MidcurveNN with Symbiosis

# Project info

Github Repo: <https://github.com/yogeshhk/MidcurveNN>

Team

* Students:
  + Neelabh Paul (NP) [neelabh.paul.btech2018@sitpune.edu.in](mailto:neelabh.paul.btech2018@sitpune.edu.in)
  + Anindyadeep Sannigrahi (AS) [proanindyadeep@gmail.com](mailto:proanindyadeep@gmail.com)
* Internal Guide: Rahee Walambe (RW) [rahee.walambe@sitpune.edu.in](mailto:rahee.walambe@sitpune.edu.in)
* External Guide: Yogesh Kulkarni (YK) [yogeshkulkarni@yahoo.com](mailto:yogeshkulkarni@yahoo.com)

# ToDos

* Image based approaches (NP)
  + <https://github.com/yogeshhk/MidcurveNN/issues/8>
* Graph-based approaches (AS)
  + <https://github.com/yogeshhk/MidcurveNN/issues/7>
  + <https://github.com/yogeshhk/MidcurveNN/issues/9>
* Implement 3DPoselite for MidcurveNN <https://openaccess.thecvf.com/content/WACV2021/papers/Dani_3DPoseLite_A_Compact_3D_Pose_Estimation_Using_Node_Embeddings_WACV_2021_paper.pdf>

# Notes

* Is VAE correct architecture to solve the Graph Dimension Reduction problem?

Standard Auto-Encoder decode has both inputs and outputs the same. So not suitable for MidcurveNN. Variational Autoencoder Decoders are similar to AE except that its latent vector follows normal distribution. Uses prediction of mean and std deviation first to fix the latent vector. But still, input and outputs have to be the same. So, even VAEs do not look suitable for MidcurveNN. But 3DPoselite seems to VAE and still attempts to train Dimension Addition. The difference is that it has 2 Encoders for skeleton and mesh then looks for similarity (?) like the Siamese network (?)

# Queries

* Which GNN framework to use?
  + <https://blog.paperspace.com/geometric-deep-learning-framework-comparison/>
  + **Pytorch-geometric** or DGL or Tf-GNN?
* Which conferences to target for publications?
  + <TBD>

# Discussions

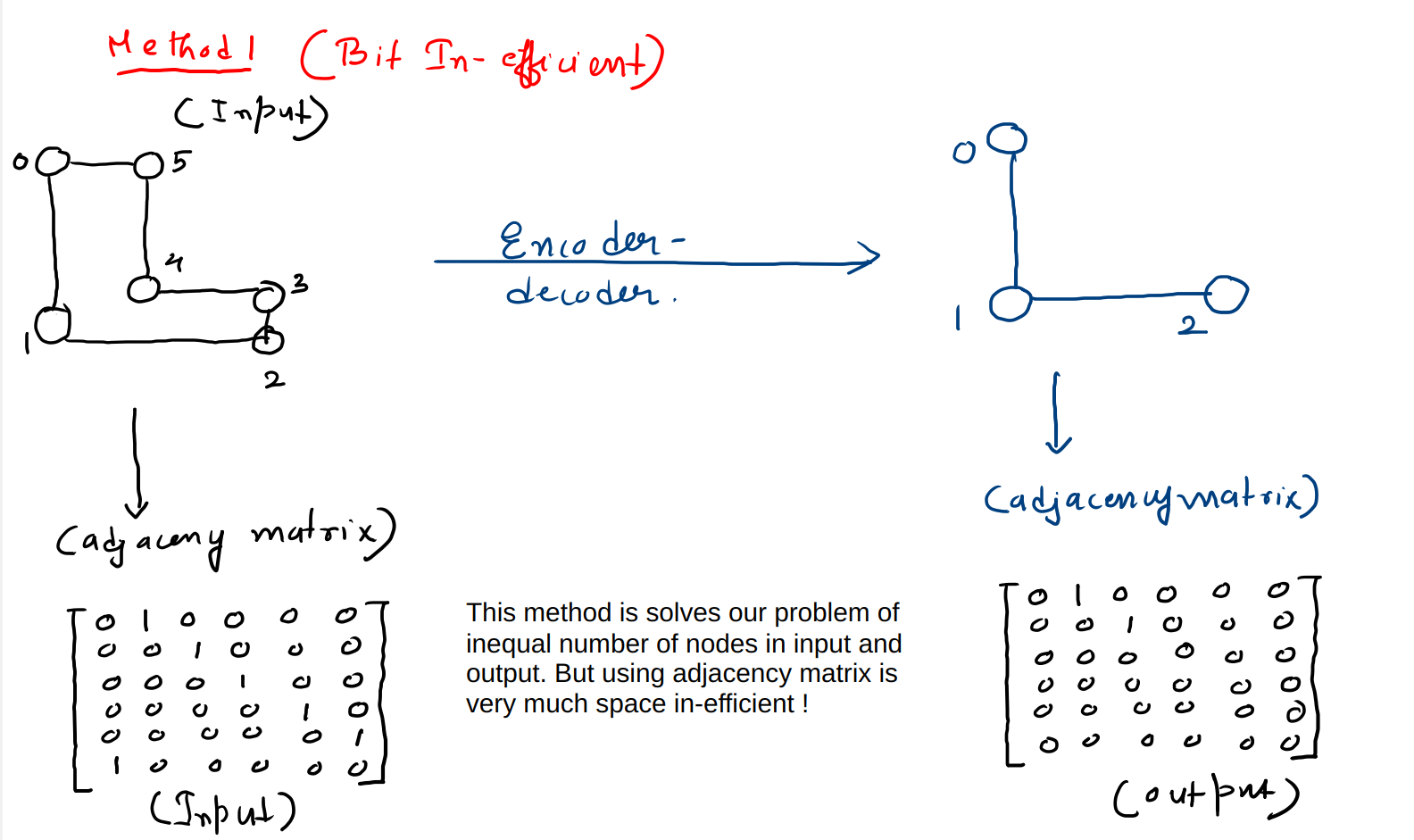
## 28-Jan-2022 Unequal Graph Sizes issue

