Recognizing Hand-drawn Hydrocarbon Structures with Neural Networks: A Practical Case Study of Deep Learning and Synthetic Data Generation in a Chemistry Context

Supporting Information

Hayley Weir,[1,2] Keiran Thompson,[1,2] Amelia Woodward,[1] Benjamin Choi,[3] Augustin Braun,[1] and Todd J. Martínez[1,2,*]

[1]Department of Chemistry, Stanford University, Stanford, CA 94305

[2]SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA 94025

[3]Department of Electrical Engineering, Stanford University, Stanford, CA 94305

I Neural network and dataset details

The code for generating the synthetic datasets, the hand-drawn datasets, and the image-to-SMILES neural network can be found here: https://github.com/mtzgroup/ChemPixCH. Detailed output of the model for each of the test set molecules can be found in Test-set-predictions.zip, also available as ESI for this paper. This zipfile contains a separate image for each test set molecule, containing the handwritten image, the ensemble model predicted molecules (including the SMILES string and RDKit-rendered image), correctness of the prediction, number of votes for the prediction and validity of the predicted SMILES string.

Chemical structure recognition is a *supervised learning* problem: each input is associated with an output label. During the training process, a defined loss function, which depends on the error between the predicted NN label and the reference label, is minimized. Our image-to-SMILES network is an example of an encoder-decoder network: the input is encoded to create a compressed representation of the data, which is subsequently decoded to the predicted output. The key idea behind these encoder-decoder workflows is the ability to learn a mapping between two different representations of the same data by compressing it to its key "features" through the central bottleneck called the *latent space*. One of the properties of encoder-decoder frameworks that has made them so successful for supervised learning is the ability for the same network architecture to be used for many different applications by simply providing data specific to that application. For example, a CNN-LSTM network can be applied to chemical structure recognition, image caption generation,[1] mathematical formula recognition,[2] optical character recognition (OCR) in natural scenes[3] and hand-writing recognition[4] to name a few. Moreover, since the encoder and decoder networks are swappable components, they generalize well beyond the CNN-LSTM applications: machine translation can be achieved by simply swapping the CNN with another LSTM to form a sequence-to-sequence model for instance. Autoencoders are a special case of encoder-decoder networks in which the target output space is equal to the input space, used as a way of performing dimensionality reduction.

In our image-to-SMILES network, the LSTM decoder uses an *attention* mechanism to improve the accuracy of the output text sequence.[1, 5-6] The attention mechanism learns a probability mask over the image by calculating a "context vector" which acts as a dynamic pointer to relevant areas of the image during decoding. This reduces the loss of higher-level image features at the encoder bottleneck. For example, a high attention score for pixels showing two parallel lines in the chemical structure might prompt "=" to be output from the LSTM.

In addition to attention, beam search was also used in the decoding layers. During training, recurrent neural networks (RNNs) output predicted characters of the SMILES string and pass them back into the network, which outputs the next predicted character and so on. Applying beam search to an RNN allows the network to keep track of the strings with the k highest cumulative probability at each decoding step, while the other predictions are pruned. The final output of the NN is a list of k SMILES strings, with the highest ranked prediction being the first entry. A *greedy* decoder would have k = 1, meaning that only the highest probability characters are used at each step.

The CNN encoder architecture outlined in Table S1 was implemented. An LSTM with 512 units and embedding dimension of size 80 was used for decoding, with beam search (k = 5) and attention mechanism intermediary vector dimension of 512. We used the Adam optimizer[7] and a batch size of 20 for training. Network weights were saved based on the validation set's perplexity, p, calculated as

$$p = -\exp\left(\frac{H_{chars}}{n_{chars}}\right)$$

where H_{chars} is the sum of the cross-entropy loss for the characters in the validation set, and n_{chars} is the number of characters in the validation set. A learning rate of $1\text{x}10^{-4}$ was used for all training runs and the model was implemented in Tensorflow.[8] We define the NN *accuracy* as the proportion of molecules predicted exactly correctly, i.e., the predicted SMILES matches the target SMILES character-by-character.

