw either sketch or natural images can generalize not only within each domain but also well beyond the training data. More generally research in this direction may lead to more lifelike image understanding inspired by how humans communicate visual concepts.

Acknowledgments

We thank Jake Snell, James Lucas and Robert Adragna for their helpful feedback on earlier drafts of the manuscript. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute (www.vectorinstitute.ai/#partners). This project is supported by NSERC and the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Antreas Antoniou and Amos J. Storkey. Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation. *CoRR*, abs/1902.09884, 2019.

Pablo Arbelaez, Michael Maire, Charless C. Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, 2011.

David Berthelot, Colin Raffel, Aurko Roy, and Ian J. Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. In *7th International Conference on Learning Representations, ICLR*, 2019.

Ayan Kumar Bhunia, Yongxin Yang, Timothy M. Hospedales, Tao Xiang, and Yi-Zhe Song. Sketch less for more: On-the-fly fine-grained sketch-based image retrieval. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020.

Nan Cao, Xin Yan, Yang Shi, and Chaoran Chen. AI-Sketcher: A deep generative model for producing high-quality sketches. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, 2019.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *15th European Conference on Computer Vision, ECCV*, 2018.

Yajing Chen, Shikui Tu, Yuqi Yi, and Lei Xu. Sketch-pix2seq: a model to generate sketches of multiple categories. *CoRR*, abs/1709.04121, 2017.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Sounak Dey, Pau Riba, Anjan Dutta, Josep Llados, and Yi-Zhe Song. Doodle to search: Practical zero-shot sketch-based image retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2019.

Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *IEEE International Conference on Computer Vision, ICCV*, 2015.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *5th International Conference on Learning Representations, ICLR*, 2017.

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. 1973.

Anjan Dutta and Zeynep Akata. Semantically tied paired cycle consistency for zero-shot sketch-based image retrieval. *CoRR*, abs/1903.03372, 2019.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD*, 1996.

Kawin Ethayarajh, D. Duvenaud, and Graeme Hirst. Towards understanding linear word analogies. In *ACL*, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.

Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, Alex Lavin, and D. Scott Phoenix. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368), 2017. ISSN 0036-8075. doi: 10.1126/science.aag2612.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *6th International Conference on Learning Representations, ICLR*, 2018.

Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27, NIPS*, 2014.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

David Ha and Douglas Eck. A neural representation of sketch drawings. In *6th International Conference on Learning Representations, ICLR*, 2018.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

Luke B. Hewitt, Maxwell I. Nye, Andreea Gane, Tommi S. Jaakkola, and Joshua B. Tenenbaum. The variational homoencoder: Learning to learn high capacity generative models from few examples. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI*, 2018.

Geoffrey E. Hinton and Vinod Nair. Inferring motor programs from images of handwritten digits. In *Advances in Neural Information Processing Systems 18, NIPS*, 2005.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. In *7th International Conference on Learning Representations, ICLR*, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The quick, draw! - A.I. experiment., 2016. URL https://quickdraw.withgoogle.com/.

Siavash Khodadadeh, Ladislau Bölöni, and Mubarak Shah. Unsupervised meta-learning for few-shot image classification. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR*, 2014.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. doi: 10.1126/science.aab3050.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. The omniglot challenge: a 3-year progress report. *Current Opinion in Behavioral Sciences*, 29:97–104, Oct 2019.

Mengtian Li, Zhe L. Lin, Radomír Mech, Ersin Yumer, and Deva Ramanan. Photo-sketching: Inferring contour drawings from images. In *IEEE Winter Conference on Applications of Computer Vision, WACV*, 2019.

M. Liwicki and H. Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition, ICDAR*, 2005.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *14th European Conference on Computer Vision, ECCV*, 2016.

Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018.

Anubha Pandey, Ashish Mishra, Vinay Kumar Verma, Anurag Mittal, and Hema A. Murthy. Stacked adversarial network for zero-shot sketch based image retrieval. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision, WACV*, 2020.

Marc'Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19, NIPS*, 2006.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR*, 2017.

Scott E. Reed, Yutian Chen, Thomas Paine, Aäron van den Oord, S. M. Ali Eslami, Danilo Jimenez Rezende, Oriol Vinyals, and Nando de Freitas. Few-shot autoregressive density estimation: Towards learning to learn distributions. *CoRR*, abs/1710.10304, 2017.

Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016.

Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29, NIPS*, 2016.

Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: learning to retrieve badly drawn bunnies. *ACM Trans. Graph.*, 35(4):119:1–119:12, 2016.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30, NIPS*, 2017.

Jifei Song, Kaiyue Pang, Yi-Zhe Song, Tao Xiang, and Timothy M. Hospedales. Learning to sketch with shortcut cycle consistency. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems 29, NIPS*, 2016.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29, NIPS*, 2016.

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *Int. J. Comput. Vis.*, 125(1-3): 3–18, 2017.