### [Anindyadeep]

Coming straight to the point, As we had discussed in the previous meeting, the only problem in this initial stage is the unequal number of input-output nodes.

I had come up with two initial possible solutions. One is somewhat space in-efficient, whereas the other is a somewhat weird kind of solution.

**Solution 1 (Using adjacency matrices, rather than using adjacency list)**

****

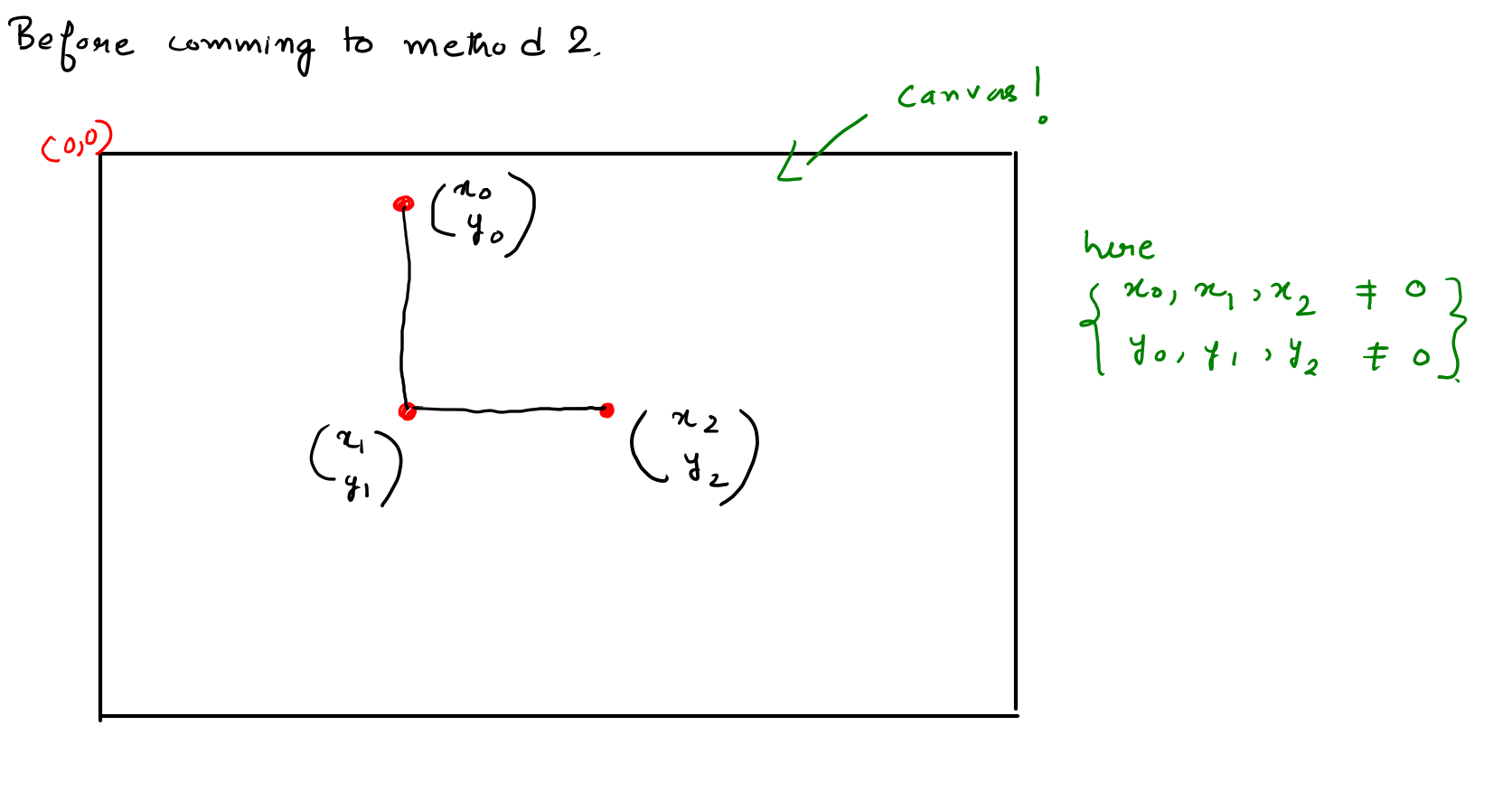
This is one of the potential solutions, considering that the size of graphs are kinda small. So suppose my input has 6 nodes, so we construct an adj matrix of size [6, 6] and we also get a [6,6] matrix output. And that output adj matrix will definitely have a lesser number of ones, and the indices of the ones, would be my connection of the required output mid-curve.

