Report for Week 13

November 23, 2021

Information: *A brief report mainly about the neural network I'm working on for predicting the number of control points*

Written by: *Zihao Xu*

E-mail: *xu1376@purdue.edu*

1 Problem Statement

1.1 Goal

- Find a way to describe a given curve from hand sketches using a B-spline as precisely as possible when not having any prior information about the curve.

1.2 Current status

According to literature review, most existing fitting methods requires the prior information of the number of control points. Some sketch interfaces do not care about the number of control points and deals with abundant number of control points. Few fitting methods use data-driven approach and solve for number of control points in a small range (e.g. 5-9).

The problem can be divided into two parts:

- [**Completed**] Given a fixed number of control points, solve for the optimization problem so that the reconstruction error is minimized by placing the control points.
  - Use the method **splprep()** in **scipy** package for stability.
  - While the package itself did not provide a handle controlling the number of control points, designing the knot vector based on curve types can produce desired number of control points
  - Compatible for both open and closed curves.
- [**In progress**] For an arbitrary curve after segmentation of the sketch interface, find the optimal number of control points.
  - [Tried] Based on local features such as derivatives and curvatures, compute the required number of control points
    - Not stable as the input curves are usually not smooth
    - Heavily depends on the parametric value dimension and requires a stable normalization process
  - [In progress] Construct a network learns the optimal number of control points and use it for prediction.

For more details and reviewed literatures, please refer to previous weekly reports.

2 Trial with CNN

2.1 Generate the Dataset

To train a model predicting the number of control points of a curve, the first step is to obtain a dataset where the curve and ground truth are available. Since right now I do not know how to find a existing B-spline curve library, I generated a dataset by randomly select the position of control points and then generate corresponding B-splines using scipy package with computed knot vectors.

2.1.1 Randomly sample the control points

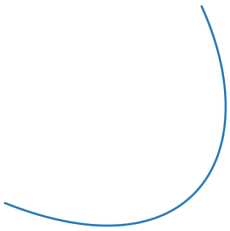
In detail, I assume all the curves start from the origin and sample the next control point uniformly in the unit circle. To avoid too many self-intersections, all the rest control points are sampled from the unit semicircle around the previous control point. Taking into consideration the prediction of the number of control points for closed curves, I've developed both the algorithms to generate closed curves and open curves.

Even though right now I only want to focus on the images and corresponding number of control points, all the information required to build the curves are saved for potential use (e.g. the positions of control points might be used if a RNN is finally selected). The information is stored in the json file and can be loaded as a dictionary object.

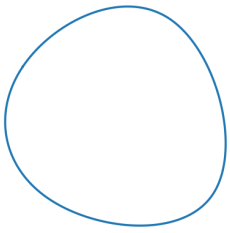
2.1.2 Generated Curves

Here are some examples showing the generated curves.

- **4** control points. Open curve.

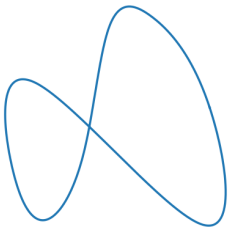


- **4** control points. Closed curve.

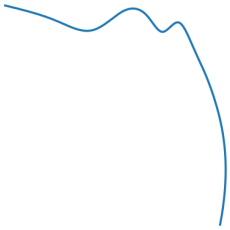


- **8** control points. Open curve.

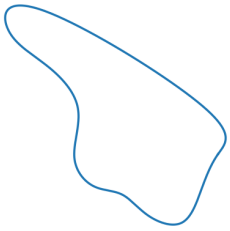
- **8** control points. Closed curve.



- **12** control points. Open curve.



- **12** control points. Closed curve.

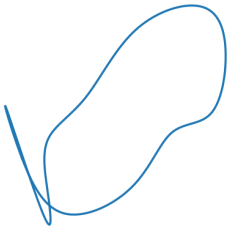


I manually set the seeds for all random processes so the reproducibility of the dataset I used for training and validation is ensured. It needs to be mentioned that in the generated dataset, there are a lot of 'bad' plots.

- **Similar plots with different number of control points**

The above two images look similar while the first one is controlled by four control points and the second on is controlled by five control points

- **Curves that might be impossible to occur in hand sketches**



The above curve is controlled by 13 control points, which shows that strange shapes might occur with my sampling strategy.

These two kinds of 'bad' plots raise two great challenges for any data-driven detection methods using this dataset:

- Difficult to provide confident and accurate predictions since sometimes two plots with different number of control points are nearly the same.
- The distribution in the training dataset and validation dataset differs from each other greatly.

