  
  
  
Fig. 2: DiPPeR - Image Conditioned Diffusion Training Pipeline: A Map Image Observations sample O is fed to the ResNet18 Visual Encoder and converted to latent embeddings o. The x and y of the start and goal positions are also added as part of O. Noise ϵk sampled from the prior Gaussian Distribution is added to the trajectory instance At. The noisy sample is passed as an input to the diffusion network ϵθ and is conditioned by O. The network ϵθ takes the form of a CNN and it outputs the denoised action A0.  
  
  
**### A. Data generation**  
Creating the training dataset includes generating random map images and feasible 2D trajectories.  
  
## 1) Map generation:  
Map generation is done through Kruskal’s Minimum Spanning Tree (MST) Algorithm. ensuring that the graph remains acyclic.  
  
## 2) Path generation:  
To provide trajectories in the training framework, the generate paths need to be feasible - avoid all obstacles. To generate feasible paths, we use the A∗ path finding algorithm. randomize the position of the start and goal node to generate a variety of trajectories per map.  
  
  
**### B. Training Framework**  
  
## Visual Encoder:  
A ResNet-18 visual encoder with special SoftMax pooling is trained end-to-end to convert the observation image O to a latent embedding o while preserving spatial information. The ResNet is trained alongside ϵθ.  
  
## DDPM Training:  
- The observation space `O` comprises 100x100 pixel images representing randomly generated maps.  
- The action space `At` comprises 2D trajectories for each map, where `t` refers to a sequence of timesteps, and `T` represents the training horizon.  
- DDPM is used to approximate the conditional distribution `p(At | O)` of the action vector `At` given the map image observation `O`.  
 - This formulation speeds up the diffusion process and improves the generated actions by predicting trajectories conditioned on the specific map image.  
 - DDPM takes as input a noisy action vector `Akt` and performs `k` denoising iterations through gradient descent, following Equation 3, to obtain the noise-free representation `A0t`.  
- Ak−1 t = α(Ak t − γϵθ(O, Ak t , k) +N(0, σ2I)) (Equation 3)  
- The noise prediction network `εθ` is trained to predict the noise added to a random sample `A0\_t`, with the aim of minimizing the mean squared error between the predicted noise and the actual noise  
- loss function: L = MSE(ϵk, ϵθ(O, A0 t + ϵk, k)) (Equation 4)  
- the noise schedule that drives the learning process, in this case, the Square Cosine Schedule is used  
- The hyperparameters `α`, `γ`, `σ` determine the scheduling learning rate, which controls the extent to which the diffusion policy captures high and low-frequency characteristics of action signals.  
- A 1D temporal CNN is used to condition the action generation on the observations `p(At | O)` through Feature-wise Linear Modulation (FiLM).  
- Path Length Determination: A variable `pathl` defines the number of noisy samples used during the denoising diffusion process. The value of `pathl` is determined as a function of the approximate length of the estimated trajectory. If the start and goal positions are further apart, `pathl` will be larger, and vice versa.  
- Noise Vector Construction: A noise vector is constructed with a length equal to `pathl` and two dimensions (x and y pixel coordinates), sampled from the prior Gaussian distribution. This noise vector is conditioned using inpainting, where the first and last columns are set to the start and goal positions, respectively.  
- Denoising Diffusion Process: The constructed noise vector, along with the map image observation `O`, is fed into the trained DiPPeR model. The reverse chain of the DDPM is used to iteratively denoise the input noise vector.  
- Output: The output of the denoising process is the final trajectory `A0`, which connects the start and goal points while aiming to follow a feasible path through the map, avoiding obstacles.

I have folder named maze containing Images I have a dataset file named tajectories.npy containing the map name, start point and goal point coordinates, or pixels, and trajectory.  
Now I want to write code for trajectory generation using the Denoising Diffusion Probability Model (DDPM), a generative model.  
  
First, write a function named Resnet\_Visual\_Encoder that loads all images from a folder named maze, which is located in the current directory. now this function passes all images through Resnet18 Visual Encoder. This Resnet-18 visual encoder should not be pretrained and Global max pooling layer should be replaced by Softmax pooling layer. This function returns the latent embading of those images.  
  
In the second step, write another function named Pathl, which takes the data from trajecrories.npy file of map\_name start point, goal point, and trajectory. and this function defines and returns the lengths of trajectories. In a further step, create another function named adding\_Gaussian\_noise that takes input from the Pathl function and dateset and creates noisy trajectories by using the cosine noise scheduler for the CNN-based DDPM diffuser. A noise vector is constructed with a length equal to `pathl` and two dimensions (x and y pixel coordinates), sampled from the prior Gaussian distribution.  
  
in next step, create a function named noise\_prediction\_network, which takes input as a Latent embedding of images from Resnet\_Visual\_Encoder function and starts and goal point details from dataset, as well as noisy trajectory from adding\_Gaussian\_noise function. this noise predection network should train to predict the noise added to sample, with the aim of minimizing the mean squared error between the predicted noise and the actual noise.  
  
now create a DiPPeR model: The constructed noise vector, along with the map image observation `O`, is fed into the trained DiPPeR model. The reverse chain of the DDPM is used to iteratively denoise the input noise vector. At the end the output of the denoising process is the final trajectory `A0`, which connects the start and goal points while aiming to follow a feasible path through the map, avoiding obstacles.