StrokeCoder: Path-Based Image Generation from Single Examples using Transformers

Sabine Wieluch and Friedhelm Schwenker
Institute for Neural Information Processing
Ulm University
89081 Ulm
sabine.wieluch@uni-ulm.de, friedhelm.schwenker@uni-ulm.de

Abstract

This paper demonstrates how a Transformer neural network can be used to learn a generative model from a single path-based example image. We further show how a data set can be generated from the example image and how the model can be used to generate a large set of deviated images, which still represent the original image's style and concept.

Introduction

Hand-drawn sketches are often used to quickly illustrate a scene or easily capture a thought. So a quick drawing often becomes the first step of a large design process and is an essential part in generating new design ideas (Suwa, Gero, and Purcell 1998).

To support users in their design processes (Do 2005) or to give users the possibility for creative experimentation with sketches, for example with with Casual Creators (Compton 2019), it will be very interesting to build such supportive tools with the help of generative machine learning.

Though for most design process it is not interesting or even not possible to collect a large training data set. For example it is not suitable for a computer game level designer to create thousands of levels to train a generative model, as the data set generation exceeds the generative model's value. For this reason, this work focuses on three main points. First: one-shot learning, where a model is trained by only one or few examples. Second: generative models that produce path-based images, not pixel-based images. For supportive drawing applications and also for other design tasks like in digital fabrication (Wang et al. 2008), it is important to work with path data (for example a laser cutter needs path data to move along a line). Third: generation processes that can derive new work from the training data but preserve the original style.

Interesting work on sketch image generation has been performed by (Singh et al. 2017), who also focus on the generation of style preserving derivatives for Co-Creative settings. Also other research groups (Yu et al. 2020; Eitz, Hays, and Alexa 2012; Yu et al. 2016; Gasques et al. 2019), worked on sketch data mainly utilizing Generative Adversarial Networks (Radford, Metz, and Chintala 2015; Goodfellow 2016). Therefore the resulting images are pixel-based and not path-based images.

Machine learning on path-based data has been performed in related research work. One very well known example is Sketch-RNN (Ha and Eck 2017) where a recursive neural net was trained with a large data set of small sketches depicting different objects. Another very interesting approach by (Balasubramanian, Balasubramanian, and others 2019) proposes a GAN-like architecture utilizing Long Short-Term Memory (LSTM) neural nets to generate path-based images from large sketch datasets.

Recent studies by Xu et al. showed that Transformer neural networks are well suited to be trained on sketch data. Transformers (Vaswani et al. 2017) are state-of-the art architectures for handling sequential data and outperform RNN or LSTM architectures in Natural Language Generation tasks. Xu et al. implemented a graph-based representation for strokes to perform free-hand sketch recognition (Xu, Joshi, and Bresson 2019; Xu et al. 2020).

Training and generation of Scalable Vector Graphics has been performed by (Lopes et al. 2019), where they used several fonts as data sets.

In our last research (Wieluch and Schwenker 2019), we examined how Dropout can be used with Generative Adversarial Networks to create different, but coherent images in image-to-image translation tasks. For this research paper we aim to learn a generative model from one single hand-drawn image. We focus on generating diverse images that match the input image's style and concept. The generated images will be especially useful for creative support tools, art or digital fabrication.

Data Structure and Transformer Architecture

In this research, we aim to learn a neural representation for path-based sketch drawings. A sketch drawing consists of a sequence of pen strokes and each stroke can be approximated by small straight line segments. This sequence of sequences can be easily flattened to one large sequence.

Therefore learning a neural representation for sketch drawings can be formulated as a sequence generation task. Transformers (Vaswani et al. 2017) are a new class of neural nets which have been especially useful to the Natural Language Processing community. They are used for sequence-to-sequence translation tasks (Devlin et al. 2018) as well as for text generation (Radford et al. 2019). Also other domains have used Transformers for various tasks like music generation (Huang et al. 2018).

A Transformer (Vaswani et al. 2017) consists of an Encoder and a Decoder, as it is usually used for sequence-to-sequence translation tasks. However, in our setting we aim to learn a generative model and therefore we will only use a Transformer Encoder.

