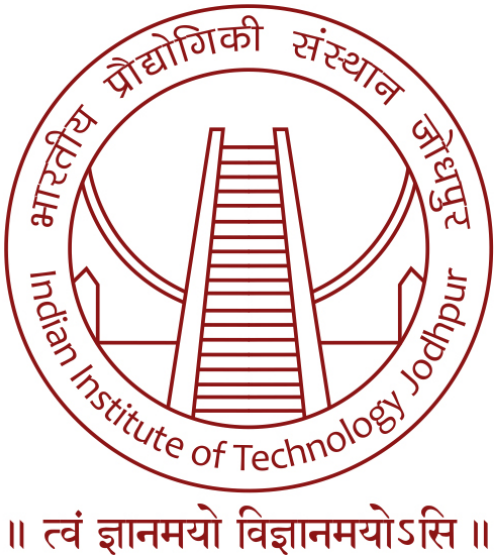
Unsupervised Deep Learning for Denoising Images

Boosting denoisers with reinforcement learning for image restoration

B.Tech Project Report

Submitted by
Aryan Tiwari (B20AI056)
Ayush Abrol (B20AI052)

Under the Supervision of
Dr. Gaurav Harit



Department of Computer Science Engineering
Indian Institute of Technology, Jodhpur

Contents

1 Abstract **3**

2 Introduction **3**
2.1 Objective . . . 3
2.2 Problem Statement . . . 4
2.3 Background Research . . . 5

3 Work Done **5**
3.1 Theory and Methodology . . . 5
3.1.1 POMDP . . . 5
3.1.2 MARL . . . 6
3.1.3 Network Proposals . . . 7
3.1.4 ToolBox . . . 8
3.1.5 Rewards . . . 8
3.1.6 TD3 proposal . . . 9
3.1.7 Final Algorithm Proposed . . . 9
3.2 Implementation . . . 10
3.2.1 Datasets and Setup Configuration . . . 10
3.2.2 Batch-wise Dataloader . . . 11
3.2.3 State Class . . . 13
3.2.4 Policy and Value Networks using Dilated Convolutions . . . 13
3.2.5 PixelWise Asynchronous Advantage Actor Critic Class . . . 14
3.2.6 Train-Test Loop and Hyperparameters . . . 15
3.2.7 Interpretability and Explainability using Agent Action Maps . . . 16

4 Results and Discussion **18**
4.1 Episode-wise training results on BSD-68 Dataset . . . 18
4.2 Testing results for 5 episodes and sigma=15 . . . 18
4.3 Results on custom images to check for multiple noises . . . 21
4.4 Interpretable Results using Agent Action maps . . . 22

5 Conclusion **24**

References **25**

Acknowledgement

We extend our sincere gratitude to **Dr.Gaurav Harit** for his invaluable guidance, unwavering support, and expertise throughout this project. His insightful feedback and encouragement have been instrumental in shaping our ideas and refining our approach.

We would like to express our appreciation to our partners, for their collaborative spirit and dedication to the project. Working together has been a rewarding experience, and their contributions have significantly enriched the overall quality of our work.

We also want to acknowledge the support and resources provided by *Computer Science Department and ANVISA Lab of Indian Institute of Technology, Jodhpur*, which have played a crucial role in the successful completion of our project.

Finally, we extend our thanks to all those who have shared their knowledge and expertise in the field of Reinforcement Learning, computer vision, and Deep Learning technology. Your insights have been instrumental in shaping the development of our Denoising Reinforcement-based Model.

This project has been a challenging yet fulfilling journey, and we are grateful to everyone who has been a part of it.