Qian Yu, Feng Liu, Yi-Zhe Song, Tao Xiang, Timothy M. Hospedales, and Chen Change Loy. Sketch me that shoe. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

Liliang Zhang, Liang Lin, Xian Wu, Shengyong Ding, and Lei Zhang. End-to-end photo-sketch generation via fully convolutional representation learning. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, ICMR*, 2015.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *14th European Conference on Computer Vision, ECCV*, 2016.

Appendix A. Rasterization

The key enabler of our novel pixel loss for sketch drawings is our differentiable rasterization function f_{raster}. Sequence based loss functions such as $\mathcal{L}_{\text{stroke}}$ are sensitive to the order of points while in reality, drawings are sequence invariant. Visually, a square is a square whether it is drawn clockwise or counterclockwise.

The purpose of a sketch representation is to lower the complexity of the data space and decode in a more visually intuitive manner. While it is a necessary departure point, the sequential generation of drawings is not key to our visual representation and we would like SketchEmbedNet to be agnostic to any specific sequence needed to draw the sketch that is representative of the image input.

To facilitate this, we develop our rasterization function f_{raster} which renders an input sequence of strokes as a pixel image. However, during training, the RNN outputs a mixture of Gaussians at each timestep. To convert this to a stroke sequence, we sample from these Gaussians; this can be repeated to reduce the variance of the pixel loss. We then scale our predicted and ground truth sequences by the properties of the latter before rasterization.

Stroke sampling At the end of sequence generation we have $N_s \times (6M + 3)$ parameters, 6 Gaussian mixture parameters, 3 pen states, N_s times, one for each stroke. To obtain the actual drawing we sample from the mixture of Gaussians:

$$\begin{bmatrix}\Delta x_t \\ \Delta y_t\end{bmatrix} = \begin{bmatrix}\mu_{x,t} \\ \mu_{y,t}\end{bmatrix} + \begin{bmatrix}\sigma_{x,t} & 0 \\ \rho_{xy,t}\sigma_{y,t} & \sigma_{y,t}\sqrt{1-\rho_{xy,t}^2}\end{bmatrix} \boldsymbol{\epsilon}\,,\, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_2). \tag{5}$$

After sampling we compute the cumulative sum of every stroke over the timestep so that we obtain the absolute displacement from the initial position:

$$\begin{bmatrix}x_t \\ y_t\end{bmatrix} = \sum_{\tau=0}^{T} \begin{bmatrix}\Delta x_\tau \\ \Delta y_\tau\end{bmatrix}. \tag{6}$$

$$\boldsymbol{y}_{t,abs} = (x_t, y_t, s_1, s_2, s_3). \tag{7}$$

Scaling Each sketch generated by our model begins at (0,0) and the variance of all strokes in the training set is normalized to 1. On a fixed canvas the image is both very small and localized to the top left corner. We remedy this by computing a scale λ and shift $x_{\text{shift}}, y_{\text{shift}}$ using labels $\boldsymbol{y}$ and apply them to both the prediction $\boldsymbol{y}'$ as well as the ground truth $\boldsymbol{y}$. These parameters are computed as:

$$\lambda = \min\left\{\frac{W}{x_{\max} - x_{\min}}, \frac{H}{y_{\max} - y_{\min}}\right\}, \tag{8}$$

$$x_{\text{shift}} = \frac{x_{\max} + x_{\min}}{2}\lambda, \quad y_{\text{shift}} = \frac{y_{\max} + y_{\min}}{2}\lambda. \tag{9}$$

$x_{\text{max}}, x_{\text{min}}, y_{\text{max}}, y_{\text{min}}$ are the minimum and maximum values of x_t, y_t from the supervised stroke labels and not the generated strokes. W and H are the width and height in pixels of our output canvas.

Calculate pixel intensity Finally we are able to calculate the pixel p_{ij} intensity of every pixel in our $H \times W$ canvas.

$$p_{ij} = \sigma\left[2 - 5 \times \min_{t=1\ldots N_s}\left(\text{dist}\big((i,j),(x_{t-1},y_{t-1}),(x_t,y_t)\big) + (1 - \lfloor s_{1,t-1}\rceil)10^6\right)\right], \tag{10}$$

where the distance function is the distance between point (i, j) from the line segment defined by the absolute points (x_{t-1}, y_{t-1}) and (x_t, y_t). We also blow up any distances where $s_{1,t-1} < 0.5$ so as to not render any strokes where the pen is not touching the paper.

Appendix B. Implementation details

We train our model for 300k iterations with a batch size of 256 for the Quickdraw dataset and 64 for Sketchy due to memory constraints. The initial learning rate is 1e-3 which decays by 0.85 every 15k steps. We use the Adam (Kingma and Ba, 2015) optimizer and clip gradient values at 1.0. $\sigma = 2.0$ is used for the Gaussian blur in $\mathcal{L}_{\text{pixel}}$. For the curriculum learning schedule, the value of α is set to 0 initially and increases by 0.05 every 10k training steps with an empirically obtained cap at $\alpha_{\text{max}} = 0.50$ for Quickdraw and $\alpha_{\text{max}} = 0.75$ for Sketchy.

