Assignment 3: Neural Rendering and Shape Processing

CPSC 532R/533R Visual AI
by Helge Rhodin and Yuchi Zhang

This assignment is on neural rendering and shape processing—computer graphics. We provide you with a dataset of 2D icons and corresponding vector graphics as shown in Figure 1. It stems from a line of work on translating low-resolution icons to visually appealing vector forms and was kindly provided by Sheffer et al. [1] for the purpose of this assignment.

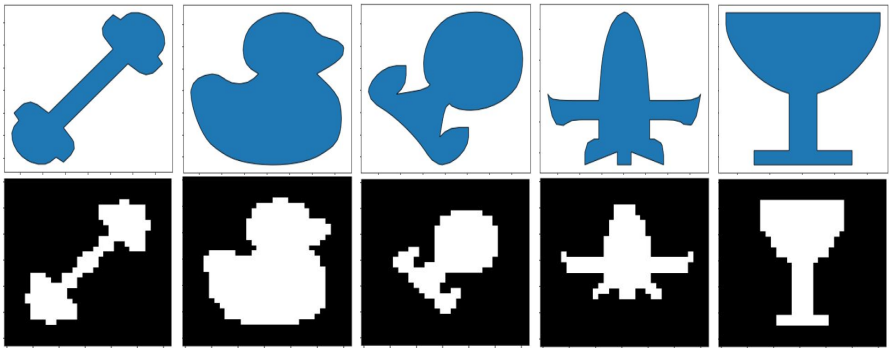

Figure 1: Icon vector graphics and their bitmap representation.

The overall goal of this assignment is to find transformation between icons. We provide the `ImagerIcon` dataset as an HDF5 file. As usual, the `Assignment3_TaskI.ipynb` notebook provides dataloading, training and validation splits, as well as display and training functionality. Compatibility of the developed neural networks with color images is ensured by storing the contained 32 x 32 icon bitmaps as $3 \times W \times H$ tensors. Vector graphics are represented as polygons with $N = 96$ vertices and are stored as $2 \times N$ tensors, with neighboring points stored sequentially. The polygon representation with a fixed number of vertices was attained by subsampling the originally curved vector graphics.

- **Task I:** Implement a neural network that takes a batch of vector images as input and outputs corresponding bitmap images, i.e., that can map the top row of Figure 1 to the bottom row. This is an instance of neural rendering via supervised learning. The core of your task is to implement a neural network that uses transposed convolution (aka. deconvolution) with stride 2 or more. On top, you can use any other neural network layers, such as `ReLU`, `linear`, `batch norm`, `max pooling`, and `bilinear interpolation` operations. Chose a suitable loss function as the criterion.
- **Task II:** Train the provided simple auto encoder (AE). Inspect the red dot in the output, which indicates the starting point of the polygon. Note, although the polygon is drawn as a closed loop, we have to store it linearly in memory. Application of the MSE loss compares the positions of the polygon vertices in the order they are stored in memory. However, the start changes arbitrarily for each of the vector icons and is difficult to predict for the network. The Chamfer distance is designed to handle such mismatch via nearest neighbors. Use the provided

`chamfer_distance` function to train the AE a second time and report what you observe.
There seems to be a problem about the order of points. Fortunately, we know the order, only the start point is arbitrary. Implement a new distance metric `roll_invariant_MSE` that is inspired by Chamfer. It should compare the predicted polygon to all possible shifts of the label and return the MSE that yields the lowers error. A shift by one here means to roll all points in the tensor with the last element becoming the first, while all 2D position are kept the same. We provide a function skeleton as starting point and utility functions `roll_1` and `roll_2` to roll elements along different axes.