This seems counterintuitive at first, because for a generation task usually the Decoder would be used. Though Transformers Encoder and Decoder are very similarly constructed and mainly differ in an additional input from the Encoder to the Decoder in a translation setting. As we only want to generate a sequence, we can discard this input and end up with with the Transformer Encoder.

The Transformer Encoder consists of multiple layers, which end in a linear and softmax layer. Decoder layers can be stacked on top of each other any number of times.

One Decoder layer consists of three sub-layers. The first and second layers are Multi-Head-Attention layers, which are a number of parallel attention layers, whose output is concatenated and finalized with a linear layer. A self-attention layer gives the neural net the ability to focus more on certain moves or ignore other moves in the sequence. A mask is applied to the first attention layer to prevent the neural net from seeing future sequence elements. Each of these two sub-layers end with a layer normalization. The third sub-layer is a feed-forward network, also ending with a layer normalization. Figure 1 gives a visual overview of our used architecture.

The Transformer architecture does not receive information of the positional order of the sequence elements. Therefore, additional positional encoding is required. In our research, we use the standard positional encoding defined in (Vaswani et al. 2017).

But before we can use the Encoder, the input needs to be prepared accordingly. The sequence of straight lines needs to be converted to a sequence of vectors that can be processed by a neural network. In Natural Language Processing, using word sequences as input is a very similar problem. Here, word embeddings (Levy and Goldberg 2014) are used to encode words into vectors that can be used as input data. Following this method, we also embed our lines, but first we need to define what such a line actually consists of:

- Pen State (indicator if line should be drawn or not)
- Position (position to move to; relative to last position)

This representation is very similar to Turtle Graphics, where a virtual pen can be moved with relative position commands. In the following, we refer to this line definition as a “move”. Additionally, we introduce a special move to indicate the end of the image, which is also added to the embedding.

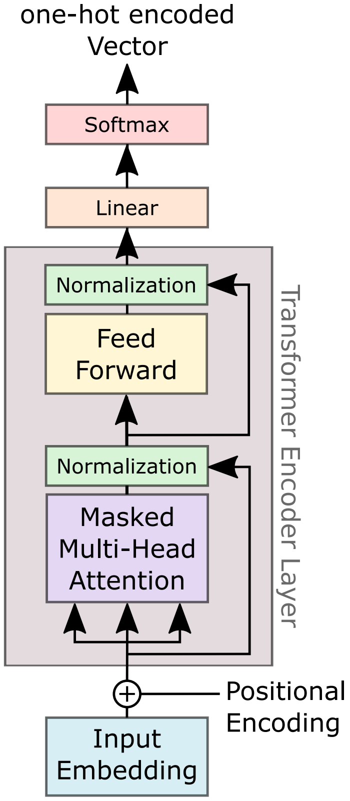

Figure 1: Transformer Encoder let as used in this work: after the input is embedded and receives positional encoding information, it uses multiple self-attention layers and a feed-forward net to process the sequence. Encoder layers can be stacked on top at any number. The Encoder layer output is processed through a linear and a softmax layer to receive the final one-hot encoded vector, which can then be used to read the final move from the embedding.

The stroke ending does not need an own indicator move, as it is encoded in the Pen State change.

The embedding works like a trainable lookup table: each pen move is assigned a real number vector of a certain length. This vector can be retrieved by the move’s index in the lookup table. The embedding vectors are initialized randomly and are trained with gradient decent. By training the embedding vectors, similar moves (in the context of the drawing) receive similar vectors. This similarity is easier to observe in language: for example the words “rose” and “flower” are often used in similar contexts and should therefore receive a more similar vector than the words “house” and “spaghetti”.

For a greedy sampling, the Transformer output is a one-hot encoded vector with the length of the embedding size. It contains one 1 at the index of the predicted next move.

Data Set Generation

In our experiments, the initial stroke drawings are hand-drawn with a digitizer pen, from which only points are recorded (see figure 2). The point sequence is then simplified to remove unnecessary points and instead describe the hand-drawn stroke by few curves instead of many points. So, a sequence of as few curves as possible is fitted through the recorded pen position points with an allowed maximum error (Schneider 1990). The resulting sequence of curves will in the following be referred to as a path. The recorded drawing is stored in this path state, because it is a better approximation as the later constructed straight moves, especially if the path is altered to create a large data set as described below.