The ResNet12 encoder uses 4 ResNet blocks with 64, 128, 256, 512 filters respectively and ReLU activations. The Conv4 backbone has 4 blocks of convolution, batch norm (Ioffe and Szegedy, 2015), ReLU and max pool, identical to Vinyals et al. (2016). We select the latent space to be 256 dimensions, RNN output size to be 1024, and the hypernetwork embedding size to be 64. We use a mixture of $M = 30$ bivariate Gaussians for the mixture density output of the stroke offset distribution.

Appendix C. Latent Space Interpolation

Like in many encoding-decoding models we evaluate the interpolation of our latent space. We select 4 embeddings at random and use bi-linear interpolation to produce new embeddings. Results are in Figures 7 and 8.

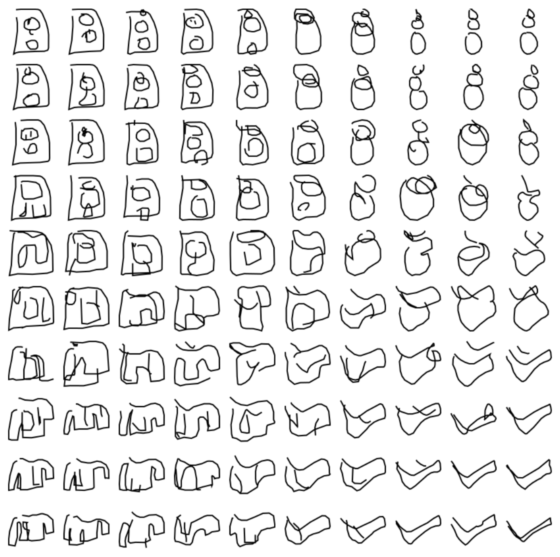


Figure 7: Interpolation of classes: power outlet, snowman, jacket, elbow

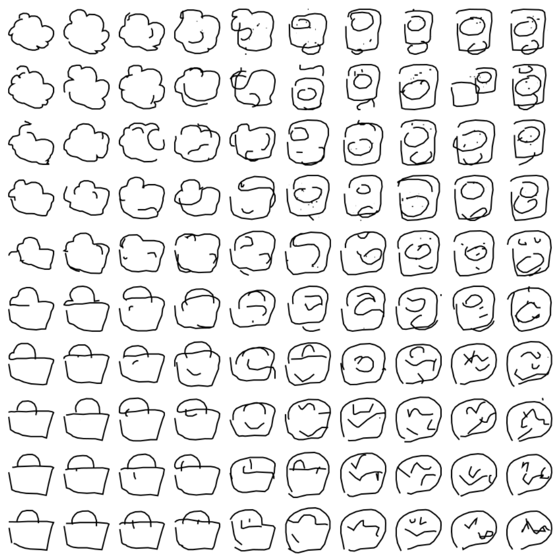


Figure 8: Interpolation of classes: cloud, power outlet, basket, compass

Figure 9: Latent space interpolations of randomly selected examples

We observe that compositionality is also present in these interpolations. In the top row of Figure 7, the model first plots a third small circle when interpolating from the 2-circle power outlet and the 3-circle snowman. This small circle is treated as single component that grows as it transitions between classes until it's final size in the far right snowman drawing.

Some other RNN-based sketching models (Chen et al., 2017; Ha and Eck, 2018) experience other classes materializing in interpolations between two unrelated classes. Our model does not exhibit this same behaviour as our embedding space is learned from more classes and thus does not contain local groupings of classes.

Appendix D. Effect of α on Few-Shot Classification

We performed additional experiments exploring the impact of our curriculum training schedule for α. The encoding component of our drawing model was evaluated on the few-shot classification task for different values of α_{max} every 25k iterations during training. A graph is shown in Figure 10 and the full table of all values of α_{max} is in Table 5.

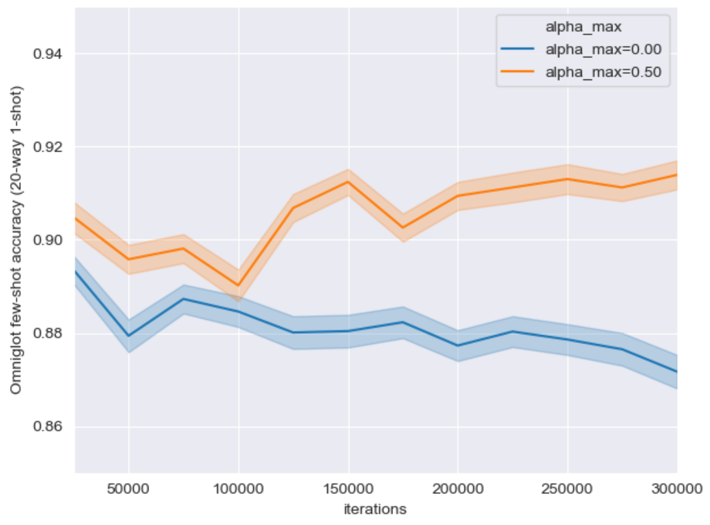


Figure 10: Few-shot classification accuracy of α_{max} values 0.0 and 0.5 over training