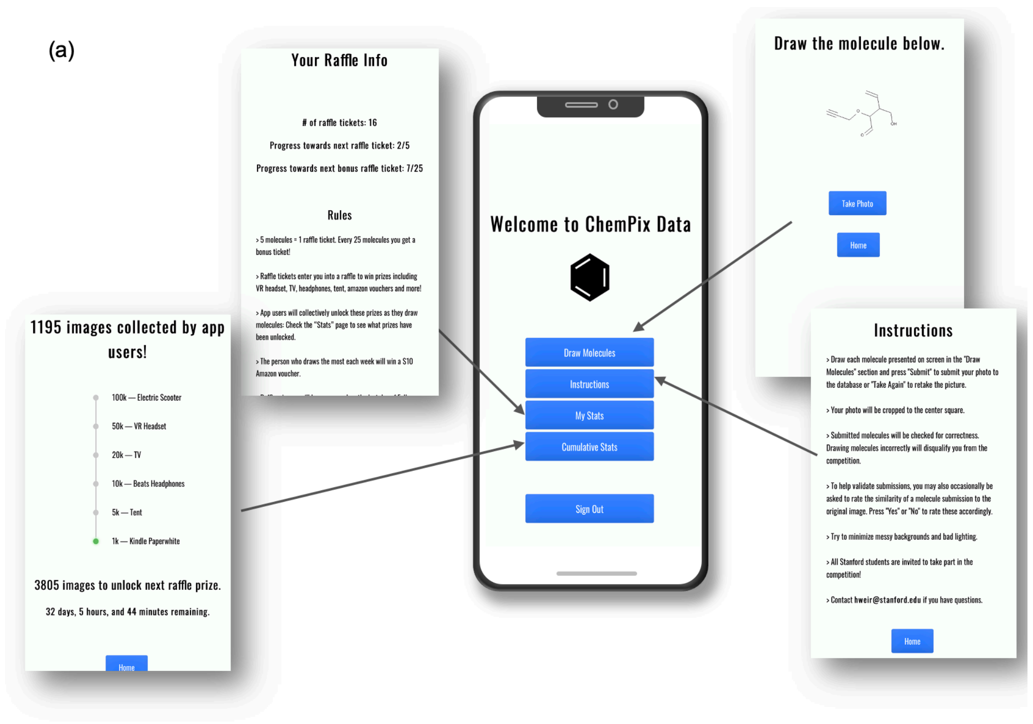
Table S1. Encoder architecture used in the image-to-SMILES neural network. "valid" refers to no padding, and "same" refers to even padding to the left/right or up/down to produce the same shape as the input.

Layer	#filters	Kernel size	Pool size	Strides	Padding	Activation
Conv2D	64	(3,3)	-	(1,1)	"same"	ReLU
Max Pool	-	-	(2,2)	(2,2)	"same"	-
Conv2D	128	(3,3)	-	(1,1)	"same"	ReLU
Max Pool	-	-	(2,2)	(2,2)	"same"	-
Conv2D	256	(3,3)	-	(1,1)	"same"	ReLU

Conv2D	256	(3,3)	-	(1,1)	“same”	ReLU
Max Pool	-	-	(2,1)	(2,1)	“same”	-
Conv2D	512	(3,3)	-	(1,1)	“same”	ReLU
Max Pool	-	-	(1,2)	(1,2)	“same”	-
Conv2D	512	(3,3)	-	(1,1)	“valid”	ReLU

II Data collection app

(a)



Your Raffle Info

# of raffle tickets: 16

Progress towards next raffle ticket: 2/5

Progress towards next bonus raffle ticket: 7/25

Rules

> 5 molecules = 1 raffle ticket. Every 25 molecules you get a bonus ticket!

> Raffle tickets enter you into a raffle to win prizes including VR headset, TV, headphones, tent, amazon vouchers and more!

> App users will collectively unlock these prizes as they draw molecules: Check the "Stats" page to see what prizes have been unlocked.

> The person who draws the most each week will win a $10 Amazon voucher.

Welcome to ChemPix Data

Draw Molecules

Instructions

My Stats

Cumulative Stats

Sign Out

Draw the molecule below.

Take Photo

Home

1195 images collected by app users!

100k — Electric Scooter

50k — VR Headset

20k — TV

10k — Beats Headphones

5k — Tent

1k — Kindle Paperwhite

3805 images to unlock next raffle prize.

32 days, 5 hours, and 44 minutes remaining.

Home

Instructions

> Draw each molecule presented on screen in the "Draw Molecules" section and press "Submit" to submit your photo to the database or "Take Again" to retake the picture.

> Your photo will be cropped to the center square.

> Submitted molecules will be checked for correctness. Drawing molecules incorrectly will disqualify you from the competition.

> To help validate submissions, you may also occasionally be asked to rate the similarity of a molecule submission to the original image. Press "Yes" or "No" to rate these accordingly.

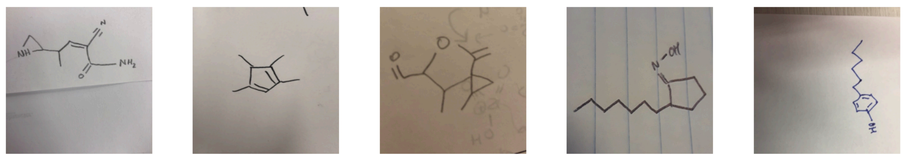
> Try to minimize messy backgrounds and bad lighting.

> All Stanford students are invited to take part in the competition!

> Contact **hweir@stanford.edu** if you have questions.

Home

(b)



(c)

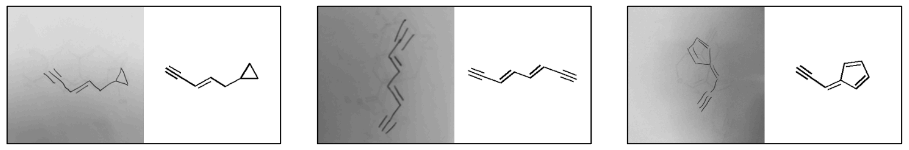


Figure S1. (a) Screen captures of the data collection web application. (b) Examples of photographs of hand-drawn chemical structures collected by the application. (c) Images collected from the app were checked against the RDKit structure to ensure that the data was accurate.

