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.

One purpose of the 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:

$$\begin{bmatrix}\Delta x_t \\ \Delta y_t\end{bmatrix} = \begin{bmatrix}\mu_{x,t} \\ \mu_{y,t}\end{bmatrix} + \begin{bmatrix}\sigma_{x,t} & 0 \\ \rho_{xy,t}\sigma_{y,t} & \sigma_{y,t}\sqrt{1-\rho_{xy,t}^2}\end{bmatrix}\boldsymbol{\epsilon} \quad (7)$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_2). \quad (8)$$

After sampling we compute the cumulative sum of every stroke over the time so that we obtain an absolute position at each timestep:

$$\begin{bmatrix}x_t \\ y_t\end{bmatrix} = \sum_{\tau=0}^{T} \begin{bmatrix}\Delta x_\tau \\ \Delta y_\tau\end{bmatrix}. \quad (9)$$

$$\boldsymbol{y}_{t,abs} = (x_t, y_t, s_1, s_2, s_3). \quad (10)$$

Sketch 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:

$$\lambda = \min\left\{\frac{W}{x_{\text{max}} - x_{\text{min}}}, \frac{H}{y_{\text{max}} - y_{\text{min}}}\right\}, \quad (11)$$

$$x_{\text{shift}} = \frac{x_{\text{max}} + x_{\text{min}}}{2}\lambda, \quad y_{\text{shift}} = \frac{y_{\text{max}} + y_{\text{min}}}{2}\lambda. \quad (12)$$

$x_{\text{max}}, x_{\text{min}}, y_{\text{max}}, y_{\text{min}}$ 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.

$$p_{ij} = \sigma\left[2 - 5 \times \min_{t=1\ldots N_s}\Big(\quad (13)$$

$$\text{dist}\big((i,j),(x_{t-1},y_{t-1}),(x_t,y_t)\big) + (1 - \lfloor s_{1,t-1} \rceil)10^6\Big)\Big], \quad (14)$$

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.

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 & 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 (Oreshkin et al., 2018) 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 & 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.

C. Data Processing

C.1. Quickdraw

We apply the same data processing methods as in Ha & 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.

The following list of classes were used for training: The Eiffel Tower, The Mona Lisa, aircraft carrier, alarm clock, ambulance, angel, animal migration, ant, apple, arm, asparagus, banana, barn, baseball, baseball bat, bathtub, beach, bear, bed, bee, belt, bench, bicycle, binoculars, bird, blueberry, book, boomerang, bottlecap, bread, bridge, broccoli, broom, bucket, bulldozer, bus, bush, butterfly, cactus, cake, calculator, calendar, camel, camera, camouflage, campfire, candle, cannon, car, carrot, castle, cat, ceiling fan, cell phone, cello, chair, chandelier, church, circle, clarinet, clock, coffee cup, computer, cookie, couch, cow, crayon, crocodile, crown, cruise ship, diamond, dishwasher, diving board, dog, dolphin, donut, door, dragon, dresser, drill, drums, duck, dumbbell, ear, eye, eyeglasses, face, fan, feather, fence, finger, fire hydrant, fireplace, firetruck, fish, flamingo, flashlight, flip flops, flower, foot, fork, frog, frying pan, garden, garden hose, giraffe, goatee, grapes, grass, guitar, hamburger, hand, harp, hat, headphones, hedgehog, helicopter, helmet, hockey puck, hockey stick, horse, hospital, hot air balloon, hot dog, hourglass, house, house plant, ice cream, key, keyboard, knee, knife, ladder, lantern, leaf, leg, light bulb, lighter, lighthouse, lightning, line, lipstick, lobster, mailbox, map, marker, matches, megaphone, mermaid, microphone, microwave, monkey, mosquito, motorbike, mountain, mouse, moustache, mouth, mushroom, nail, necklace, nose, octopus, onion, oven, owl, paint can, paintbrush, palm tree, parachute, passport, peanut, pear, pencil, penguin, piano, pickup truck, pig, pineapple, pliers, police car, pool, popsicle, postcard, purse, rabbit, raccoon, radio, rain, rainbow, rake, remote control, rhinoceros, river, rollerskates, sailboat, sandwich, saxophone, scissors, see saw, shark, sheep, shoe, shorts, shovel, sink, skull, sleeping bag, smiley face, snail, snake, snowflake, soccer ball, speedboat, square, star, steak, stereo, stitches, stop sign, strawberry, streetlight, string bean, submarine, sun, swing set, syringe, t-shirt, table, teapot, teddy-bear, tennis racquet, tent, tiger, toe, tooth, toothpaste, tractor, traffic light, train, triangle, trombone, truck, trumpet, umbrella, underwear, van, vase, watermelon, wheel, windmill, wine bottle, wine glass, wristwatch, zigzag, blackberry, power outlet, peas, hot tub, toothbrush, skateboard, cloud, elbow, bat, pond, compass, elephant, hurricane, jail, school bus, skyscraper, tornado, picture frame, lollipop, spoon, saw, cup, roller coaster, pants, jacket, rifle, yoga, toilet, waterslide, axe, snowman, bracelet, basket, anvil, octagon, washing machine, tree, television, bowtie, sweater, backpack, zebra, suitcase, stairs, The Great Wall of China