Table 5: Few-shot classification accuracy of all α_{max} values

α_{max}	**25k**	**50k**	**75k**	**100k**	**125k**	**150k**	**175k**	**200k**	**225k**	**250k**	**275k**	**300k**
0.00	89.35	87.94	88.73	88.46	88.01	88.04	88.23	87.73	88.03	87.86	87.65	87.17
0.25	89.21	90.39	90.20	89.75	87.78	88.37	88.64	88.05	87.98	88.41	88.15	87.82
0.50	90.48	89.58	89.81	89.02	90.68	91.24	90.26	90.94	91.12	91.30	91.12	91.39
0.75	91.39	89.95	89.56	89.81	89.95	90.79	91.02	91.09	91.82	90.76	91.42	90.59
0.95	90.23	90.15	90.10	89.55	90.27	92.37	92.27	90.29	91.58	91.02	89.73	89.77

Appendix E. Intra-alphabet Lake Split

The creators of the Omniglot dataset and one-shot classification benchmark originally proposed an intra-alphabet classification task. This task is more challenging than the common Vinyals split as characters from the same alphabet may exhibit similar stylistics of sub-components that makes visual differentiation more difficult. This benchmark has been less explored by researchers; however, we still present the performance of our SketchEmbedding model against evaluations of other few-shot classification models on the benchmark. Results are shown in Table 6.

Table 6: Few-shot classification results on Omniglot (Lake split)

Omniglot (Lake split)			**(way, shot)**			
Algorithm	**Backbone**	**Train Data**	**(5,1)**	**(5,5)**	**(20,1)**	**(20,5)**
Conv-VAE	Conv4	Quickdraw	73.12 ± 0.58	88.50 ± 0.39	53.45 ± 0.51	73.62 ± 0.48
SketchEmbedding *(Ours)*	Conv4	Quickdraw	89.16 ± 0.41	97.12 ± 0.18	74.24 ± 0.48	89.87 ± 0.25
SketchEmbedding *(Ours)*	ResNet12	Quickdraw	**91.03** ± 0.37	**97.91** ± 0.15	**77.94** ± 0.44	**92.49** ± 0.21
BPL *(Supervised)* (Lake et al., 2015, 2019)	N/A	Omniglot	-	-	96.70	-
ProtoNet *(Supervised)* (Snell et al., 2017)	Conv4	Omniglot	-	-	86.30	-
RCN *(Supervised)* (George et al., 2017)	N/A	Omniglot	-	-	92.70	-
VHE *(Supervised)* (Hewitt et al., 2018)	N/A	Omniglot	-	-	81.30	-

Unsurprisingly, our model is outperformed by supervised models and does fall behind by a more substantial margin than in the Vinyals split. However, our SketchEmbedding approach still achieves respectable classification accuracy overall and greatly outperforms a Conv-VAE baseline.

Appendix F. Latent Variable Recovery: Additional Results

We perform retrieval on the latent variable in the spatial relationships section with different amounts of data (100, 1000) as well as with a non-linear model. We also provide mean squared error as a metric.

Table 7: Spatial latent retrieval (1000)

	Linear		**Non-Linear**	
	R^2	MSE	R^2	MSE
Angle (SketchEmbedding)	0.99	0.0013	0.98	0.0012
Angle (VAE)	0.11	0.0793	0.95	0.0432
Distance (SketchEmbedding)	0.97	0.090	0.98	0.058
Distance (VAE)	0.28	2.287	0.97	0.092
Size (SketchEmbedding)	0.97	0.69	0.98	0.48
Size (VAE)	0.20	21.72	0.98	0.44

Table 8: Spatial latent retrieval (100)

	Linear		**Non-Linear**	
	R^2	MSE	R^2	MSE
Angle (SketchEmbedding)	0.96	0.0037	0.95	0.0047
Angle (VAE)	0.00	0.0969	0.82	0.0173
Distance (SketchEmbedding)	0.82	0.585	0.96	0.14
Distance (VAE)	0.00	3.24	0.91	0.28
Size (SketchEmbedding)	0.95	1.20	0.87	3.53
Size (VAE)	0.02	26.70	0.85	4.01

We find that not only are SketchEmbeddings more effective with the same amount of data but also allow for identifying the latent spatial variable with far fewer examples.

Appendix G. Effect of Random Seeds on Few-Shot Classification

The training objective for SketchEmbedNet is to reproduce sketch drawings of the input. This task is unrelated to few-shot classification, where its performance may vary given different initialization. We quantify this variance by training our model with 15 unique random seeds and evaluating the performance of the latent space on the few-shot classification tasks.