III Image processing

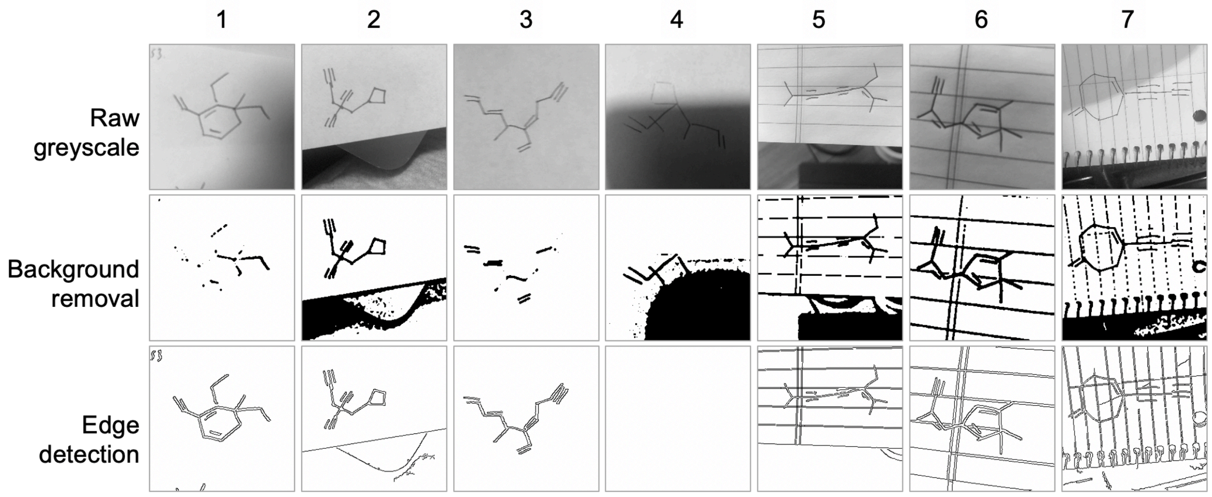


Figure S2. Demonstration of the brittleness of background removal (2nd row) and edge detection (3rd row) pre-processing algorithms on representative examples from the hand-drawn dataset (1st row). The algorithms break down when there are shadows, page features such as lines, or faint pen marks.

Table S2. Descriptions of image transformation functions used in augment molecule, augment background and degrade image pipelines. OpenCV and PIL python packages were used for all transformations.

Name	**Description**
Rotate	Rotate image randomly from 0 to 360° and fill blank space with white for (augment molecule) and reflected image (augment background)
Resize	Resize image randomly to (N,N) pixels where N is between 200 and 300.
Blur	Convolves image with kernel
Dilate	Dilate image
Erode	Erodes image
Aspect_ratio	Randomly adjusts the image aspect ratio by adding a border to top, bottom, left and right of image with width up to 50 pixels.
Affine	Applies as random affine transform with +/- 20 pixels from each corner
Flip_V	Flips the image vertically with a 50% probability
Flip_H	Flips the image vertically with a 50% probability
Distort	Applies a random distortion to the image using a set of nine different distortion options
Translate	Augment molecule: generates bounding box around molecule and translates randomly up to the edge of the image. Augment background: randomly translate image +/- 100 pixels and reflect image in the shifted portion.
Crop	Crop image on left, right, top and bottom up to 50 pixels each.
Border	Add a border on left, right, top and bottom of up to 40 pixels each. Reflect image to fill border 80% of the time, and 20% of the time use a constant value taken from the image
Salt+pepper	Set random pixels in image to 1 and 0 in an even ratio
Scale	Resize image up to half its size to 1.5x its size and resize back to (256,256)
Contrast	Randomly enhance (80%) or decrease (20%) the contrast
Quantize	Apply image quantization
Sharpness	Increase or decrease sharpness with equal probability.
Darken	Subtract value between 0 and 50 from image
Invert	Invert the image bitwise