Future works are needed to solve these two problems.

2.2 A network similar to ResNet-18

Here I'm trying to use a convolutional neural network because the CNN structure is supposed to be able to capture image features better. As a start point, I use a structure almost the same as **ResNet-18** with **xavier uniform initialization**.

2.2.1 Training Process

Dataset: Due to limitation of my hardware (I only have one RTX 2070 with MaxQ design), a small dataset is used for training and validation.

- Only generated curves with variable control points ranging from **4** to **15**
- For each class (number of control points), **1000** open curves and **1000** closed curves are generated for training, **200** open curves and **200** closed curves are generated for validation.

Optimizer: For better local stability, the **SGD** optimizer is used.

Loss: As other classification problem, then cross entropy loss is used.

Epochs: 30 epochs are used to observe the convergence of the training loss.

Models: Three different models are trained.

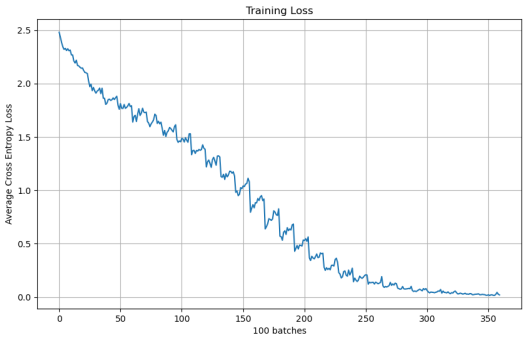
- Model only trained with open curves
- Model only trained with closed curves
- Model trained with both open curves and closed curves.

Time: The training process roughly took me 5 hours on my own computer with single GPU.

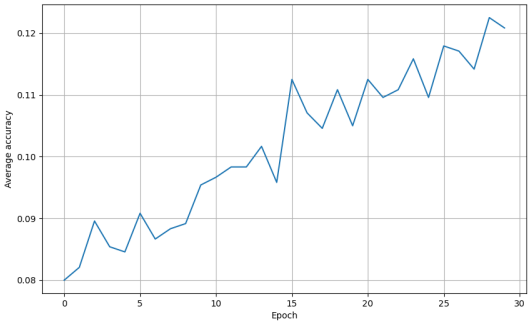
2.2.2 Model only trained with open curves

Here are the results of model only trained with open curves.

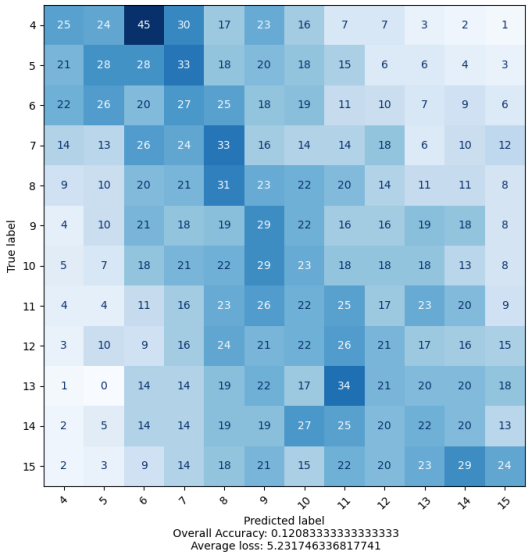
- **Training loss**



- **Validation Accuracy**



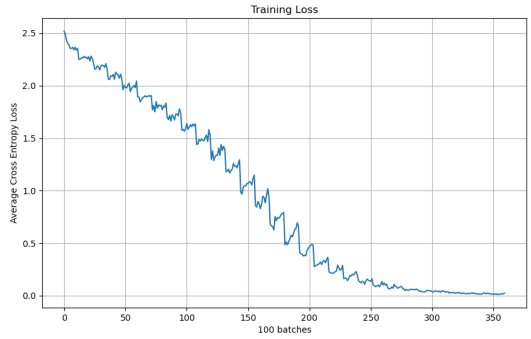
- **Confusion Matrix for Validation**



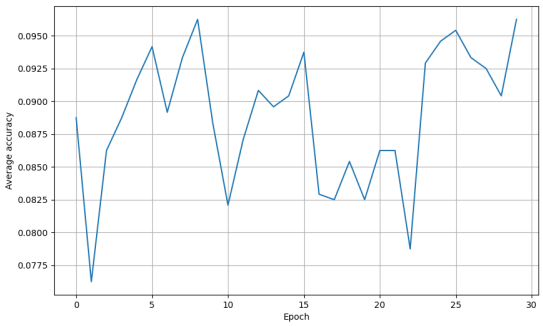
2.2.3 Model only trained with closed curves

Here are the results of model only trained with closed curves.