We disregard the per (evaluation) episode variance of our model in each test stage and only present the mean accuracy. We then compute a new confidence interval over random seeds. Results are presented in Tables 9, 10, 11.

Table 9: Random Seeding on Few-Shot Classification results on Omniglot (Conv4)

	(way, shot)			
Seed	**(5,1)**	**(5,5)**	**(20,1)**	**(20,5)**
1	96.45	99.41	90.84	98.08
2	96.54	99.48	90.82	98.10
3	96.23	99.40	90.05	97.94
4	96.15	99.46	90.50	97.99
5	96.21	99.40	90.54	98.10
6	96.08	99.43	90.20	97.93
7	96.19	99.39	90.70	98.05
8	96.68	99.44	91.11	98.18
9	96.49	99.42	90.64	98.06
10	96.37	99.47	90.50	97.99
11	96.52	99.40	91.13	98.18
12	**96.96**	**99.50**	**91.67**	**98.30**
13	96.31	99.38	90.57	98.04
14	96.12	99.45	90.54	98.03
15	96.30	99.48	90.62	98.05
Average	96.37 $\pm$ 0.12	99.43 $\pm$ 0.02	90.69 $\pm$ 0.20	98.07 $\pm$ 0.05

Table 10: Random Seeding on Few-Shot Classification results on Omniglot (ResNet12)

	(way, shot)			
Seed	**(5,1)**	**(5,5)**	**(20,1)**	**(20,5)**
1	**96.61**	**99.58**	**91.25**	**98.58**
2	96.37	99.52	90.44	98.40
3	96.04	99.58	89.86	98.27
4	96.44	99.50	90.76	98.40
5	95.95	99.52	89.88	98.29
6	95.63	99.45	89.28	98.17
7	96.24	99.52	89.90	98.34
8	95.41	99.45	88.75	98.05
9	96.04	99.49	89.70	98.24
10	95.40	99.41	88.91	98.05
11	95.82	99.51	89.67	98.24
12	96.25	99.51	90.21	98.28
13	95.84	99.53	89.71	98.18
14	96.04	99.56	89.89	98.31
15	96.04	99.57	89.97	98.32
Average	96.00 $\pm$ 0.31	99.51 $\pm$ 0.04	89.89 $\pm$ 0.56	98.27 $\pm$ 0.12

Table 11: Random Seeding on Few-Shot Classification results on mini-ImageNet

	(way, shot)			
Seed	**(5,1)**	**(5,5)**	**(5,20)**	**(5,50)**
1	37.15	52.99	63.92	68.72
2	39.38	55.20	65.60	69.79
3	39.40	55.47	65.94	70.41
4	**40.39**	**57.15**	**67.60**	**71.99**
5	38.40	54.08	65.36	70.08
6	37.94	53.98	65.24	69.65
7	38.88	55.71	66.59	71.35
8	37.89	52.65	63.42	68.14
9	38.25	53.86	65.02	69.82
10	39.11	55.29	65.99	69.98
11	37.39	52.88	63.66	68.33
12	38.24	53.91	65.19	69.82
13	38.62	53.84	63.83	68.69
14	37.73	53.61	64.22	68.41
15	39.50	55.23	65.51	70.25
Average	38.55 ± 0.45	54.39 ± 0.63	65.14 ± 0.59	69.69 ± 0.56

Appendix H. Data processing

H.1. Quickdraw

We apply the same data processing methods as in Ha and Eck (2018) with no additional changes to produce our stroke labels $\boldsymbol{y}$. When rasterizing for our input $\boldsymbol{x}$, we scale, center the strokes then pad the image with 10% of the resolution in that dimension rounded to the nearest integer.

H.2. Omniglot

We derive our Omniglot tasks from the stroke dataset originally provided by Lake et al. (2015) rather than the image analogues. We translate the Omniglot stroke-by-stroke format to the same one used in Quickdraw. Then we apply the Ramer-Douglas-Peucker (Douglas and Peucker, 1973) algorithm with an epsilon value of 2 and normalize variance to 1 to produce $\boldsymbol{y}$. We also rasterize our images in the same manner as above for our input $\boldsymbol{x}$.

H.3. Sketchy

Sketchy data is provided as an SVG image composed of line paths that are either straight lines or Bezier curves. To generate stroke data we sample sequences of points from Bezier curves at a high resolution that we then simplify with RDP, $\epsilon = 5$. We also eliminate continuous strokes with a short path length or small displacement to reduce our stroke length and remove small and noisy strokes. Path length and displacement are considered with respect to the scale of the entire sketch.

Table 12: Model comparisons between generative autoregressive models that produce pixel or vector sketch drawings.

