MOTION DIFFUSION MODEL

Puspak Priyabrata Sahu 21676

❖ Objective

This assignment focuses on implementing a diffusion model for human dance motion. The goal is to train a generative model that can learn from a single dance sequence and generate new dance variations inspired by the training sequence.

- The first task is to implement a **UNet model called "punet"** that maps a sequence of 2D key points to another sequence of 2D key points. This will be used to generate a dataset of small sequences by sampling a set of N equal-length sequences from the original sequence of time T.
- The second task is to train a **Denoising Diffusion Probabilistic Model (DDPM)** using the standard Mean Squared Error (MSE) diffusion loss to denoise the motion sequence. The training curves and observations are then reported.
- Finally, **multiple output sequences** are generated from the trained DDPM models, and a dance video is rendered using the provided renderer.py script.

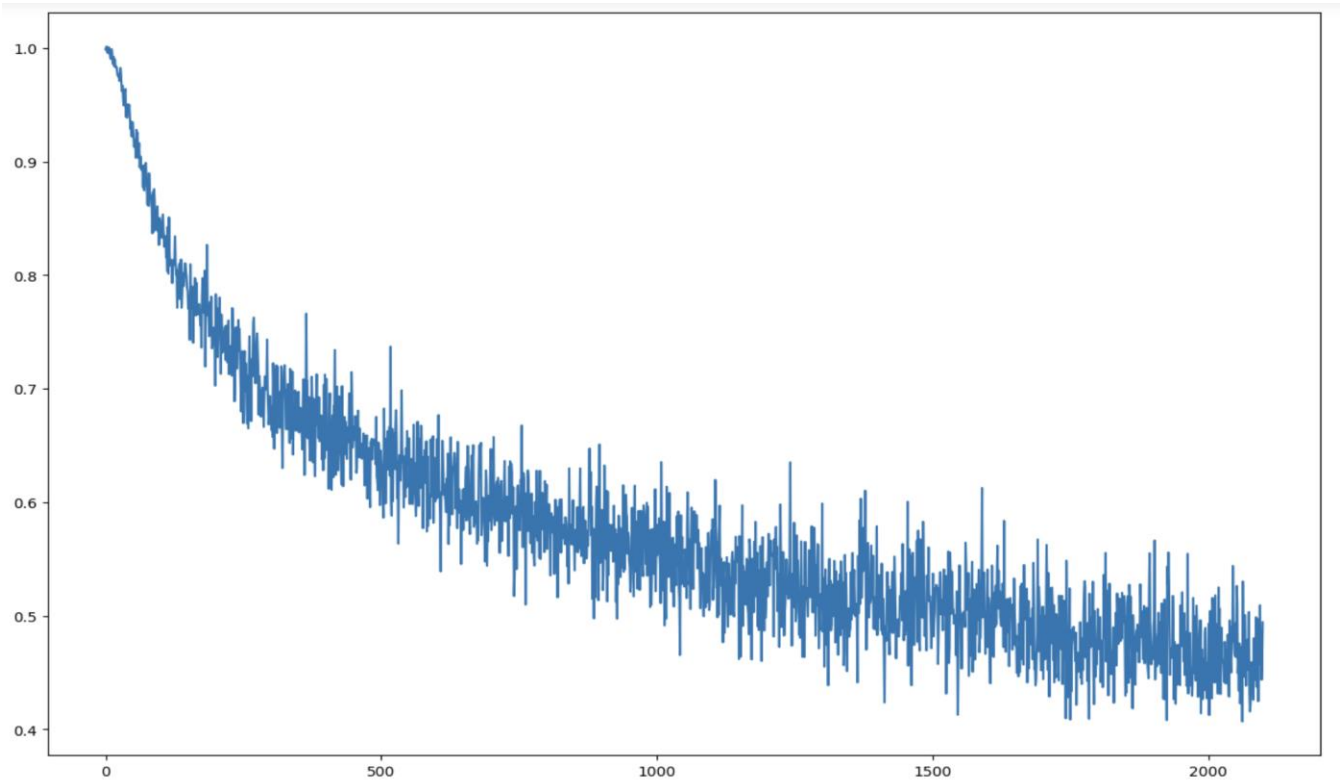
❖ Train methodology

- We are given approx. 895 frames where each frame consists of 20 joints, each joint being a point in the 2D plane.
- **make_Frames**: I Made overlapping and non-overlapping N length sequences/clips from those frames and tried out with batch size 128. Taken N to be 48.
- In my implementation, each batch dimension was (128,20,2,48)
- **DDPM:** This class implements a Diffusion Probabilistic Model (DDPM) for learning a latent representation of a dataset. It takes in a U_net argument, an instance of *punet* used to model the noise process. The class provides methods for computing the mean and variance of the latent representation at each diffusion step, as well as for computing the MSE loss function used to train the model.
- **qt_q0:** Simple function taking x_0 and outputting x_t by the formula using alpha_bars and betas. **mu_sigma_xt** helps qt_q0 by giving them calculated mean and variance for finding x_t.