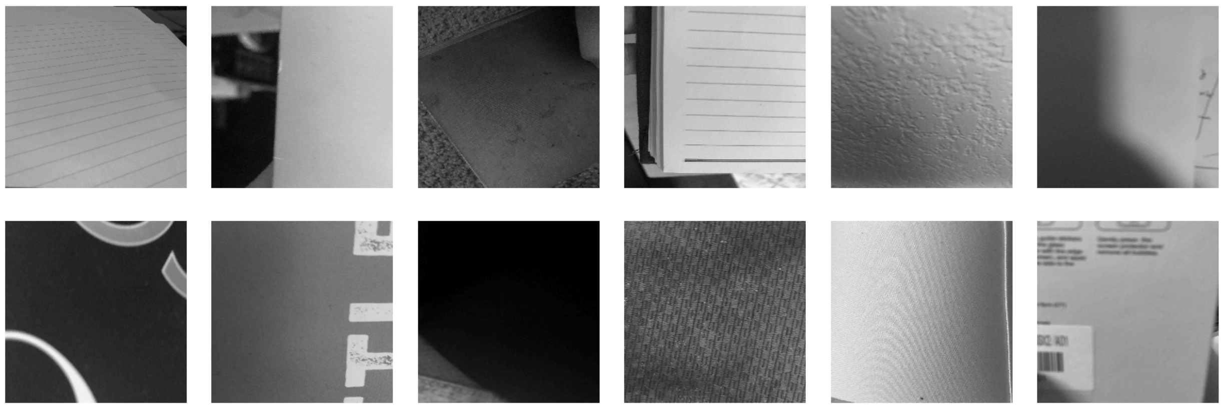


Figure S3. Representative examples taken from dataset of background images (greyscale). These images are added to augmented RDKit images during the synthetic data generation workflow.

The background images are randomly selected from a dataset of 1052 photographs (Figure S3). This backgrounds dataset was collected relatively easily as it did not require labelling. By adding the photographed backgrounds to a known molecule, a labelled synthetic dataset with realistic background textures and photograph features is produced. Since it is common for the act of labelling data to be the most time intensive step of dataset generation, sourcing a large dataset of an unlabelled component of the data and combining it with a synthetic labelled component can be an inexpensive way of generating synthetic data with realistic features.

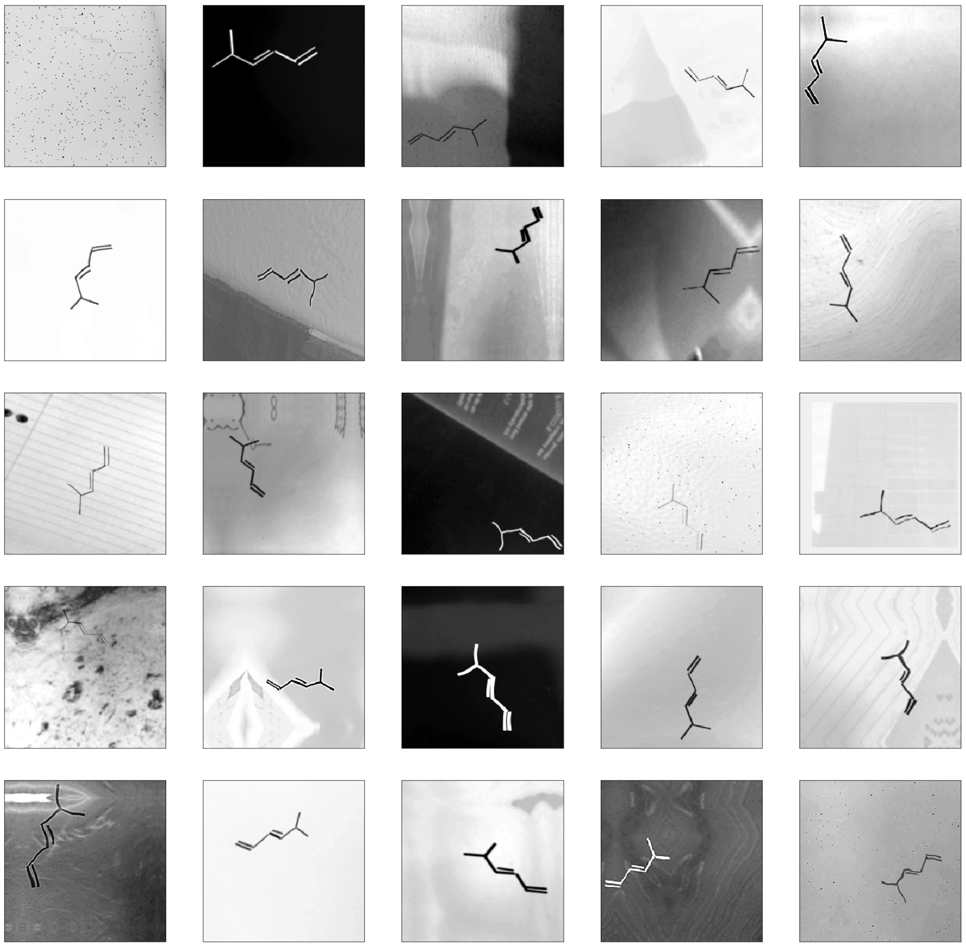


Figure S4. Example of the same molecule (SMILES: `CC(C)C=CC=C`) being passed through the synthetic data pipeline many times.

Representative examples of the synthetic and real hand-drawn datasets are compared in Figure S5. By eye, the synthetic images strongly resemble the hand-drawn data. However, since neural networks read the images as an array of pixel values, an important comparison metric is the frequency of the pixel values found in the images. We do this by comparing histograms of pixel intensity, which ranges from 0 (black) to white (255), for the synthetic and hand-drawn data. It can be seen that the synthetic data often has less of a smooth, continuous pixel count and less texture than the real-life data. Also, the frequency of pixel intensities is generally higher for the synthetic data in comparison to the hand-drawn data. These differences are due to the heavy augmentations of the backgrounds in the synthetic data pipeline (e.g., cropping and adding borders) which results in reduced image texture. This discrepancy could be reduced by increasing the size of the background dataset such that less aggressive augmentations would be required.