Autoregressive sketching models					
Model	**Dataset**	**# classes**	**Encoder**	**Decoder**	Loss function
Handwriting Sequence (Graves, 2013)	IAM-OnDB (Liwicki and Bunke, 2005)	1	RNN	Mixture Density RNN	$\mathcal{L}_{\text{stroke}}$
DRAW (Gregor et al., 2015)	SVHN, MNIST	10	RNN	RNN	$\mathcal{L}_{\text{pixel}} + \mathcal{L}_{\text{KL}}$
Sketch-RNN (Ha and Eck, 2018)	Quickdraw	1	Bi-directional RNN	Mixture Density RNN	$\mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{stroke}} + \mathcal{L}_{\text{KL}}$
Sketch-pix2seq (Chen et al., 2017)	Quickdraw	3, 6	simple CNN	Mixture Density RNN	$\mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{stroke}}$
AI-Sketcher (Cao et al., 2019)	Quickdraw, FaceX (Cao et al., 2019)	5, 10, 15, 20	Bi-directional RNN + CNN Autoencoder	Mixture Density RNN	$\mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{stroke}} + \mathcal{L}_{\text{KL}}$
deep_p2s (Song et al., 2018)	Quickdraw, ShoesV2, ChairV2 (Yu et al., 2016)	1	Bi-directional RNN, CNN	CNN, Mixture Density RNN	$\mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{stroke}} + \mathcal{L}_{\text{l2}} + \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{shortcut}}$
SketchEmbedding *(ours)*	Quickdraw	300	ResNet12	Mixture Density RNN	$\mathcal{L}_{\text{pen}} + \mathcal{L}_{\text{stroke}} + \mathcal{L}_{\text{pixel}}$

Once again we normalize stroke variance and rasterize for our input image in the same manners as above.

Appendix I. Autoregressive drawing model comparisons

We summarize the key components of SketchEmbedNet in comparison to other autoregressive drawing models in Table 12.

Appendix J. Few-shot Classification on Omniglot – Full Results

The full results table for few-shot classification on the Omniglot dataset, including the ResNet12 model.

Table 13: Few-shot classification results on Omniglot

Omniglot			**(way, shot)**			
Algorithm	**Backbone**	**Train Data**	**(5,1)**	**(5,5)**	**(20,1)**	**(20,5)**
Training from Scratch (Hsu et al., 2019)	N/A	Omniglot	52.50 ± 0.84	74.78 ± 0.69	24.91 ± 0.33	47.62 ± 0.44
k-NN (Hsu et al., 2019)	N/A	Omniglot	57.46 ± 1.35	81.16 ± 0.57	39.73 ± 0.38	66.38 ± 0.36
Linear Classifier (Hsu et al., 2019)	N/A	Omniglot	61.08 ± 1.32	81.82 ± 0.58	43.20 ± 0.69	66.33 ± 0.36
MLP + Dropout (Hsu et al., 2019)	N/A	Omniglot	51.95 ± 0.82	77.20 ± 0.65	30.65 ± 0.39	58.62 ± 0.41
Cluster Matching (Hsu et al., 2019)	N/A	Omniglot	54.94 ± 0.85	71.09 ± 0.77	32.19 ± 0.40	45.93 ± 0.40
CACTUs-MAML (Hsu et al., 2019)	Conv4	Omniglot	68.84 ± 0.80	87.78 ± 0.50	48.09 ± 0.41	73.36 ± 0.34
CACTUs-ProtoNet (Hsu et al., 2019)	Conv4	Omniglot	68.12 ± 0.84	83.58 ± 0.61	47.75 ± 0.43	66.27 ± 0.37
AAL-ProtoNet (Antoniou and Storkey, 2019)	Conv4	Omniglot	84.66 ± 0.70	88.41 ± 0.27	68.79 ± 1.03	74.05 ± 0.46
AAL-MAML (Antoniou and Storkey, 2019)	Conv4	Omniglot	88.40 ± 0.75	98.00 ± 0.32	70.20 ± 0.86	88.30 ± 1.22
UMTRA (Khodadadeh et al., 2019)	Conv4	Omniglot	83.80	95.43	74.25	92.12
Random CNN	Conv4	N/A	67.96 ± 0.44	83.85 ± 0.31	44.39 ± 0.23	60.87 ± 0.22
Conv-VAE	Conv4	Omniglot	77.83 ± 0.41	92.91 ± 0.19	62.59 ± 0.24	84.01 ± 0.15
Conv-VAE	Conv4	Quickdraw	81.49 ± 0.39	94.09 ± 0.17	66.24 ± 0.23	86.02 ± 0.14
SketchEmbedding *(Ours)*	Conv4	Omniglot	94.88 ± 0.22	99.01 ± 0.08	86.18 ± 0.18	96.69 ± 0.07
SketchEmbedding-avg *(Ours)*	Conv4	Quickdraw	96.37	99.43	90.69	98.07
SketchEmbedding-best *(Ours)*	Conv4	Quickdraw	**96.96** ± 0.17	**99.50** ± 0.06	**91.67** ± 0.14	98.30 ± 0.05
SketchEmbedding-avg *(Ours)*	ResNet12	Quickdraw	96.00	99.51	89.88	98.27
SketchEmbedding-best *(Ours)*	ResNet12	Quickdraw	96.61 ± 0.19	**99.58** ± 0.06	91.25 ± 0.15	**98.58** ± 0.05
MAML *(Supervised)* (Finn et al., 2017)	Conv4	Omniglot	94.46 ± 0.35	98.83 ± 0.12	84.60 ± 0.32	96.29 ± 0.13
ProtoNet *(Supervised)* (Snell et al., 2017)	Conv4	Omniglot	98.35 ± 0.22	99.58 ± 0.09	95.31 ± 0.18	98.81 ± 0.07

Appendix K. HyperNetwork Activations

To further explore how our network understands drawings, we examine the relationships between the activations of the hypernetwork of our HyperLSTM (Ha et al., 2017).