C.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 & 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 $\boldsymbol{x}$.

C.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.

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

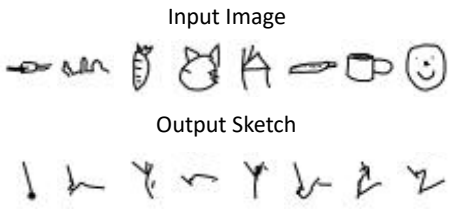
The following classes were use for training after removing overlapping classes with mini-ImageNet: hot-air_balloon, violin, tiger, eyeglasses, mouse, jack-o-lantern, lobster, teddy_bear, teapot, helicopter, duck, wading_bird, rabbit, penguin, sheep, windmill, piano, jellyfish, table, fan, beetle, cabin, scorpion, scissors, banana, tank, umbrella, crocodilian, volcano, knife, cup, saxophone, pistol, swan, chicken, sword, seal, alarm_clock, rocket, bicycle, owl, squirrel, hermit_crab, horse, spoon, cow, hotdog, camel, turtle, pizza, spider, songbird, rifle, chair, starfish, tree, airplane, bread, bench, harp, seagull, blimp, apple, geyser, trumpet, frog, lizard, axe, sea_turtle, pretzel, snail, butterfly, bear, ray, wine_bottle, , elephant, raccoon, rhinoceros, door, hat, deer, snake, ape, flower, car_(sedan), kangaroo, dolphin, hamburger, castle, pineapple, saw, zebra, candle, cannon, racket, church, fish, mushroom, strawberry, window, sailboat, hourglass, cat, shoe, hedgehog, couch, giraffe, hammer, motorcycle, shark

D. Pixel-loss Weighting $\alpha_{\max}$ Ablation for Generation Quality

Table 6. Effect of $\alpha_{\max}$ on classification accuracy of generated sketches.

$\alpha_{\max}$	0.00	0.25	0.50	0.75	0.95	1.00
Seen	87.76	87.35	81.44	66.80	36.98	04.80
Unseen	84.02	85.32	77.94	63.10	32.94	04.50

(a) Autoregressive generation.



(b) Teacher-forced generation.

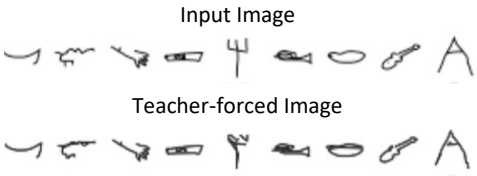


Figure 9. Sketches of SketchEmbedNet trained with $\alpha_{\max} = 1.0$.

We also ablate the impact of pixel-loss weighting parameter $\alpha_{\max}$ on the classification accuracy of the ResNet models from Section 4.5. The evaluation process is the same, generating sketches of examples from classes that were either seen during training or new to the model and classifying them in 45-way classification. Results are shown in Table 6.