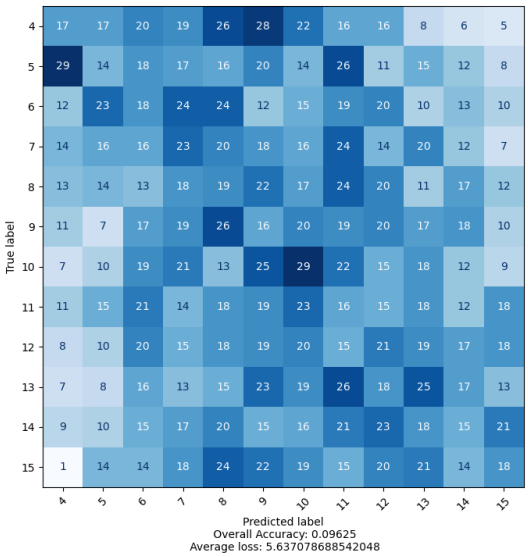
- **Training loss**



- **Validation Accuracy**

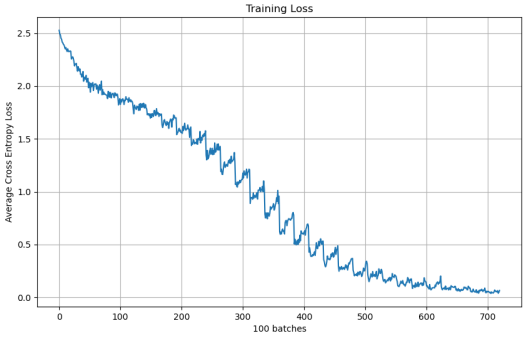


- **Confusion Matrix for Validation**

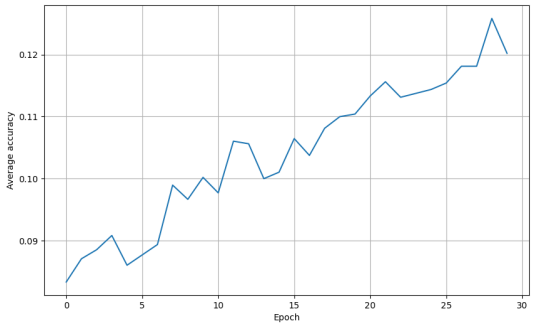


2.2.4 Model trained with both open curves and closed curves

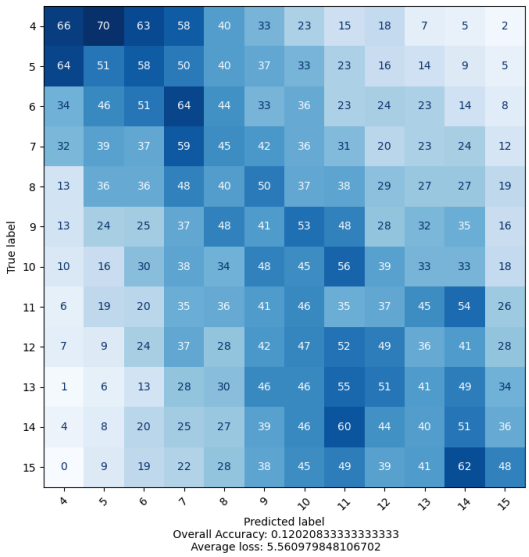
- **Training loss**



- **Validation Accuracy**



- **Confusion Matrix for Validation**



2.2.5 Conclusion

Considering the fact that the training loss are almost zero while the validation accuracy almost has no improvements, I checked the validation loss and surprisingly find that the validation loss is even increasing. Obviously, the model is overfitting the dataset, which means it is trying to memorize the dataset rather than finding some general rules from the dataset. There're several possible reasons:

- The **size of training dataset** is too small. For each class, only 1000 images are used to train the model.
- The **complexity of the model** is too huge. This model has about 12,000,000 parameters which might be too huge.
- The **distribution** of the training dataset and the validation dataset differs greatly. Although they are sampled based on the same rule, it's till possible that their distributions are slightly different. especially with such a small dataset.
- Given that plots with different number of control points might be similar, the **disturbance to the model** is much higher than expectation.