Problems with this approach:

1. Matrices are very much sparse in nature. So if we see in the above fig 1, then it's clear that both input and output matrices are very much sparse in nature. Which might potentially affect both the efficiency and the performance of the model in the long run.
2. We have to code everything from scratch. As this problem could be framed in a graph auto-encoder problem, so there is no such way in libraries, where there is an option to pass adj matrices rather than edge indices. So for that we have to create the model from scratch. But that's doable.

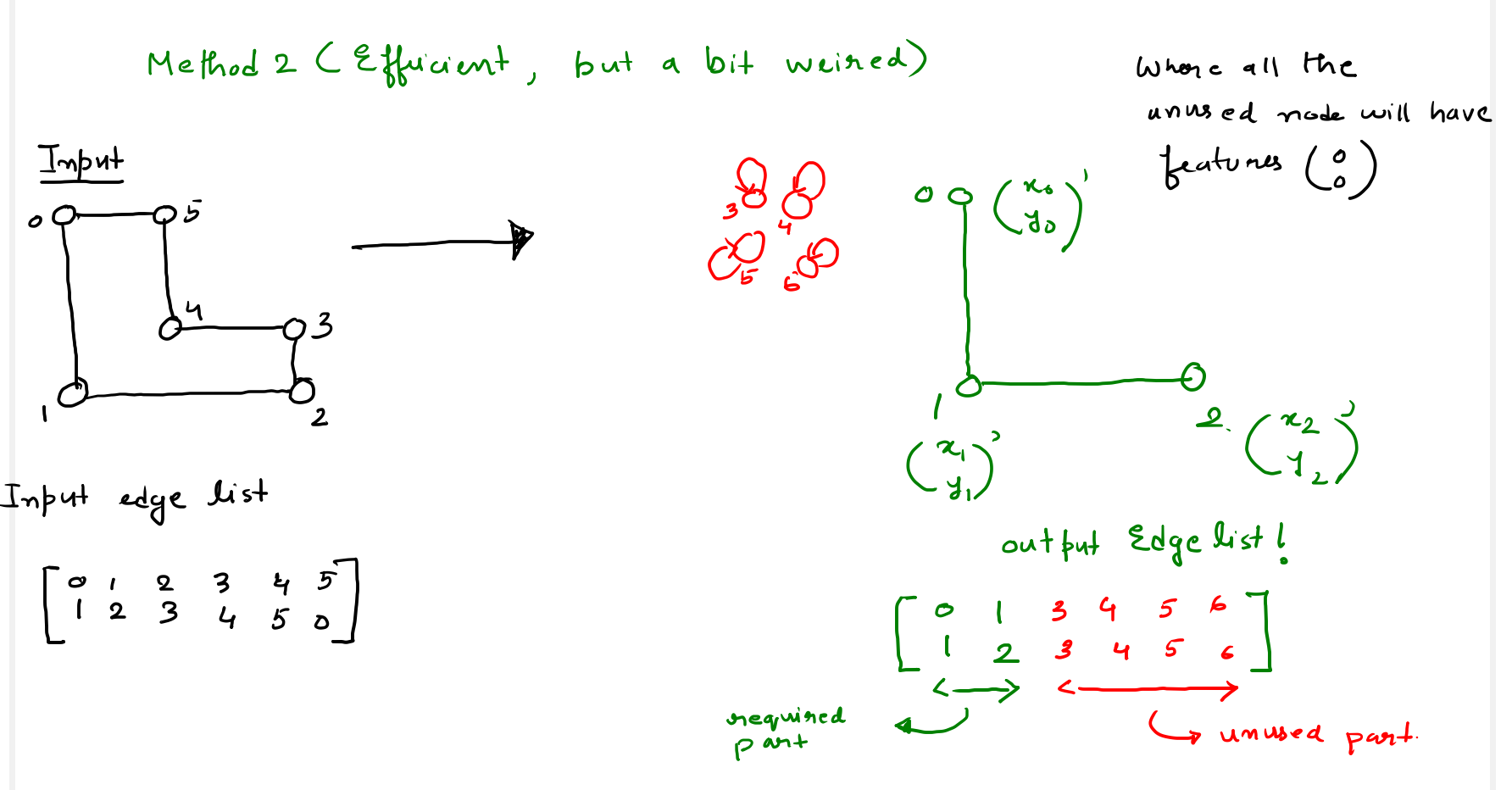
**Solution 2 (using adjacency list in a different way)**