Results are only shown for the Quickdraw (Jongejan et al., 2016) setting. Increasing pixel-loss weighting has a minor impact on classification accuracy at lower values but has a significant detriment at higher weightings. This is due to the teacher-forcing training process. As we de-weight

the stroke loss, the model no longer learns to handle the uncertainty of the input position in the space of the 2D canvas by predicting a distribution that explains the next ground truth point. It only matches the generation in pixel space and no longer generates a sensible stroke trajectory on the canvas. While training under teacher forcing, this is not an issue as it is fed the ground truth input point every time, but in autoregressive this generation quickly degrades as each step no longer produces the a point that is a meaningful input for the next time step. We can see the significant difference between generation quality under techer forcing and autoregressive generation in Figure 9.

E. 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 10a and 10b.

We observe that compositionality is also present in these interpolations. In the top row of Figure 10a, 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 (Ha & Eck, 2018; Chen et al., 2017) 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.

F. 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 7.

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.

G. Effect of Random Seeding on Few-Shot Classification

The training objective for SketchEmbedding is to reproduce sketch drawings of the input. This task is unrelated to few-shot classification may perform variably 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.

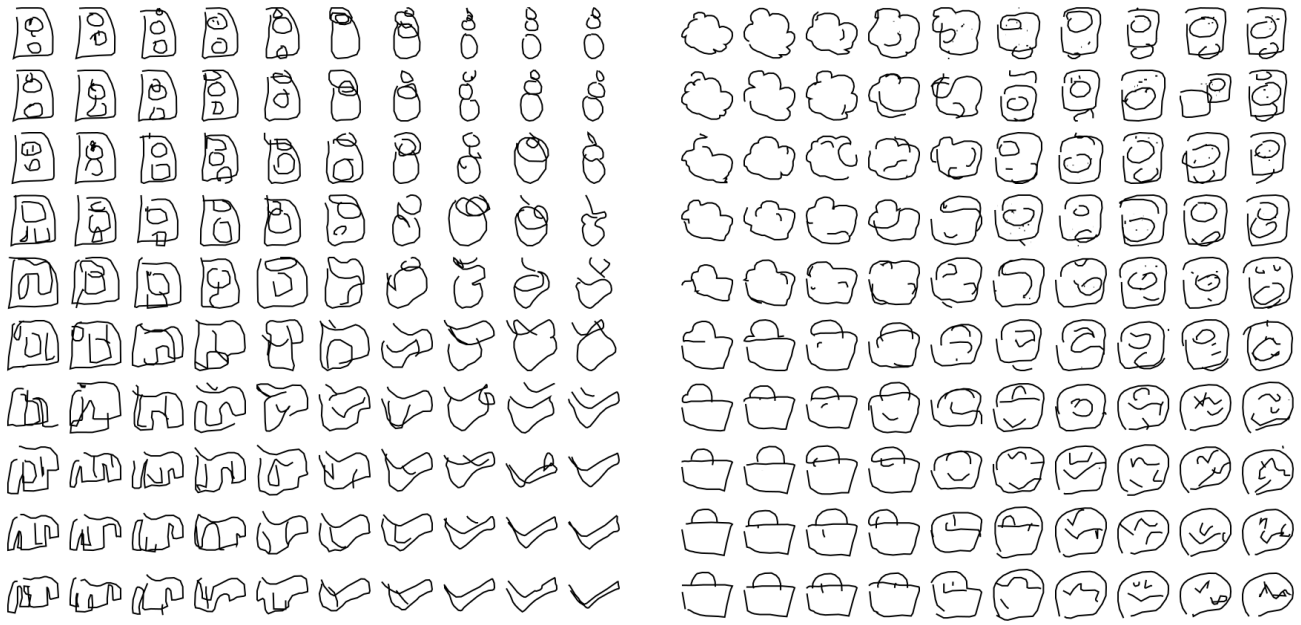
We disregard the per (evaluation) episode variance of our model in each test stage and only present the mean accuracy. We then compute a new confidence interval over random seeds. Results are presented in Tables 8a, 8b.