1 Abstract

Learning-oriented techniques for image restoration typically involve teaching systems how to convert flawed images into clean ones. Many existing methods concentrate on creating diverse deep neural network structures and innovative loss functions to address multiple mixed distortions with unknown proportions. Despite their success in restoring images, these methods demand costly training data and generate outcomes that lack interpretability. Our study introduces a novel approach utilizing deep reinforcement learning (DRL) to repair distorted images. We frame the image restoration challenge as a Partially Observable Markov Decision Process (POMDP), where actions correspond to various pixel-level image denoising operations. In our method, individual agents handle pixels, learning to adjust their respective values by selecting appropriate combinations of actions. Additionally, we introduce an innovative exploration strategy that ensures similar actions hold similar value, preventing overfitting during estimation of state-action values. Through comprehensive experiments, we demonstrate our method's efficacy in restoring images with multiple combined distortions, showcasing comparable or superior performance to prior learning-based approaches. By visualizing the weighting process of multiple pixel-level operations, we can discern the operation combinations employed for each pixel at each stage. This advancement represents a stride toward enhancing the explainability and interpretability of learning-based image restoration methods. Here is a link to the GitHub repository of the implemented code: **GitHub Repository**

2 Introduction

2.1 Objective

This Bachelor of Technology project aims to implement a novel method for image denoising using deep reinforcement learning, which casts the image denoising problem as a Partially Observable Markov Decision Process (POMDP) and allows for multiple pixel-wise image denoising operations. We aim to demonstrate the effectiveness of our approach through extensive experiments and show that it performs comparably or better than previous learning-based approaches. Additionally, We aim to contribute to the explainability and interpretability of learning-based image denoising methods by visualizing the process of weighting multiple pixel-wise operations.

The specific goals of the project are outlined as follows:

- **Reconceptualizing Restoration as a Multi-Agent Reinforcement Learning Problem**: The project redefines the challenge of multi-noise image restoration by framing it as a Multi-Agent Reinforcement Learning (MARL) problem under Partially Observable Markov Decision Process (POMDP), providing a novel perspective to address image restoration.

- **Continuous Action Space for Pixel-Wise Restoration**: Introduces a groundbreaking methodology that operates in a continuous action space for pixel-wise image restoration. This approach sidesteps the limitations of discrete action space methods, allowing for more precise and nuanced pixel-level adjustments.

- **Integrated Denoising Methodology with Traditional Filters**: Proposes an integrated denoising method that concurrently applies multiple traditional image filters to the noisy image. Then, a deep reinforcement learning algorithm learns the optimal weight synthesis for fusing these filtered images into a clearer restoration, enhancing restoration effectiveness.

- **Addressing Challenges of State Information in POMDP**: Tackles the challenge of insufficient state information in POMDP frameworks, offering solutions specifically addressing the deterministic policy gradient method's limitations within the continuous action space.

- **Toolbox and Agent Framework: The proposed method comprises two main components**: a toolbox housing various traditional image filters and an agent dynamically selecting actions to modify pixel weights for these filters. This dual-framework design aims to achieve superior restoration results by leveraging the strengths of both components.

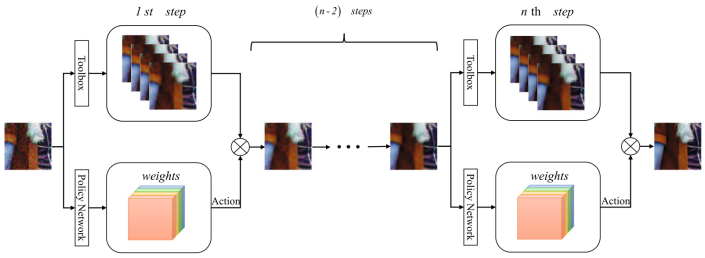


Figure 1: Illustration of the Approach

By successfully accomplishing these goals, the project makes a significant contribution to the development of easily interpretable and universal Image Denoising technology, opening the door for wider application in a variety of contexts, including academic and professional settings.

2.2 Problem Statement

A human expert removes multiple combined distortions by applying a set of image denoising operations. To imitate this process, we formulate image denoising as a problem of finding an optimal operation combination of denoising actions.