This approach requires a bit of input preparation. Suppose I have a graph with 6 nodes, connected in an “L” fashion. So my adj list will be of shape (2, 6) and each node will have some kind of (x, y) coordinate features. So, if we draw this thing in a canvas, then it will somewhat look like this:



As you can see, in this figure 2 above, the start of the canvas is (0, 0) and we draw the feature vectors in the canvas of the output graph. **Also I am assuming this hypothesis that, the number of the nodes in the mid-curve (output graph) will always be far less than the number of the input curve (graph)**

Now consider this figure below:



As you can see, that input edge list has some valid edges, where each node has some kind of valid non-zero coordinate (even if some valid coordinate has some zero value, that's not a problem). So now lets see the output adj list, which just has 3 nodes, and the expected shape of the output adj list was (2, 2), and would look something like this:

[[0, 1]

[1. 2]]

But we also add (padd) this adjacency list to match the number of nodes of the original input graph. **Where all of these left over edge indices will be some nodes, but all of these nodes will have a feature vector of 0, so all of these nodes will be a (0, 0) vector, and all of them will be pointing themselves.** As you can see in the above fig 3. So during the time of seeing the output in the canvas, we will trim the output adj list till the part it is required, and also, for the reconstruction loss, we will compare the reconstructed edgelist with the actual edge list till the required part, as all the rest of the part is JUNK. And if that's the case, then all of the initial problems could be solved.