The hypernetwork determines the weights of the LSTM that generates the RNN at each decoding timestep. These activations are 512-dimensional vectors. We collect the activations from many examples, cluster them in 512-dimensional space and visualize the strokes belonging to each cluster for each example. A full decoding is also rendered where each cluster within an example is assigned a color.

Single class: snowman First we explore this clustering using only the snowman class from Quickdraw (Jongejan et al., 2016). We expect substantial reuse of a "circle" both within and over many examples. Clustering of the strokes is done with the DBSCAN (Ester et al., 1996) and parameter $\epsilon = 3.9$. Results are in Figure 11. Each row is a separate input; the far left column is the color-coded, composed image, the second is the noise cluster and every subsequent column is a unique cluster.

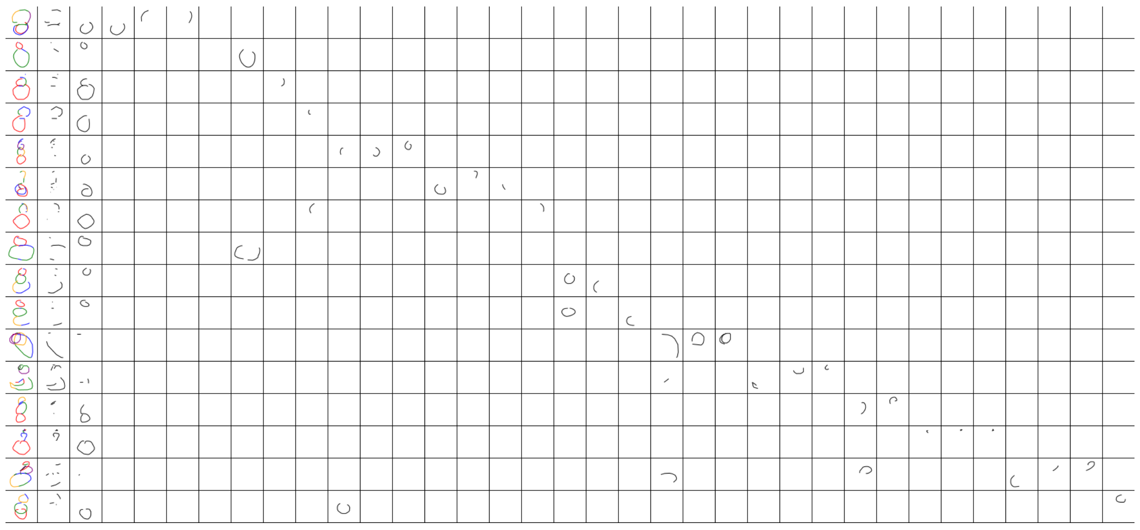


Figure 11: Snowman class stroke clustering

While cluster re-use is limited, cluster 0 often contains a large, fully enclosed circle. Many other clusters may contain circles or partial strokes with some reuse. Larger, fully composed and coloured sketches are presented in Figure 12

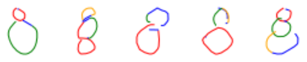


Figure 12: Fully composed images with coloured cluster assignments

Many classes: round objects We repeat the above experiment with a mixture of classes that generally can be expected to contain circles. These classes were circles, snowmen, clocks and cups. The two former classes are frequently composed only of circles while the latter are expected to consistently contain other distinct shapes. Results are presented in Figure 13 and select examples in Figure 14.