- **Task III:** The new training loop in Task III contains data augmentation code that randomizes the start point for every iteration. Although the new loss makes the objective invariant, the network remains sensitive to the order of points and does not train well (give it a try). It is your task to improve the provided autoencoder to be invariant to the polygon starting point by burrowing ideas from PointNet (c.f. lecture 5). Use 1D convolutions (e.g., with kernel size 1 or 3), ReLU activations, and global pooling to construct the encoder that is free of linear (fully-connected) layers. For us, 6 to 8 convolutional layers and two max-pooling layers worked, but other architectures are likely to improve. **Warning:** 1D convolution with `padding_mode='circular'` would be perfectly suited but the PyTorch implementation appears broken. Although zero padding introduces boundary artifacts and is hence sensitive to the point order, this effect is minimal. You are allowed to use convolutions with zero padding.
- **Task IV:** The representation learned by autoencoders is well-suited for interpolating within the manifold of feasible shapes, a so-called *shape space*. It is your task to test different ways of interpolating between two vector icons. a) Interpolate two point clouds M_1 and M_2 from the training dataloader, i.e., $M_{\text{out}} = \lambda M_1 + (1-\lambda)M_2$. Display the outcome for $\lambda \in (0, 0.2, \ldots, 1)$. b) Apply your encoder from task III on M_1 and M_2 and save the corresponding latent codes h_1 and h_2. Interpolate these latent codes and reconstruct them using the decoder. Plot these as before. Try a few different icon pairs and describe and explain the difference between a) and b). Do you see issues? Could a VAE overcome these problems? Explain why.

Hints and comments.

- It is not immediately obvious whether your `roll_invariant_MSE` distance implementation is correct. To be sure, construct a ty example for which you can compute the result by hand.
- The provided dataset is relatively small scale. One epoch contains only a handful of images. You will have to run many epochs.
- Training deeper neural networks takes a while. Don't draw conclusions too early. Particularly your network from Task III may take 1000 or 2000 iterations to show reasonable results. See Figure 2 for training curve example.
- Due to the small dataset, you have to focus on a compact architecture with relatively few parameters to avoid overfitting. Note that this is not an unusual setting. For instance, it is a recent trend to train GANs on a single image [2]. Nevertheless, we do not expect your networks to generalize well to the validation set.

Optional add-ons if you are curious or want to play save. Solutions to each add-on task give up to one bonus point that can offset points deduced in the main tasks. The

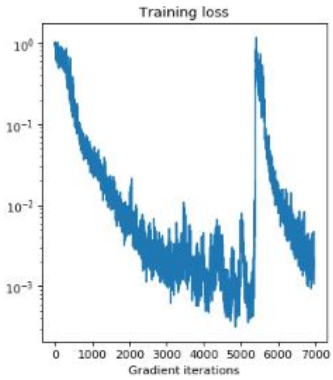

Figure 2: Training the network in Task III can take more than 1000 iterations and the error may spike. The depicted spiking pattern is common when using the Adam optimizer.

maximum number of points is still capped at 10.

- Add a prior term for Task II that encourages smooth polygon reconstructions or a distance that penalizes the difference between the Laplacian of the predicted polygon and the Graph Laplacian of the label.
- Try to implement a neural rendering model that generalizes well, despite the small training set. Special network architectures, hardcoded transformations, as well as data augmentation appears viable. This is a quite hard challenge and could be turned into a course project!
- Turn your Task III AE into a VAE that has better latent space interpolation behavior. For us, down-weighting the KL divergence term (prior) by 0.001 in relation to the log likelihood (data term) was necessary.

Submission. Once finished, submit your jupyter notebook on Canvas. If you have dependencies, add them to a .zip archive. Name your submission `assignment3_firstName_lastName.ipynb` (or `.zip`). The jupyter notebook that you submit must contain the cell output from a clean execution (restart kernel and run all cells sequentially) to ease grading. If some of your outputs are displayed with external tools, such as Tensorboard, please include screenshots of those. Do not submit downloaded datasets and other temporary files (max 5 MB total submission size).

References

[1] Shayan Hoshyari, Edoardo A. Dominici, Alla Sheffer, Nathan Carr, Zhaowen Wang, Duygu Ceylan, and I-Chao Shen *Perception-driven semi-structured boundary vectorization*, ACM Transactions on Graphics (TOG), 2018.

[2] Tamar R. Shaham, Tali Dekel and Tomer Michaeli *SinGAN: Learning a generative model from a single natural image*, ICCV, 2019.