**And finally, another approach could be using Graph RNN, that is a way harder and more complex approach.**

### [Yogesh]

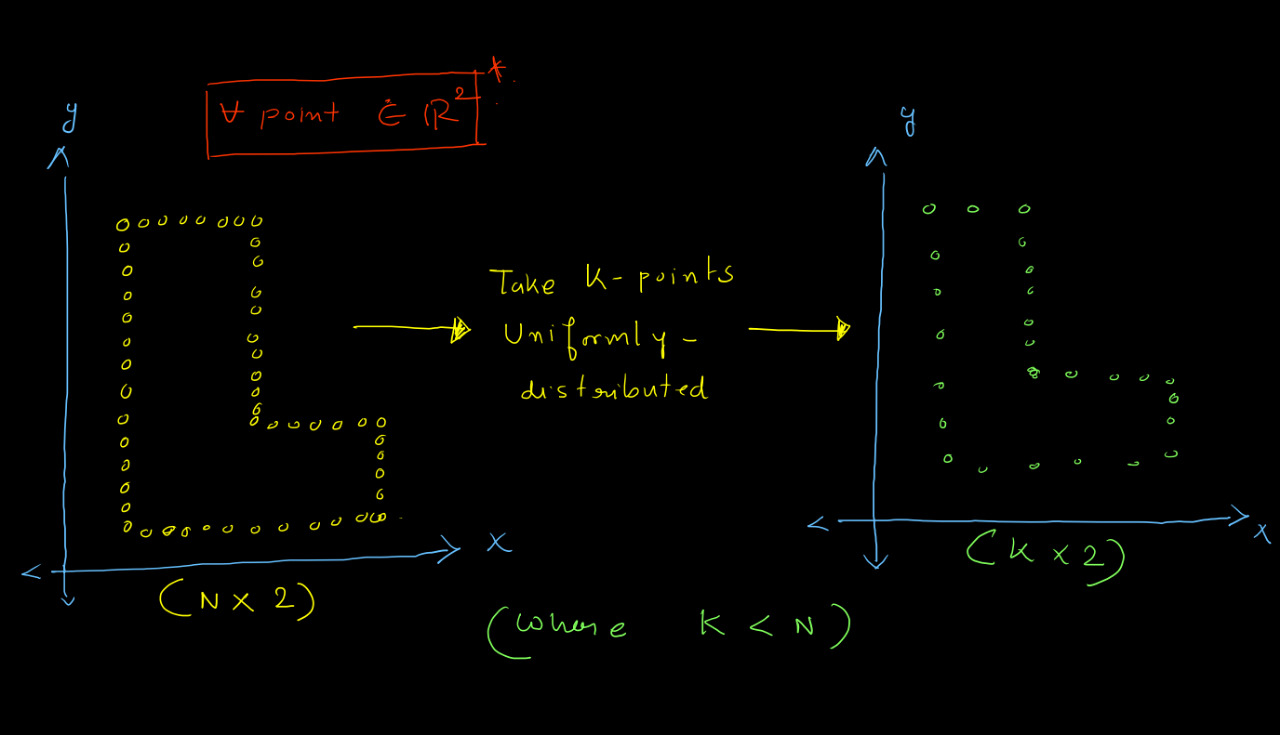
Adjacency matrix may not be the correct way to address the unequal graph sizes as node-ids of inputs are different from node-ids of output. They are different nodes altogether. Problem is challenging as this is just not link prediction between existing nodes, but actually predicting new nodes themselves. So, 'unused nodes' (approach #2) cannot be used during training, as they themselves need to be predicted at inference times.

**Update Jan 31st**

After some research and doing some experiments, I got a solution that solves the problem of unequal input, output nodes of the point clouds in one single example. So previously, suppose for any example, if the profile point cloud is having 1000 nodes then the mid curve point cloud has 450 points. This had two problems:

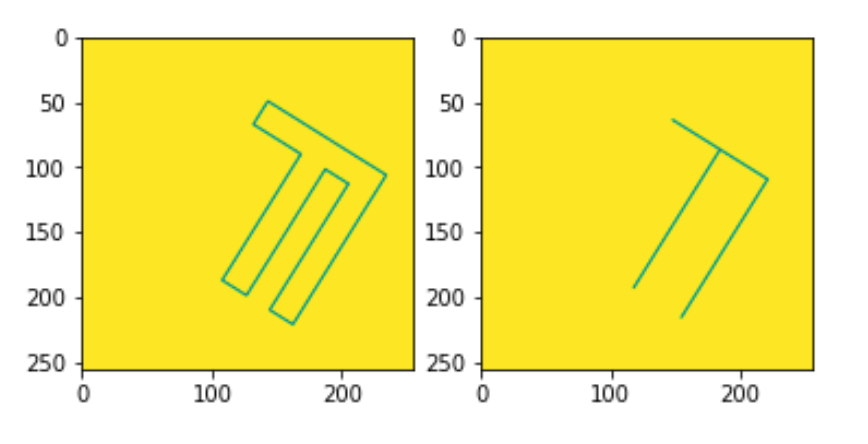
1. We might have to search or made a whole new algorithm to tackle with this complexity
2. More computations
3. No guarantee for results

So for this, we need to tackle this problem. And this is done by the following way, shown in the image below:



Suppose we are having two different point clouds (profile and mid curve) with different nodes. We need to sample some number of these 2d points in some distribution (taken gaussian) such that both have the same kind of nodes at the end. In this example shown in the figure above, the left point cloud shows the point cloud of profile (of suppose 1000 points) and the right is the sampled point cloud of just 256 points. So the shape remains intact, we also get the approximate edge list for all these point clouds from KNN-graph algorithm. So here are some more examples:

This is the initial images of the profile and its mid curve



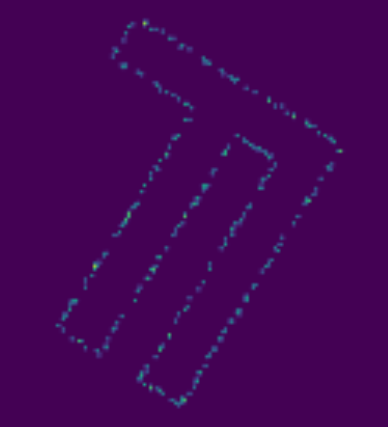
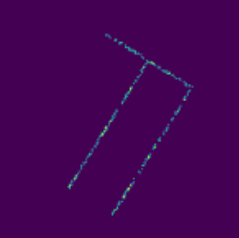
The initial (unequal) point clouds of profile and mid curve



The sampled point clouds (300 pts each) for both profile and mid curve

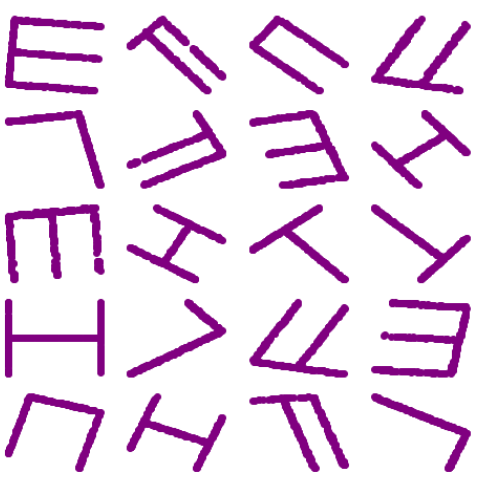
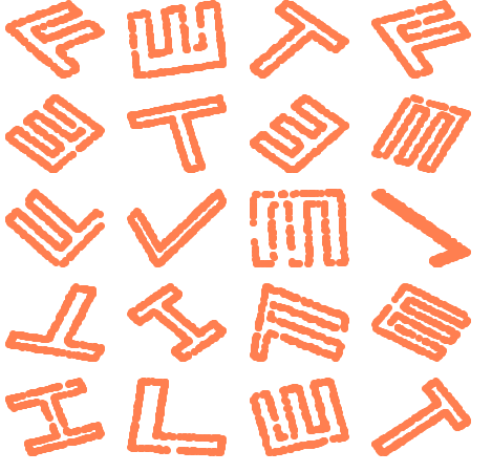


Now after this we revert back these point clouds to corresponding image



These above are .jpeg images of the sampled point cloud. As we care about our mid curve and we can see that mid curve images are a lot more dense, also if we become successful in generating similar sort of point clouds of mid curve, then we could further devise a simple algorithm that will make these points dense based on nearest neighbors.

So here some more examples of the sampled point clouds of some random profiles and some random mid curves.



Advantages now we get:

1. Input and output are now same
2. It is now Pytorch Geometric **Data** class compatible, so batching is possible.
3. We can see that some profile point clouds are kind of distorted, so if we any how device our generative model that can produce similar kind of mid curves shown above (right), then its an advantage in real life, as if any how the profile image is kind of distorted also, we could get the correct point cloud. (Please correct me in this point if, I am wrong)
4. We could now have more than 1 algorithm to frame this problem.

Now, if this makes some kind of sense, then here is the next steps:

1. Making the data loader to batch these data
2. This problem is somewhat similar to molecule generation, so we could now follow pure GNN based approaches for multiple graphs.
3. Or we can modify the Point Net ++ based approach and use it for our generative modeling.
4. In the end our aim should be to make a permutation invariant, affine transformation invariant based generative models.
5. So our goal would be to generate **node features (point of mid curve)** and also the edge list and compare those with the actual mid curve and its list, as a **reconstruction loss** and also need to get the **KL loss**. So we can start with very simple approaches, (but it needs some research, for multi graph generative modeling), and implementation get successful, and if we manage to get some baseline results we could step forward by:
   1. Making architecture complex.
   2. Introducing one more feature which is the angle between two point vectors
   3. Introducing edge attributes, as a vector containing the (m, c) of a line, as edges from two nodes can be thought of as a line equation.

So, let's hope for the best.