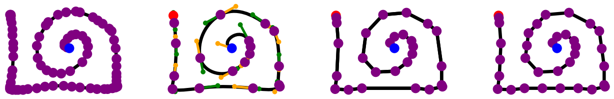

Figure 2: Simplification of a path: the first image depicts a hand-drawn stroke, where on each mouse event a new point is recorded. The next image shows a simplified version (a path), where multiple points have been substituted with a curve. In the next step, the path is converted to moves where too long moves are divided into multiple shorter moves.

When learning generative models from few natural images (Shaham, Dekel, and Michaeli 2019), the images are altered into a variety of so called patches. These patches are cut out parts of the original images, which are also often slightly deformed, scaled or changed in other manners to produce a larger amount of training data as the initial images would have provided.

To learn a generative model on one single sketch image, we propose a similar method: the initial strokes are altered in different ways to produce a large and diverse training data set. As stroke-based images differ a lot from natural, pixel-based images, the altering methods need to be adapted accordingly. All proposed altering methods are visualized in figure 3 and will be described below:

- **Translation:** The whole path-image is moved to a new position in a way, that it is still contained in the initial image boundaries.
- **Rotation:** The whole path-image is rotated by a random angle.
- **Mirror:** the path image is mirrored along an axis.
- **Path Reversal:** As each path consists of a list of curves with start and end points, a path has an implicit direction. In our setting, the path direction is not important, so a path can be reversed to generate new patches. The path direction might be important for other settings like sketch or handwriting classification (Xu, Joshi, and Bresson 2019), where the stroke direction and path order are very similar in one letter. As Path Reversal is a binary state (either the path is reversed or not), this manipulation is applied with a probability of 0.5.
- **Scaling:** The whole path-based image is scaled to a smaller size.

After all paths have been manipulated, they are rearranged in a new order. We sort them in a greedy way by distance, so that the pen travel is as short as possible. We start at a randomly chosen path in the image. With this new path order, we add more variety in the data set while assuring that the model will learn to draw close by the last stroke. If paths would be not sorted but shuffled randomly, the resulting images will look more scattered, as new paths would appear in greater distances.

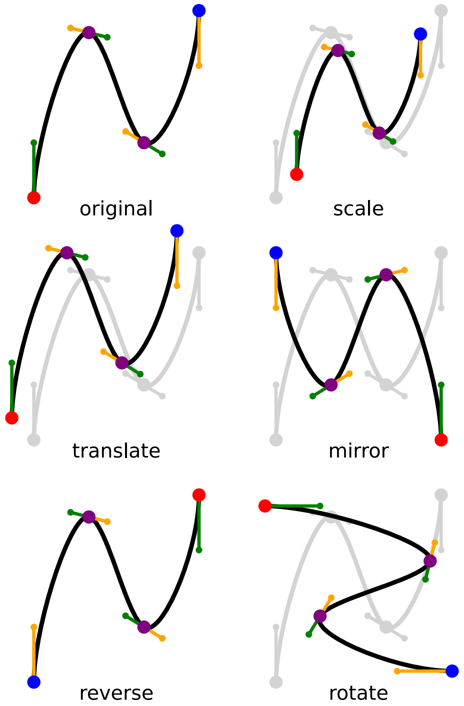

Figure 3: Five types of path manipulations used to create a large data set from one example. Translation, Rotation, Scaling and Mirroring are used on the whole stroke-based image, whereas Path Reversal is used on individual paths in the image.

Training

We recorded 4 different drawings with an image boundary of 180x180 units to serve as initial stroke-based images, which can be seen in figure 8.

Each model was trained for 200 epochs, where for each epoch a new data set of 500 patches is generated from the original image. These patches were then converted to moves with a maximum line length of 15 units.

It is important to generate a new patch data set each epoch to prevent overfitting to a small patch set. Instead, with the changing patch training data, the model sees a larger portion of the possible patchs.