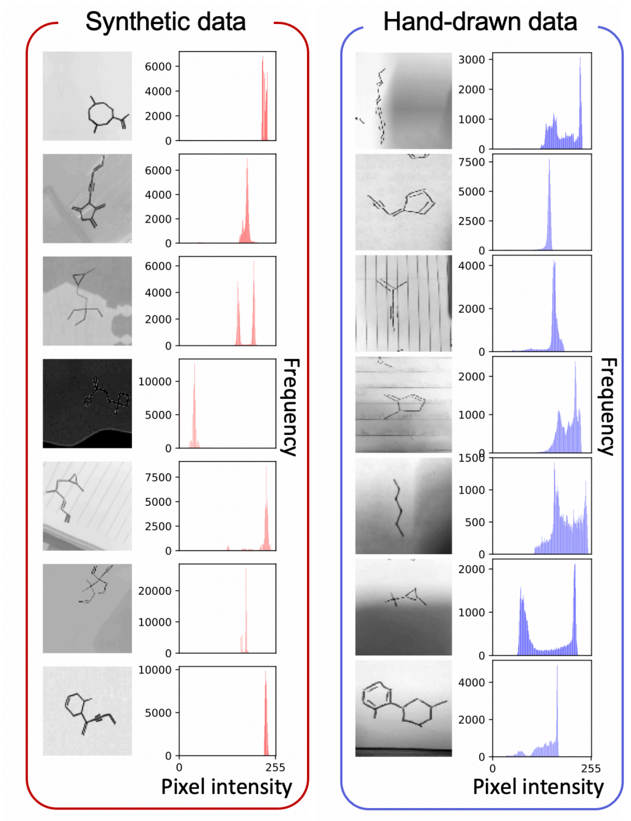


Figure S5. Comparison of representative images taken from synthetic dataset (left) and real-life hand-drawn dataset (right) and their pixel count histogram calculated by flattening the greyscale images and calculating the frequency of pixel intensities (values from 0 to 255 where 0 is black and 255 is white).

Analysis of the pixel counts in Figure S5 highlights the dangers of over-augmentation. A compromise must be reached between augmenting enough to prevent overfitting, but not so much that the data no longer resembles the target. An example of an overly-augmented background image can be found in Figure S6, which was generated with heavily cropped backgrounds. Since we have access to only a limited number of background images, we choose to augment our synthetic data relatively aggressively. However, we limit overly excessive cropping and resizing so not to remove completely the continuous texture of the image. Heavy augmentation can also lead to uninterpretable data, for example, molecules may be distorted such that bonds cannot be distinguished (Figure S6). This can confuse the training process and result in an increased error rate.

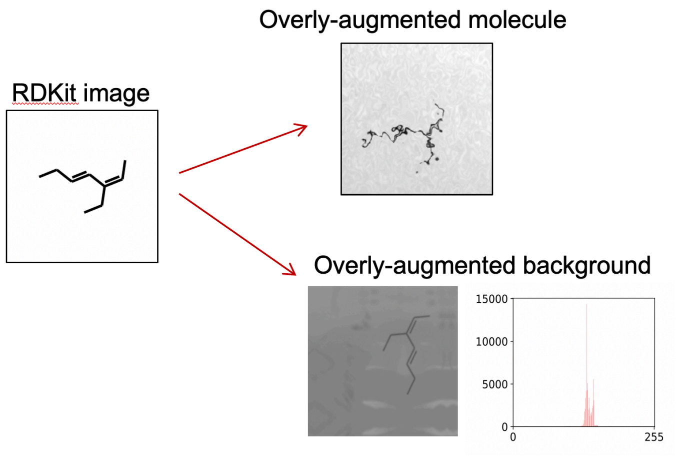


Figure S6. Demonstrating the effects of over-augmentation on data. Top: Overly-augmented molecule *via* heavy distortion leads to an indiscernible molecule. Bottom: Overly-augmented background image *via* excessive cropping shows the lack of structure and high value pixel intensities of the image.

IV Neural network training

In order to examine how each stage of the synthetic data generation workflow (Figure 5a) affects training, we train our CNN-LSTM network with data from each stage of the pipeline. Datasets of 200,000 images from each of the four steps in the workflow were split into train, validation and test sets as detailed in the Methods section. The results of the training are presented in Figure S7a. As the steps proceed through the synthetic data generation pipeline, the non-uniformity of the data increases, making it more complex, and hence more challenging for the NN to learn. As a result, slower optimization and a reduction in final accuracy is observed (Figure S9). The image-to-SMILES network is tested on the same datatype used for training (e.g. if the network was trained with RDKit'-aug data, it would also be tested on RDKit'-aug data) as well as our real-life hand-drawn dataset. The test set accuracy of the hand-drawn data is seen to increase from ~8% to ~47% as we proceed through the steps in the data generation pipeline, illustrating that each stage performs the desired effect of bringing the computer-generated data and the hand-drawn data distributions closer together.