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

- **Train:** trains the Unet model used to predict noise. Optimises by considering the loss between predicted and actual noise at sampled *t* timestamps randomly. Uses *qt_q0* to directly obtain *qt* instead of recursing.

Observed loss values for a batch for 300 epochs, N=48, BS=128



❖ Sampling Methodology

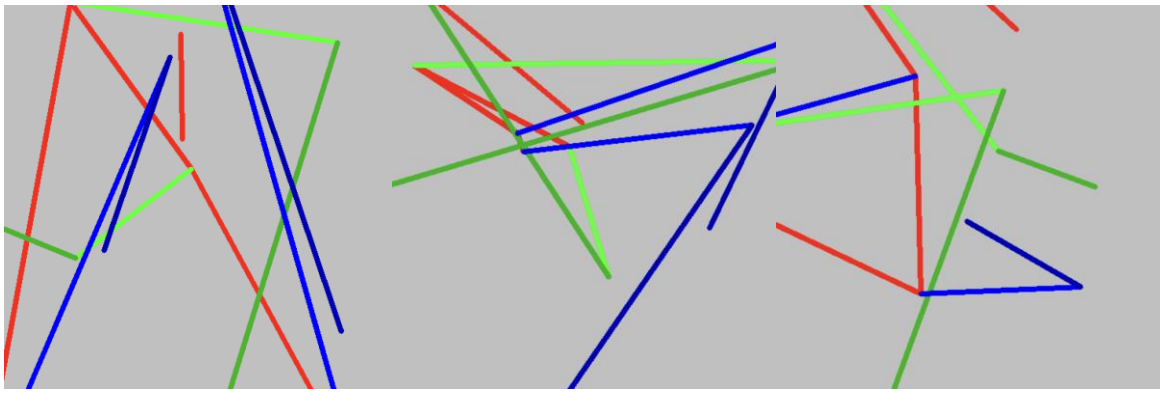
- ***xt_1_xt:*** method which gives *x_t-1* given *x_t* and *t with the help of the UNet model which gives the noise given x_t* and *t*.
- xt_1_xt then uses that noise and subtracts from *x_t* with some coefficients and adds some variance defined properly using alphas and betas.

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$

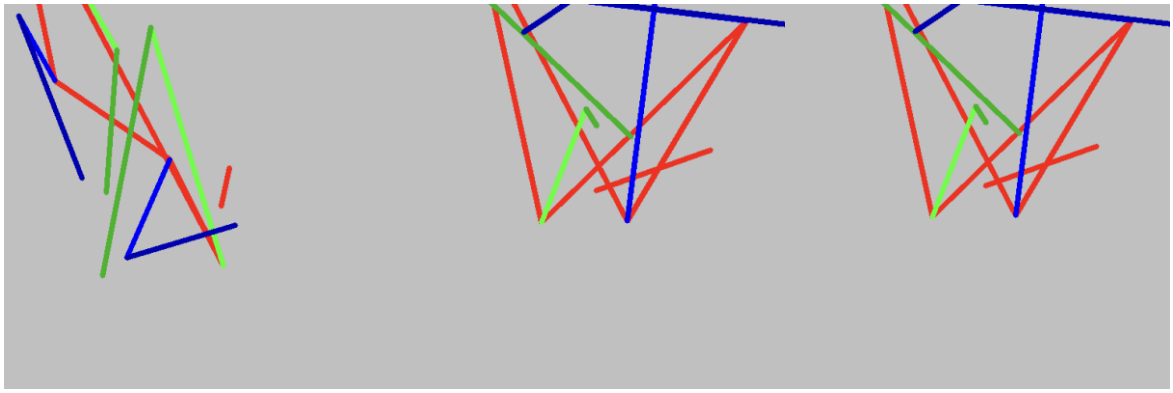
- I took 1000 steps to get a generated sequence from a standard gaussian noise. I.e. Given *x_1000* as *N (0, I),* find *x_0* (new clip).
- Generated clip from the 2D points set using renderer.py.
- As can be observed below, as we increase the epochs, the joints become more realistic, but we could not observe any dance clip.

❖ Generated Clips

Epochs: 10, N=48, BS= 128



Epochs: 50, N=48, BS= 128



Epochs: 300, N=48, BS= 128