H. Few-shot Classification on Omniglot – Full Results.

The full results (Table 9) for few-shot classification on the Omniglot (Lake et al., 2015) dataset, including the ResNet12 (Oreshkin et al., 2018) model.

I. Few-shot Classification on mini-ImageNet – Full Results

The full results (Table 10) for few-shot classification on the mini-ImageNet dataset, including the ResNet12 (Oreshkin et al., 2018) model and Conv4 models.



(a) Interpolation of classes: power outlet, snowman, jacket, elbow.

(b) Interpolation of classes: cloud, power outlet, basket, compass.

Figure 10. Latent space interpolations of randomly selected examples.

Table 7. 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; Lake et al., 2019)	Conv4	Omniglot	-	-	86.30	-
RCN *(Supervised)* (George et al., 2017; Lake et al., 2019)	N/A	Omniglot	-	-	92.70	-
VHE *(Supervised)* (Hewitt et al., 2018; Lake et al., 2019)	N/A	Omniglot	-	-	81.30	-

Table 8. Few-shot classification random seeding experiments.

(a) 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 ± 0.12	99.43 ± 0.02	90.69 ± 0.20	98.07 ± 0.05

(b) 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

Table 9. Full table of 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
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
Conv-AE	Conv4	Quickdraw	81.54 ± 0.40	93.57 ± 0.19	67.24 ± 0.24	84.15 ± 0.16
β-VAE ($\beta = 250$) (Higgins et al., 2017)	Conv4	Quickdraw	79.11 ± 0.40	93.23 ± 0.19	63.67 ± 0.24	84.92 ± 0.15
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 & Storkey, 2019)	Conv4	Omniglot	84.66 ± 0.70	88.41 ± 0.27	68.79 ± 1.03	74.05 ± 0.46
AAL-MAML (Antoniou & 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
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
SketchEmbedding(KL)-avg *(Ours)*	Conv4	Quickdraw	96.06	99.40	89.83	97.92
SketchEmbedding(KL)-best *(Ours)*	Conv4	Quickdraw	96.60 ± 0.18	99.46 ± 0.06	90.84 ± 0.15	98.09 ± 0.06
SketchEmbedding *(w/ Labels) (Ours)*	Conv4	Quickdraw	88.52 ± 0.34	96.73 ± 0.13	71.35 ± 0.24	88.16 ± 0.14
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

* Stroke data used for training

Table 10. Full table of 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
UMTRA (Khodadadeh et al., 2019)	Conv4	mini-ImageNet	39.93	50.73	61.11	67.15
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 & Storkey, 2019)	Conv4	mini-ImageNet	37.67 ± 0.39	40.29 ± 0.68	-	-
AAL-MAML (Antoniou & Storkey, 2019)	Conv4	mini-ImageNet	34.57 ± 0.74	49.18 ± 0.47	-	-
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	23.30 ± 0.21	26.22 ± 0.20	29.93 ± 0.21	32.57 ± 0.20
Conv-VAE	Conv4	Sketchy	23.27 ± 0.18	26.28 ± 0.19	30.41 ± 0.19	33.97 ± 0.19
Random CNN	ResNet12	N/A	28.59 ± 0.34	35.91 ± 0.34	41.31 ± 0.33	44.07 ± 0.31
Conv-VAE	ResNet12	mini-ImageNet	23.82 ± 0.23	28.16 ± 0.25	33.64 ± 0.27	37.81 ± 0.27
Conv-VAE	ResNet12	Sketchy	24.61 ± 0.23	28.85 ± 0.23	35.72 ± 0.27	40.44 ± 0.28
SketchEmbedding-avg *(ours)*	Conv4	Sketchy*	37.01	51.49	61.41	65.75
SketchEmbedding-best *(ours)*	Conv4	Sketchy*	38.61 ± 0.42	53.82 ± 0.41	63.34 ± 0.35	67.22 ± 0.32
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

* Stroke data used for training