From Figure S7a, it can be seen that augmenting the images increases the hand-drawn hydrocarbon recognition accuracy by ~20 percentage points. Moreover, degrading the images increases the accuracy from ~30% to nearly 50%. The change in accuracy when adding in backgrounds is insignificant in comparison to the addition of augmentation and degradation. This is a surprising result, since observation by human eye would suggest the opposite.

Next, we investigate how the size of our synthetic dataset (RDKit'-aug-bkg-deg) impacts the training and test set accuracies. The network was trained with datasets of size 50,000, 100,000, 200,000 and 500,000 images (split between training, validation and test sets according to the Methods section). As the number of images in the synthetic dataset increases, the out-of-sample recognition accuracy on the synthetic data grows from 0% to nearly 90% (Figure 7b). It can be seen that the difference between the accuracy of the synthetic and hand-drawn test sets increases with dataset size, demonstrating how the network begins to overfit to the synthetic data. Remarkably, the NN trained with 500,000 images achieves an accuracy of over 50% on real-world hand-drawn data, despite not having been exposed to hand-drawn data at any point during the learning process. This result suggests that the auxiliary data bears significant resemblance to the target datatype, and hence assigns some confidence that the workflow developed in the previous section behaves as desired. For reference, training with 500,000 raw RDKit images results in a real-life hand-drawn hydrocarbon recognition accuracy of 0%.

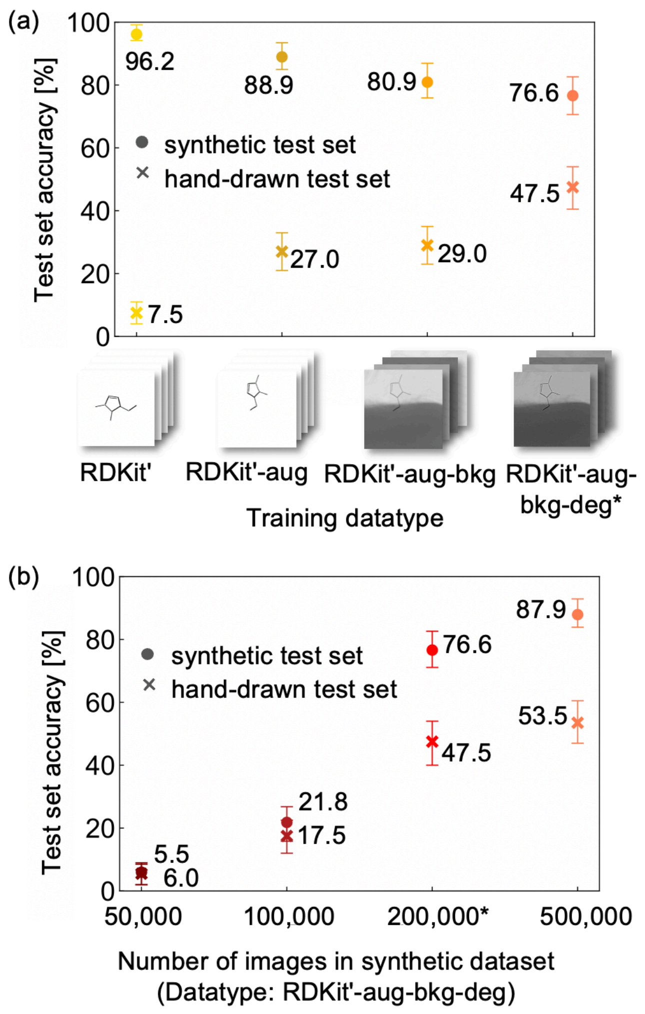


Figure S7. Training experiments with synthetic data showing the test set accuracy of the datatype used for training (x) and hand-drawn data (o). (a) Results of training with 200,000 images from each stage of the synthetic data generation pipeline: modified RDKit images (RDKit'), augmented RDKit images (RDKit'-aug), augmented RDKit images with background addition (RDKit'-aug-bkg), and augmented RDKit images with background addition and degradation (RDKit'-aug-bkg-deg). (b) Results of training with different sized training sets of the final synthetic data (RDKit'-aug-bkg-deg). Equivalent training runs in the two sets of experiments are indicated (*).