Let I_t^i be the i-th pixel of the modified image $I_t \in R^{H\times W\times C}$ that has N pixels ($i = 1, \cdots, N$). Here, $N := H \cdot W, I_0$ is denoted as the original distorted image, H, W and C are its height, width and the number of channels, respectively.Since different areas of the image may be distorted by multiple noises, it is necessary to restore the image at the pixel level. Each pixel corresponds to an agent, each agent $a \in A \equiv \{1, \cdots, N\}$ receives the local observation $o_t^a \in \mathcal{O}$ provided by an observation function $O(I_t, a)$ and takes an action $u_t^a \in U$ according to a stationary policy $\pi^a(u_t^a \mid o_t^a)$. Here, the action $\mathbf{u}_t^a \in R^M$ denotes the attention weights on the pixel-wise outputs of the toolbox containing M parallel image denoising operations, U is a probability simplex. After adjusting its corresponding pixel value I_t^a, each agent obtains a reward r_t^a that measures how much the modified pixel value I_{t+1}^a has improved compared to the previous one. Given the input image I_t and the joint action $\mathbf{u}_t := \left[u_t^1, \ldots, u_t^N\right]$ at time step t, the environment change to the next state I_{t+1} according to the state transition probability $P(I(t+1) \mid I(t), \mathbf{u}_t)$. All agents work together to enhance the image in an iterative way, and terminate this process when the maximum time step T is achieved.

The goal of the RL-based image denoising problem is to learn the optimal joint policies $\boldsymbol{B} = \left(\pi^1, \ldots, \pi^N\right)$ that maximize the mean of the total expected rewards at all pixels:

$$\max_{\mathbf{B}} E_{\tau \sim p_{\boldsymbol{B}}(\tau)} \left[\sum_{t=0}^{T} \gamma^t \frac{1}{N} \sum_{a=1}^{N} r_t^a \right]$$
$$\text{s.t.} \sum_{k=1}^{M} u_t^{a,k} = 1, \quad u_t^a \sim \pi^a \left(u_t^a \mid o_t^a \right), \quad a = 1, \ldots, N.$$

where γ is the discounted factor, the trajectory $\tau := \left\{ o_0^1, u_0^1, \ldots, o_0^N, u_0^N, \ldots, o_T^1, u_T^1, \ldots, o_T^N, u_T^N \right\}$.the induced trajectory distribution $p_\pi(\tau)$ is given by

$$p_\pi(\tau) = p\left(I_0\right) \prod_{t=0}^{T} \left(\prod_{a=1}^{N} \pi \left(u_t^a \mid o_t \right) \right) P\left(I(t+1) \mid I(t), \mathbf{u}_t \right),$$

The common approach is to divide this decision-making problem into N independent subproblems and train N networks, where the i-th policy learns to maximize the expected discounted cumulative rewards at the i-th pixel $J\left(\pi^i\right) = E_{\tau \sim p_{\boldsymbol{B}}(\tau)} \left[\sum_{t=0}^{T} \gamma^t r_t^i \right]$. However, this method is not suitable for situations where the size of an input image in the test is different from the one in the training, and it becomes computationally impractical as the number of agents increases to thousands. Moreover, since the pixel-wise outputs of the toolbox are invisible to each agent, the policy $\pi\left(\cdot \mid o_t\right)$ leads to the poor performance of image restoration in this partially observable setting.

2.3 Background Research

Ensemble learning, as discussed by **Polikar(2012)**, offers insights into combining weak classifiers to form a strong one through boosting. This concept applies to image restoration, where multiple algorithms with weak restoration capabilities can merge into a single, powerful algorithm. Recent studies explore using deep reinforcement learning for this purpose, with **Yu et al. (2018)** being the pioneers. They introduced RL-Restore, realizing that the sequence in which denoising methods are applied significantly impacts the final restoration quality for contaminated images. This led them to frame the problem as an MDP (Markov Decision Process) and create a toolbox of simple denoising networks, employing deep reinforcement learning to determine the best order for these networks. Their approach demonstrated superior restoration compared to large-scale neural networks while using fewer parameters. However, **Suganuma et al. (2019)** found that although RL-Restore showed good restoration, it significantly reduced recognition accuracy in subsequent tasks—a drawback that shouldn't occur. **Xie et al. (2019)** further elucidated why neural network-based image denoising leads to decreased recognition accuracy.