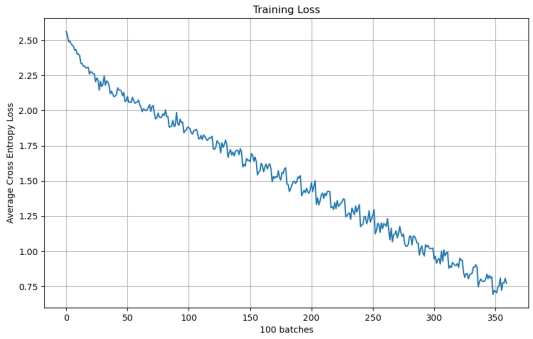
2.3 A much simpler network

Considering the possible reasons in the conclusion above, the only thing I can test in a short time is to try with a much simpler neural network. All the other settings for the training process are the same.

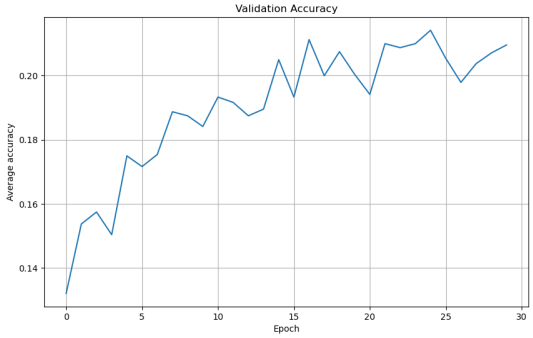
Time: The training process roughly took me 3 hours on my own computer.