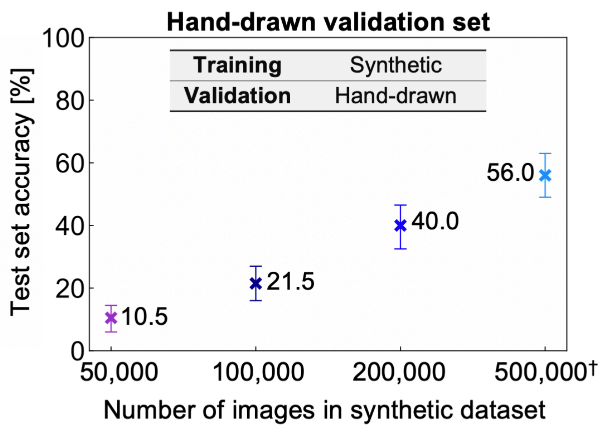


Figure S8. Recognition accuracy of the hand-drawn hydrocarbon test set of neural network trained with increasing sized synthetic training sets and hand-drawn hydrocarbon validation set.

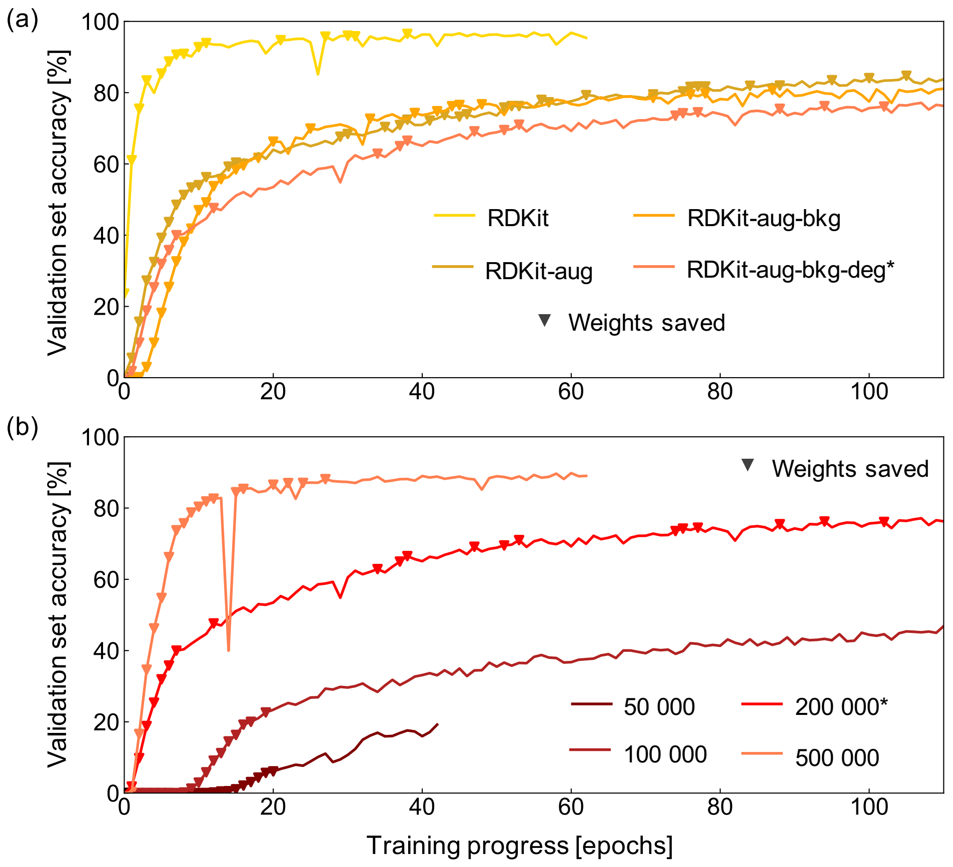


Figure S9. The accuracy of the validation set (measured as percentage of exactly matching predicted and reference SMILES) during the neural network training (measured in number of epochs) for (a) stages of the synthetic data generation pipeline and (b) different sized training sets of the synthetic data (RDKit'-aug-bkg-deg). The weights are saved according to maximum perplexity, indicated with triangle markers.

Since the model is constructed based on only the data used for training, if the training data more closely matches the desired testing data the model will perform better. After each epoch, the NN weights are tested on the validation set to track the model's accuracy as the training proceeds. Weights that achieve the best results are saved according to the network's perplexity score, a measure of the uncertainty of the prediction. Although the validation set is not directly used for optimizing the weights, it can be thought of as a "target" that the NN is aiming for and therefore should be equivalent to the desired use case. As a result, we expect that adding hand-drawn structures to the training and validation sets will increase the hand-drawn molecule recognition accuracy.