To verify that the model learns to represent the whole patch distribution, we calculated the Cross Entropy Loss of an unseen data set of 500 patches after each training episode. The results can be seen in figure 9. After an initial increase, the loss sinks for each new training epoch. Because the model sees a large portion of the data set distribution, it

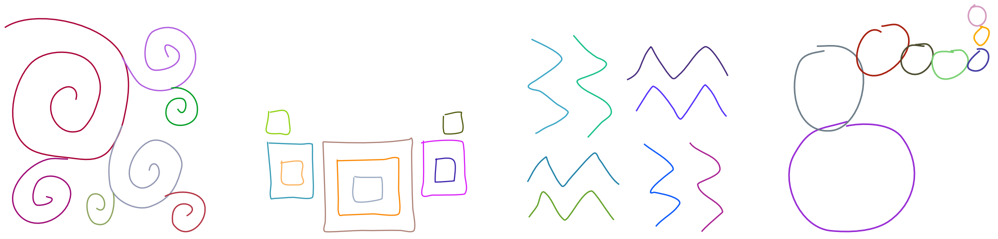

Figure 4: “curles” Figure 5: “boxes” Figure 6: “spikes” Figure 7: “cirlces”

Figure 8: Four recorded stroke-based images to be used in the experiments. Each stroke is colored randomly for better distinction.

better learns to represent the whole data set distribution.
If the model would only be trained on one patch data set, the model quickly begins to overfit as can be seen in figure 10. Here the “boxes” model was trained with one single patch data set of size 100, 500 and 1000. The plot depicts the Cross Entropy Loss of these models between an unseen patch set of size 500. The larger the training patch set is, the lower the loss. Though, the loss increases for each episode as one small patch set badly represents the whole data set distribution and the model overfits to the small sample.

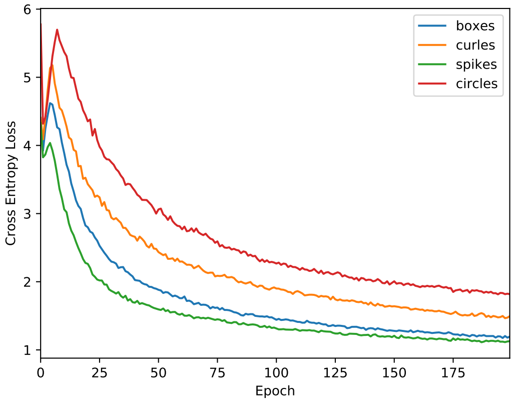

Figure 9: Loss of one unseen patch set of size 500. Loss is calculated with every model after each epoch. Models were trained with newly generated patch sets for every epoch. The falling loss curve indicates that the model learns to represent the whole patch distribution. (Absolute Loss differs because of different sequence lengths between models).

When generating moves, the approximation error for the curve flattening algorithm should not be set too high, as the resulting image quality suffers. The difference of an error between 1 and 3 can be seen in figure 11. With an large allowed error, especially details in narrow curves are lost. For our research we chose a maximum error value of 1.

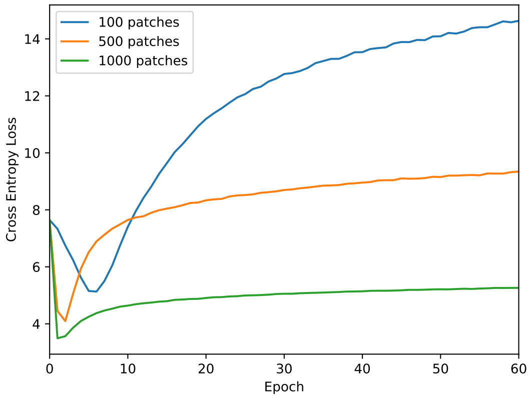

Figure 10: Loss of one unseen patch set of size 500. Loss is calculated with three versions of the “boxes” model: one which was trained with a patch set size of 100, 500 and 1000. The rising loss curve indicates that the model overfits to the small patch sample.

In the training phase, we do not feed patch by patch to the Transformer. Instead we feed in a continuous stream of patches (though the input vectors are shuffled). So one input vector can contain for example the end half of a patch and the beginning halt of another patch. This way the Transformer learns to generate a stream of images. This will be helpful in the generation phase.
For our training, we use the Adam optimizer (Kingma and Ba 2014) with the described changes and parameters in (Vaswani et al. 2017), where the learning rate is first linearly increased for the first warm-up steps and thereafter decreased again.
As a loss function, we use Cross Entropy Loss.
For all of our experiments, we used the following Transformer settings:

- Batch Size: 200
- Sequence Length: max. move length of recorded image
- Hidden Embedding Size: 52
- Decoder Layers: 6
- Attention Heads: 4
- Feed Forward Size: 2048

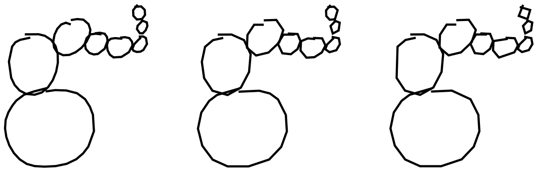

Figure 11: Curve flattening results with maximum allowed error of 1(left) to 3(right). With an higher allowed error, smaller details like narrow curves suffer and are approximated to sharp edges.