2.3.1 Simple model only trained with open curves

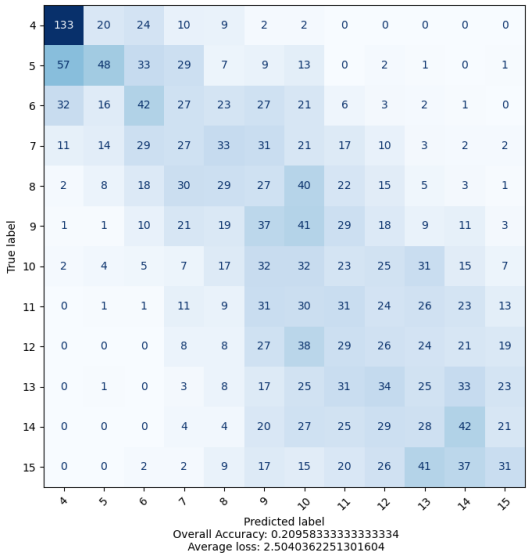
- **Training loss**



- **Validation Accuracy**

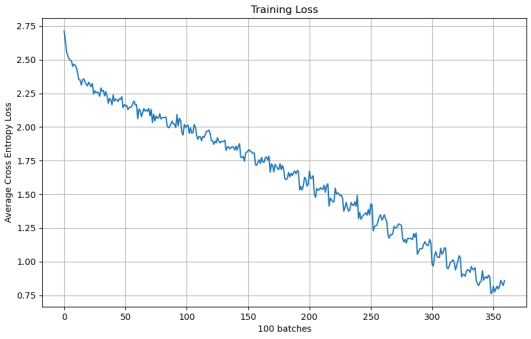


- **Confusion Matrix for Validation**

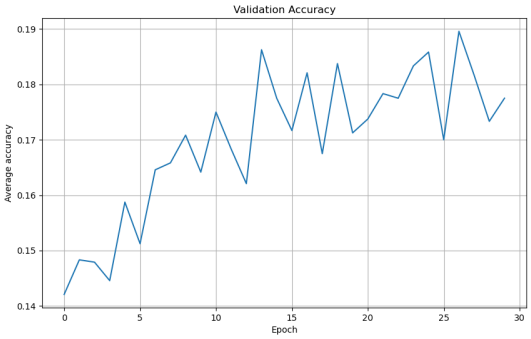


2.3.2 Simple model only trained with closed curves

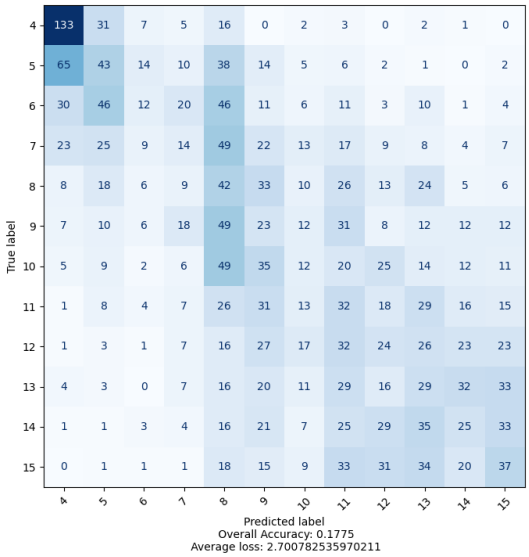
- **Training loss**



- **Validation Accuracy**

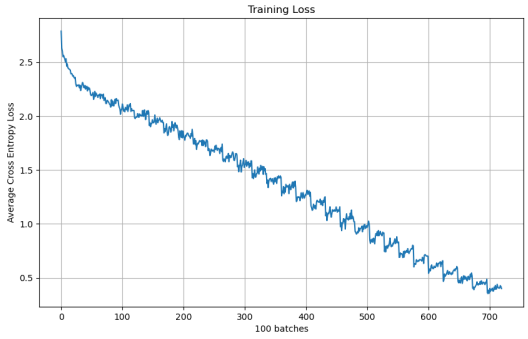


- **Confusion Matrix for Validation**

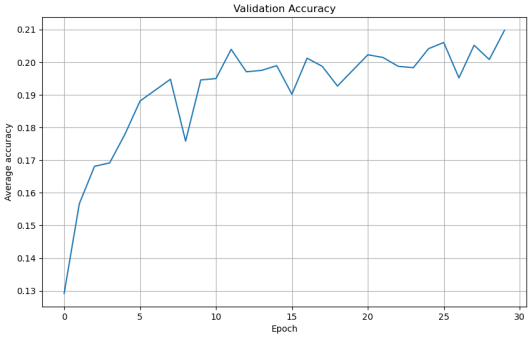


2.3.3 Simple model trained with both open curves and closed curves

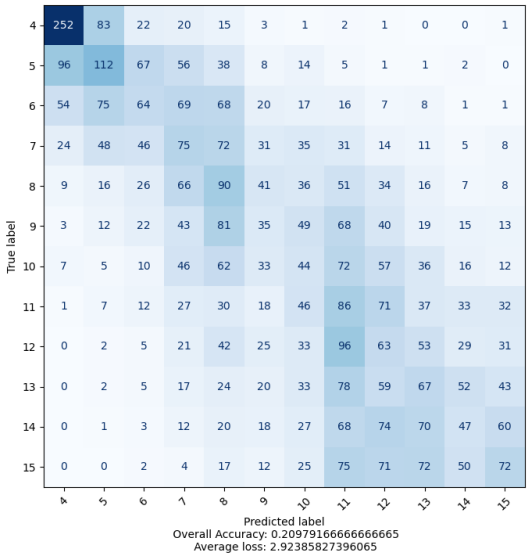
- **Training loss**



- **Validation Accuracy**



- **Confusion Matrix for Validation**



2.3.4 Conclusion

It is easy to observe that the overall accuracy is slightly higher than that of the complex model. However, it seems that the model still begin to overfit as the training loss keeps decreasing while the validation accuracy almost keeps the same in the last several epochs. Therefore, the most possible reasons for these poor performances might be the poor distributions among the training dataset and small size of the dataset.

In the meanwhile, it seems that the closed curve features are somehow more difficult to learn, indicated by the fact that both the complex model and the simple model are performing worse with closed curves. Still, it might also be mainly caused by the dataset, which means the 'quality' of closed curves are even worse.

3 Summary

3.1 Completed works

So far, I'm focusing on finding the optimal number of control points using machine learning methods. The workflow of the overall learning algorithm has been successfully created and initialized. The completed works are as follows.

- Compute knot vectors for both open and closed curves.
- Sample control points of different numbers.
- Generate curve images and save all the related information as a dictionary object
- Build a complex model similar to ResNet-18 and a relatively simpler model to compare the performances
- Construct the training and validation process and check the models' performance when trained with different types of curves (e.g. open curves only, closed curves only, and both open and closed curves)
- Use the trained model to predict the number of control points given a numpy.ndarray representing the sampled curve from the sketch interface.

3.2 Github Repo

I've uploaded the codes I'm working on to Github and written a detailed description, so that anyone who would also like to work on this project can get familiar with what I've done and what I'm doing conveniently.

3.3 Future works

Further improvements are required since the current model is not working well on the validation dataset. Possible future works are as follows.

- Improve the generated **dataset distributions**
  - Make curves controlled by different number of control points more distinct
  - Make the distribution of sampled curves more rational and similar to real drawings
- Train the complex model with a **bigger dataset** to avoid overfitting.
  - This might not be possible to be tested on my own PC for the low training speed.
- Modify the **model structure** and hyperparameters for better performance.
  - The direction of modification should be discussed since I'm not sure which factors are more important.
- Get **an optimal range** instead of an optimal number to decrease the dependence on the model's deterministic output.