3 Work Done

3.1 Theory and Methodology

3.1.1 POMDP

The true environment is often not entirely visible to an agent, restricting its observation to a limited state. A Partially Observable Markov Decision Process (POMDP) represents this scenario, where decision-making relies on incomplete environmental information. POMDPs are challenging to solve due to the computational complexity arising from incomplete observations.

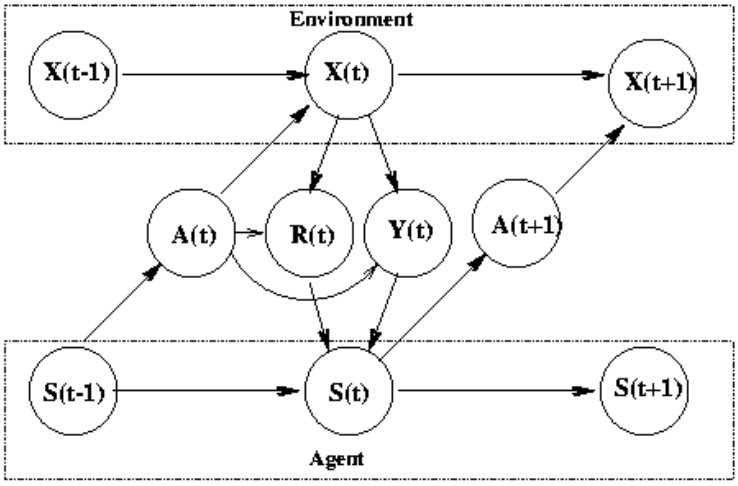


Figure 2: Partially Observable Markov Decision Process (POMDP): State S(t), Environment X(t), Observation Y(t), Reward R(t), Action A(t)

Value iteration, a method proposed by White and Scherer in 1989, approximates POMDP solutions, but it escalates the problem's complexity exponentially, potentially leading to overwhelming dimensions. To mitigate this, approaches like those by Koller and Parr in 2013, along with Guestrin et al. in 2001, break down the problem into smaller components, reducing its overall scale. Additionally, recent advancements involve leveraging learning-based algorithms (as highlighted by Bertsekas and Tsitsiklis in 1995, and Lin and Mitchell in 1992) to achieve favorable outcomes. Particularly, the advent of Recurrent Neural Networks (RNNs) enables agents to base decisions on historical data, presenting a promising development in addressing these challenges.

3.1.2 MARL

Multi-Agent Reinforcement Learning (MARL), introduced by Tan in 1993, stands as a significant component within multi-agent systems. Unlike single-agent reinforcement learning, MARL involves multiple agents, each impacted not only by the environment but also by the actions of other agents. It addresses the challenge of decision-making among multiple agents within a shared environment, where each agent must engage with both the environment and fellow agents to maximize rewards (Lowe et al., 2017).

Compared to single-agent systems, multi-agent setups bring forth distinct characteristics: firstly, the state transitions within such systems hinge upon actions taken by all agents involved. Secondly, the rewards an agent receives are contingent not only on its own actions but also on those of other agents. Consequently, solving problems within multi-agent systems becomes notably intricate and challenging. Broadly, multi-agent reinforcement learning algorithms fall into three categories: full cooperation, full competition, and hybrids designed for specific applications (Yang et al., 2018).

Various fundamental algorithms exist for tackling multi-agent reinforcement learning, such as MiniMax-Q (Littman, 1994), NashQ (Singsanga et al., 2010), FFQ (Littman, 2001), WoLF-PHC (Bowling and Veloso, 2001), alongside mainstream methods like MADDPG (Rashid et al., 2018), QMIX, MFMRL (Bu¸soniu et al., 2010), and others.

3.1.3 Network Proposals

In RL based pixel-wise image restoration, the input information o_t^i each agent i receives at step t consists of the pixel value I_t^i and its neighborhood pixels provided by a observation function $O\left(I_t, i\right)$, based on which the corresponding policy performs inference. The field-of-view of the observation function has an important influence on restoring images, the small field contains little useful information, but the large field will include redundant observation that is useless to the i-th agent, leading to high computational burden. Rather than designing the observation function in a hand-crafted way, we use convolutional blocks to provide the agent with its neighborhood information. Another advantage of convolutional blocks is that all the N agents can share their parameter, leading to the high-efficient computation.