Inference and Sampling

The Transformer can only predict one new move dependent on the previous moves. So the Transformer needs at least one move to start generating. We experimented with different lengths of random input vectors as initialization. These random vectors always ended with the "image-end" move to trigger the generation of a new image. The random initialization vector is of course discarded and not shown in the drawings. Figure 12 shows the results of three different initialization vector lengths. The black group was generated only by passing the "image-end" move to the Transformer. The images are often rather short (in sense of stroke count) and rarely show strokes that don't seem to fit to the original image. Also some empty images occurred.

The red block was generated with a random initialization vector of half the sequence length. The images show a high variety and seem to fit well to the original image.

The blue group on the bottom of figure 12 was generated with an initialization vector of the full sequence length. Here, the model seems "confused" by the random input and produces a large variety of long or unfitting strokes.

We suggest to use an initialization vector in the range of half the sequence length to create enough randomness for interesting results but not to confuse the model too much. Though other initialization vectors might be better suited certain situations. For example the blue result group in figure 12 might be interesting to use in creativity support tasks where an AI helps the drawing user to get new inspiration. Here it might be beneficial that the model results differ more from the original concept.

As a sampling strategy we use top-k sampling. Here, only the best k predictions are considered and are then chosen according to their probability. If $k = 1$ top-k sampling is equivalent to a greedy approach where every time only the best prediction is chosen. Figure 13 shows results from all trained models with two different k values: the left side is a greedy sampling with $k = 1$ where the right side shows a sampling with $k = 10$.

The mayor difference in both sets is the image variety. With the greedy approach, the model often falls back to a small set of images. But these images fit well to the original image style. Though when generating, we experienced that the Transformer got stuck in generative loops, where it would draw one shape indefinitely.