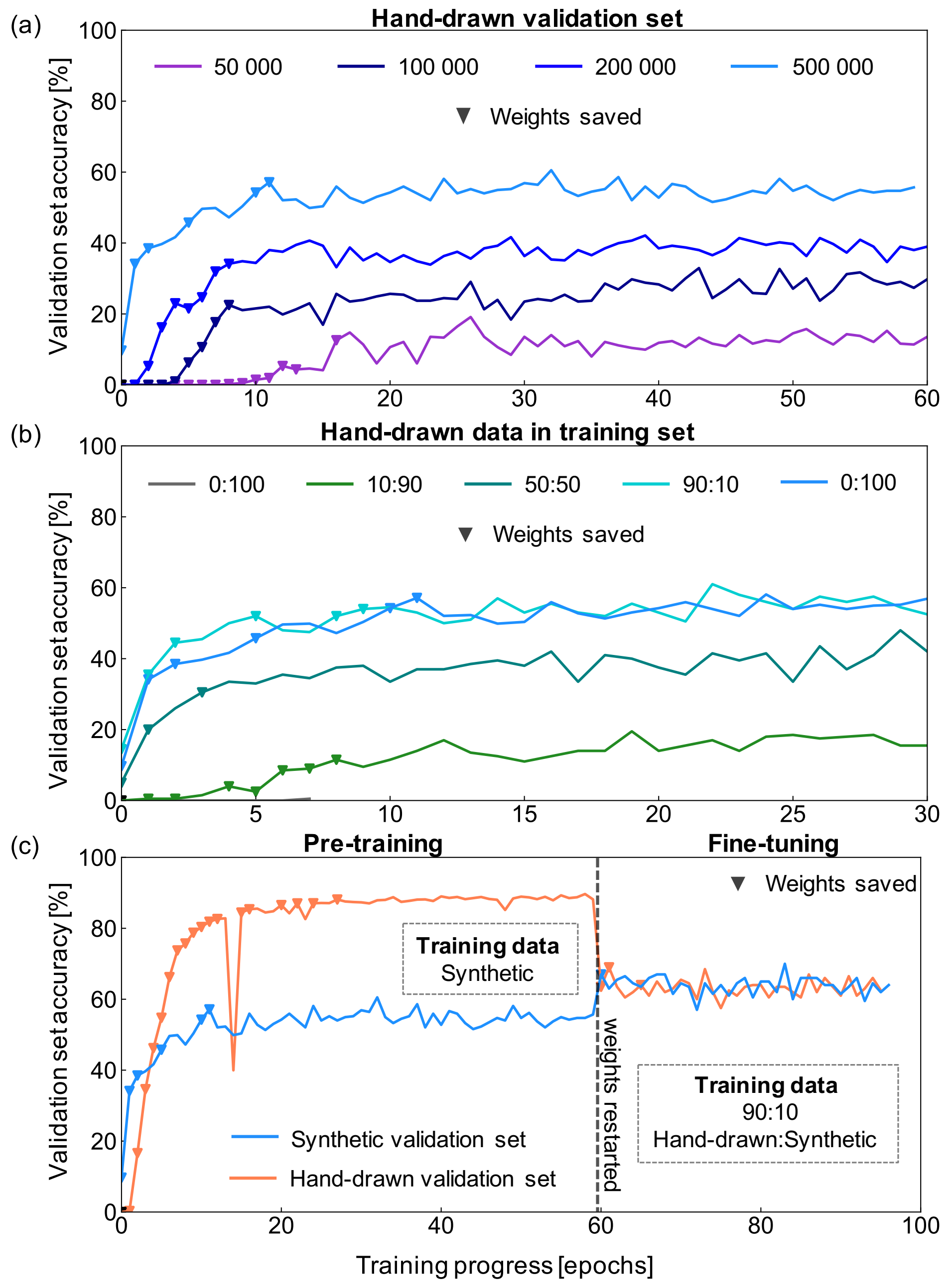


Figure S10. The accuracy of the validation set (measured as percentage of exactly matching predicted and reference SMILES) during the neural network training (measured in number of epochs). (a) Varying sized synthetic data training sets and hand-drawn validation set. (b) Varying ratios of synthetic data to augmented hand-drawn data training set and validation set of hand-drawn hydrocarbons. (c) Pre-training with synthetic data and a synthetic dataset (orange) or hand-drawn validation set (blue) before fine-tuning with 90:10 synthetic:augmented hand-drawn training dataset and hand-drawn validation set. The weights are saved according to maximum perplexity, indicated with triangle markers and point at which weights are restarted for fine-tuning is indicated (black dotted line).

References

1. Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; Bengio, Y. In *Show, attend and tell: Neural image caption generation with visual attention*, International conference on machine learning, 2015; pp 2048-2057.
2. Deng, Y.; Kanervisto, A.; Ling, J.; Rush, A. M. In *Image-to-markup generation with coarse-to-fine attention*, International Conference on Machine Learning, PMLR: 2017; pp 980-989.
3. Shi, B.; Bai, X.; Yao, C., An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Trans. Patt. Anal. Mach. Learn.* **2016,** *39* (11), 2298-2304.
4. Ingle, R. R.; Fujii, Y.; Deselaers, T.; Baccash, J.; Popat, A. C. In *A scalable handwritten text recognition system*, 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE: 2019; pp 17-24.
5. Bahdanau, D.; Cho, K.; Bengio, Y., Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* **2014**.
6. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. In *Attention is all you need*, Advances in neural information processing systems, 2017; pp 5998-6008.
7. Kingma, D. P.; Ba, J., Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* **2014**.
8. Abadi, M. i.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; Zheng, X. In *TensorFlow: A System for Large-Scale Machine Learning*, Savannah, GA, 2016/11//; {USENIX} Association: Savannah, GA, 2016; pp 265-283.