Partial observability arises from two sources including a restricted field-of-view and the invisibility of the output of the toolbox to all agents. Each agent i should learn to form memories based on interactions with the environment to handle partially observed problems, thus the optimal policy of agents in principle require to access to the historical experience $h_t = \{I_0, \mathbf{u}_0, \ldots, I_{t-1}, \mathbf{u}_{t-1}, I_t\}$. Here, we use Gate Recurrent Unit (GRU) network to effectively extract this historical information in their recurrent state, which is given by

$h_t = \mathrm{GRU}\left(h_{t-1}, I_t, \mathbf{u}_{t-1}\right),$

where h_{-1} and $\mathbf{u}_{-1}$ are the zero start state. The attention weights $\mathbf{u}_t \in R^{N\times M}$ on the pixel-wise output of the toolbox are calculated as follows

$$u_t^{i,m} = \frac{\exp\left(\mathbf{z}_t^{i,m}\right)}{\sum_{k=1}^{M}\exp\left(\mathbf{z}_t^{i,k}\right)},$$

$$\mathbf{z}_t = \mathcal{F}\left(h_t\right), \quad i = 1, \ldots, N$$

where $\mathcal{F}(\cdot)$ is the convolution operator. The modified image at step $t+1$ is given by

$$I_{t+1}^i = \sum_{m=1}^{M} u_t^{i,m}\bar{I}_t^{m,i}, \quad \bar{I}_t^m = g_m\left(I_t\right) \in R^{N\times C},$$

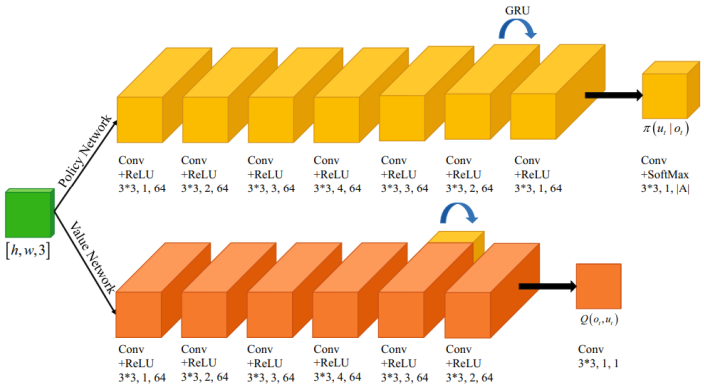


Figure 3: Architecture of Policy and Value Network

where $g_m(\cdot)$ is the m-th image denoising operation in the toolbox. Therefore the entire architecture of the policy network in Fig. 3 includes three modules: it uses convolutional blocks to learn low-level features. The GRU block combines these features with historical information extracted from past experience to learn high-level features, based on which all agent make decisions. A major challenge of training deterministic

policies in image restoration is exploration. One choice to improve the exploration ability of policies is to construct an exploration policy, which is represented by a Gaussian noise source and a deterministic neural network that transform a draw from that noise source, i.e., $\mathbf{u}_t = \boldsymbol{B}_\phi(h_t, \varepsilon)$ with $\varepsilon \sim \mathcal{N}\left(0, \delta^2\right)$. Specially, the modified action of the i-th agent is given by

$$\tilde{u}^{i,m} = \frac{\exp\left(\mathbf{z}^{i,m} + \varepsilon^m\right)}{\sum_{k=1}^{M} \exp\left(\mathbf{z}^{i,k} + \varepsilon^k\right)}$$
$$\varepsilon_k \sim \text{clip}\left(\mathcal{N}\left(0, \sigma^2\right), -c, c\right).$$

where the added noise is clipped to keep the modified action close to the original one. Further, we can easily obtain that

$\frac{\exp\left(\mathbf{z}^{i,m}-c\right)}{\sum_{k=1}^{M}\exp(\mathbf{z}^{i,k}+c)} < \tilde{u}^{i,m} < \frac{\exp\left(\mathbf{z}^{i,m}+c\right)}{\sum_{k=1}^{M}\exp(\mathbf{z}^{i,k}-c)}$,

then the modified action $\tilde{u}^{i,m}$ is a random variable with support in $\left(\exp(-2c)u^{i,m}, \exp(2c)u^{i,m}\right)$.