The images sampled with top-10 show a larger variety and

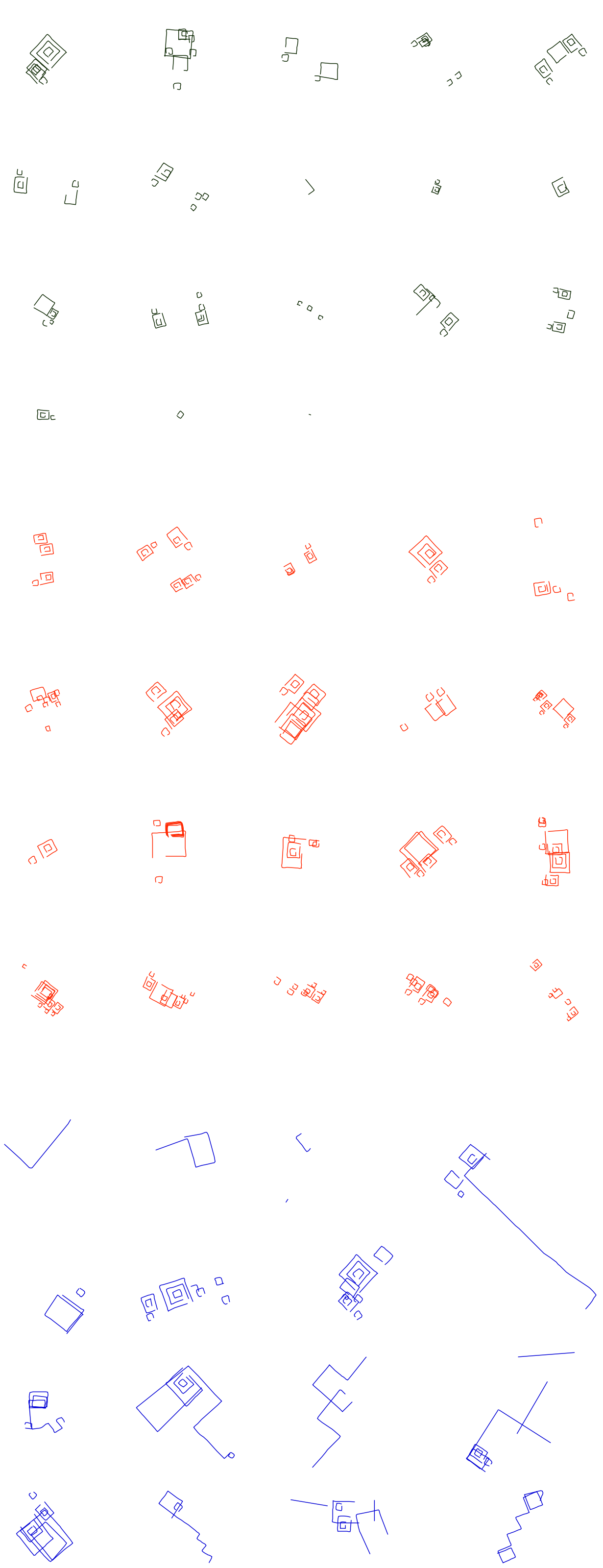

Figure 12: Group of sampled sketches, differing in the inizialization vector length: 1 (black), half of sequence length (red) and full sequence length (blue).