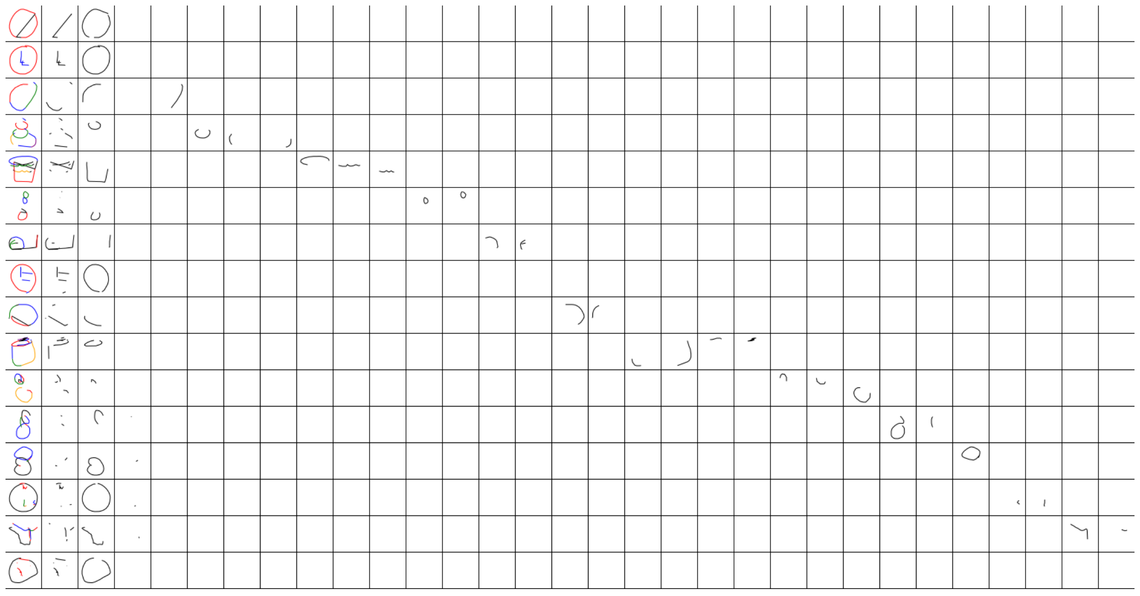


Figure 13: Snowman class stroke clustering

We still observe that the model continues to isolate circles in the first column and note it continues to do so for the cup and clock classes which are not exclusively circular.

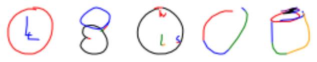


Figure 14: Fully composed images with coloured cluster assignments

Many random classes: Finally, we repeat the above clustering with the 45 randomly selected holdout classes from the Quickdraw training process of SketchEmbedding. Results are once again presented in Figure 15 and select examples in Figure 16.

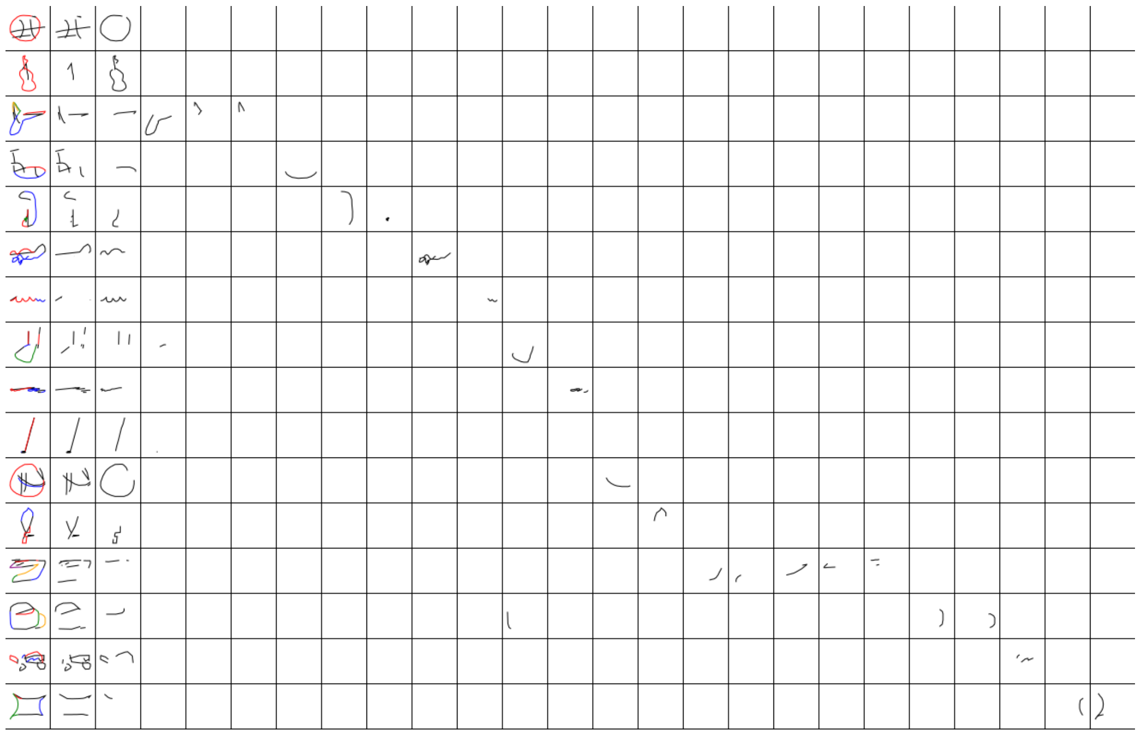


Figure 15: Snowman class stroke clustering

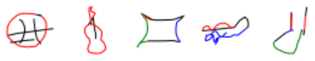


Figure 16: Fully composed images with coloured cluster assignments