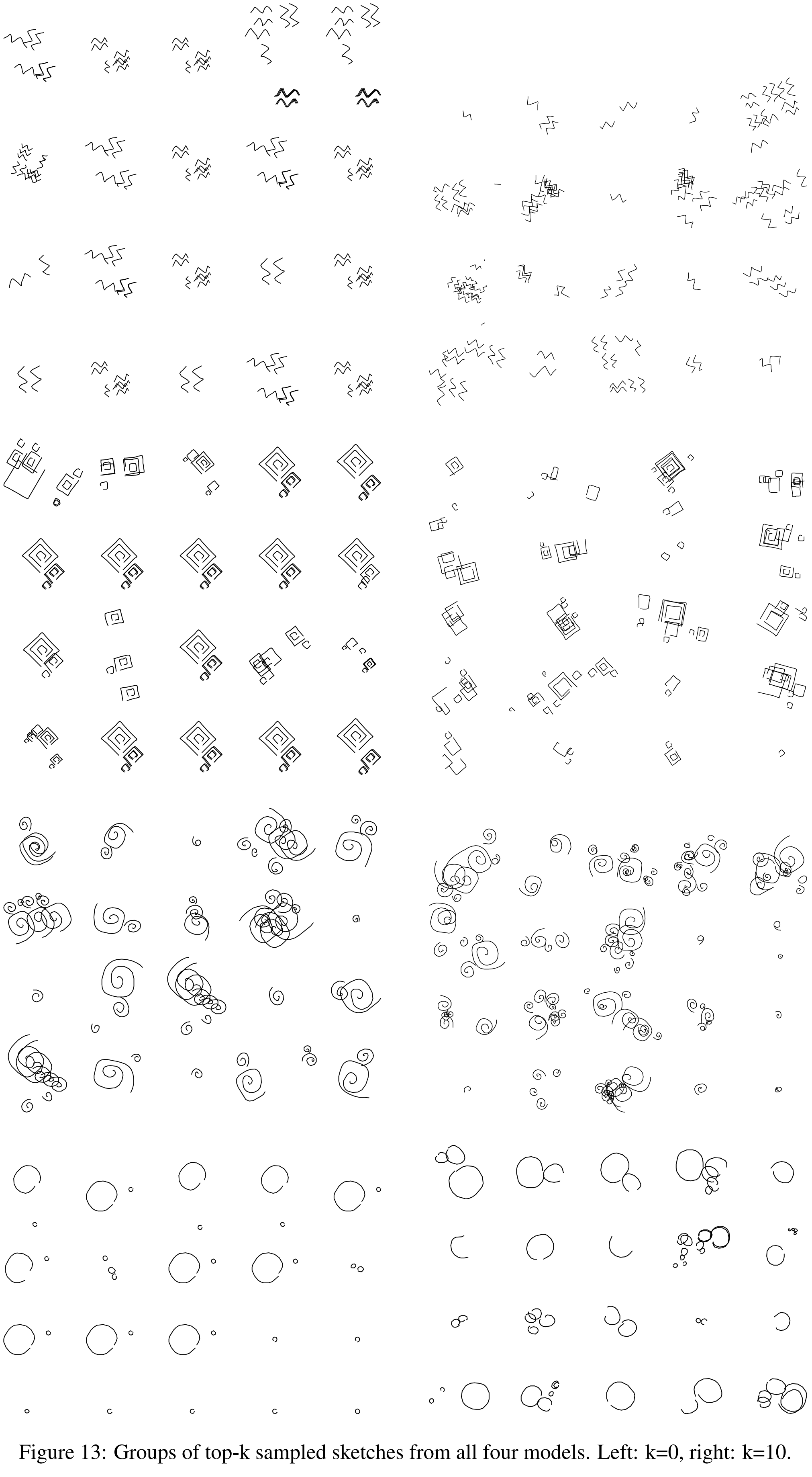

Figure 13: Groups of top-k sampled sketches from all four models. Left: k=0, right: k=10.

also fit well to the original image, except some individual strokes. Over all these generated images look a bit more chaotic and also show new shapes that are not part of the original image, but fit the style (f.e. double-curled lines in the “curles” image).
In Natural Language Processing, the sampling technique is often adjusted to the generation task. So in case of generating stroke images, it might also be best to adjust the sampling algorithm to the intended purpose.

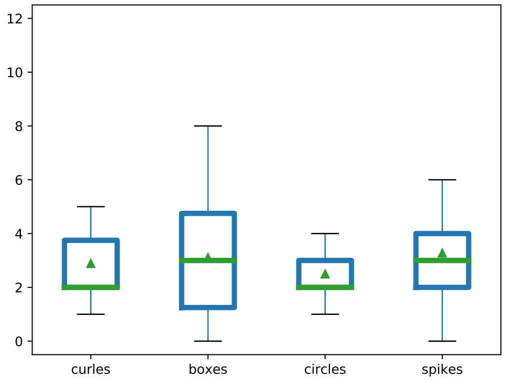

Figure 14: 18 participants chose out of a set of 12 pattern describing adjectives, which fit best for the original image and a set of generated images. Plot shows the error distribution grouped by model. The average error is 2.94.

To evaluate if the generated images preserve the style of the original sketch, we asked 18 participants to choose from a set of 12 adjectives (parallel, ordered, stacked, chaotic, curly, straight, symmetric, repetitive/rhythmic, tangled, circular, jagged, nested) which ones best describe the original image and a set of generated images. The adjectives were chosen in a way that in combination they are able to describe a large variety of pattern. The question order was counterbalanced to avoid bias. We calculated the difference in chosen adjectives for each person. Figure 14 shows the results for each model: the mean error over all models is 2.94 from a maximum possible error of 12. From the results we can see, that some models better preserve the style than others. The “boxes” and “spikes” models most likely perform worse in this test, because the trained models seem to be good in replicating drawing shapes, but not in positioning them. And these two model’s main features lie in the stroke positions (for example the “boxes” input image is symmetric).
Depending on the model, an average of two to three attributes changed between the assessment of the original image and the generated images. So the majority of attributes is shared between both images, which allows the conclusion that the generated model does not perfectly preserve the style but learned to replicate the large majority of attributes.

Conclusion

In our work, we showed that a Transformer neural net can be used to learn and generate stroke-based images from one single input image. We generated large training data sets from one image, using different path altering methods. Four hand-drawn image samples were recorded and used to generate our training data sets. We proved that is is essential to train the Transformer with a changing subset of training data in each epoch. This training data rotation prevents the model from overfitting to a too small sample.

We compared different initialization vector lengths and found that a lengths of half the Transformer sequence length gives the best results concerning variety and similarity to the original image. We also compared different sampling parameters in top-k sampling. We could observe that a greedy sampling method like $k = 1$ results in a very low variety in images which fit very well to the original image. Increasing k also increases variety but also introduces strokes that do not fit well to the original image’s style. Therefore it is important to choose k depending on the application needs.
Finally we evaluated the style preservation capabilities of our models in an assessment of 18 participants. The results showed that most of the style attributes are learned by the model, though style fitting stroke placement was not always achieved.

In our future research we want to expand these generative methods to create larger path-based pattern images from one input image. For this it might be interesting to use hierarchical approaches, as they have been successfully used in other domains like dialogue generation (Serban et al. 2017). An hierarchical approach might be helpful to give the neural net an overview of already generated paths and their places beyond the sequence length memory. We see future applications for these large generated pattern or also expanded pattern in games, art, digital fabrication and many more creative domains.
Another interesting field is Co-Creative Design (Guzdial, Liao, and Riedl 2018), where a user is cooperating with a supportive artificial agent in a design task. Here, a stroke-based image representation can be very useful, especially in the domain of Casual Creators, because users can very intuitively interact by drawing with a pen.
It will be interesting to explore these applications and domains in future research.

References

Balasubramanian, S.; Balasubramanian, V. N.; et al. 2019. Teaching gans to sketch in vector format. *arXiv preprint arXiv:1904.03620*.

Compton, K. 2019. *Casual creators: Defining a genre of autotelic creativity support systems*. University of California, Santa Cruz.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Do, E. Y.-L. 2005. Design sketches and sketch design tools. *Knowledge-Based Systems* 18(8):383–405.

Eitz, M.; Hays, J.; and Alexa, M. 2012. How do humans sketch objects? *ACM Transactions on graphics (TOG)* 31(4):1–10.

Gasques, D.; Johnson, J. G.; Sharkey, T.; and Weibel, N. 2019. What you sketch is what you get: Quick and easy augmented reality prototyping with pintar. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–6.

Goodfellow, I. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Guzdial, M.; Liao, N.; and Riedl, M. 2018. Co-creative level design via machine learning. *arXiv preprint arXiv:1809.09420*.

Ha, D., and Eck, D. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

Huang, C.-Z. A.; Vaswani, A.; Uszkoreit, J.; Simon, I.; Hawthorne, C.; Shazeer, N.; Dai, A. M.; Hoffman, M. D.; Dinculescu, M.; and Eck, D. 2018. Music transformer: Generating music with long-term structure.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Levy, O., and Goldberg, Y. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, 2177–2185.

Lopes, R. G.; Ha, D.; Eck, D.; and Shlens, J. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE International Conference on Computer Vision*, 7930–7939.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1(8):9.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Schneider, P. J. 1990. An algorithm for automatically fitting digitized curves. In *Graphics gems*, 612–626. Academic Press Professional, Inc.

Serban, I. V.; Sordoni, A.; Lowe, R.; Charlin, L.; Pineau, J.; Courville, A.; and Bengio, Y. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Shaham, T. R.; Dekel, T.; and Michaeli, T. 2019. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, 4570–4580.

Singh, K. Y.; Davis, N. M.; Hsiao, C.-P.; Macias, R.; Lin, B.; and Magerko, B. 2017. Unified classification and generation networks for co-creative systems. In *ICCC*, 237–244.

Suwa, M.; Gero, J. S.; and Purcell, T. 1998. The roles of sketches in early conceptual design processes. In *Proceedings of Twentieth Annual Meeting of the Cognitive Science Society*, 1043–1048. Lawrence Erlbaum Hillsdale, New Jersey.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, J.; He, Y.; Tian, H.; and Cai, H. 2008. Retrieving 3d cad model by freehand sketches for design reuse. *Advanced Engineering Informatics* 22(3):385–392.

Wieluch, S., and Schwenker, F. 2019. Dropout induced noise for co-creative gan systems. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 0–0.

Xu, P.; Song, Z.; Yin, Q.; Song, Y.-Z.; and Wang, L. 2020. Deep self-supervised representation learning for free-hand sketch. *arXiv preprint arXiv:2002.00867*.

Xu, P.; Joshi, C. K.; and Bresson, X. 2019. Multi-graph transformer for free-hand sketch recognition. *arXiv preprint arXiv:1912.11258*.

Yu, Q.; Liu, F.; Song, Y.-Z.; Xiang, T.; Hospedales, T. M.; and Loy, C.-C. 2016. Sketch me that shoe. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 799–807.

Yu, J.; Xu, X.; Gao, F.; Shi, S.; Wang, M.; Tao, D.; and Huang, Q. 2020. Toward realistic face photo-sketch synthesis via composition-aided gans. *IEEE Transactions on Cybernetics*.