3.1.4 ToolBox

In image restoration with reinforcement learning, a toolbox refers to a collection of techniques, algorithms, and methodologies used to address image restoration problems using reinforcement learning (RL) approaches. Our project also uses a toolbox that contains multiple traditional filters for image denoising, Namely:

Table 1: Tools in toolbox

Tools	Filter size	Parameters
Gaussian filter	$5 * 5$	$\sigma = 0.5$
Gaussian filter	$5 * 5$	$\sigma = 1.5$
Bilateral filter	$5 * 5$	$\sigma_c = 0.1, \sigma_s = 5$
Bilateral filter	$5 * 5$	$\sigma_c = 1.0, \sigma_s = 5$
Median filter	$5 * 5$	-
Box filter	$5 * 5$	-

Each of the filter is associated with a number and colour, in order for us to implement agent action map, which will help us visualize what filter has been used by each agent(pixel) at each time step.

3.1.5 Rewards

The goal of reinforcement learning is to obtain the largest cumulative reward, and the reward determines the quality of the policy adopted by the agent. In this paper, the image quality of each step is used to determine the reward. The reward is defined by $r_t = P_{t+1} - P_t$, where P_{t+1} is the peak signal-tonoise ratio (PSNR) value between the image processed in the t th step and the real image. Therefore, the cumulative reward is defined as $R_{ij}|_{i\in(0,h),j\in(0,w)} = \sum_{t=0}^{T-1} r_t = P_T - P_0$, which indicates that the cumulative reward is related to the PSNR value of the last processed image and the PSNR value of the initial state image.

3.1.6 TD3 proposal

Similar to TD3 algorithm, we use parameterized function approximators for both the Q-function Q_θ and policy π_ϕ, and then alternatively performs policy evaluation and policy improvement.

$$J_Q(\theta_s) = E\left[N^{-1}\sum_{i=1}^{N}\left(Q_{\theta_s}\left(h_t^i, u_t^i\right)\big|_{u_t^i=\pi_\phi\left(h_t^i\right)} - y_t^i\right)^2\right]$$

$$y_t^i = r_t^i + \gamma \min_{s=1,2} Q_{\bar{\theta}_s}\left(h_{t+1}^i, \pi_{\bar{\phi}}\left(h_{t+1}^i, \varepsilon\right)\right)$$

$$\nabla_\phi J(\phi) = E\left[N^{-1}\sum_{i=1}^{N}\nabla_{u_t^i} Q_{\theta_1}\left(h_t^i, u_t^i\right)\Big|_{u_t^i=\pi_\phi\left(h_t^i\right)} \nabla_\phi \pi_\phi\left(h_t^i\right)\right].$$

where $\bar{\phi}$ and $\bar{\theta}_s, s = 1, 2$ are delayed parameters, and fitting the value of the modified action can alleviate the narrow peak of overfitting to the value estimation, decreasing the variance of the target Q. The pseudo code of our algorithm is shown in the Algorithm 1.

3.1.7 Final Algorithm Proposed

Based on above theorization and proposed methodology, we have the following final algorithm for Image Denoising:

Algorithm 1

Initialize critic networks Q_{θ_1}, Q_{θ_2} and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$, $\bar{\phi} \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for episodes=1, M **do**
  Initialize empty history h_0 and action u_0.
  Receive the orignal distored image I_0.
  for t = 1, T **do**
    $h_t \leftarrow I_t,\ \mathbf{u}_{t-1},\ h_{t-1}$
    Select raw action with exploration noise $\mathbf{z}_t = \pi_\phi(h_t) + \varepsilon, \quad \varepsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.
    Make the $action \in (0,\ 1)$, $\mathbf{u}_t = \text{softmax}(\mathbf{z}_t)$.
  end for
  Store transition sequence $(I_0,\ \mathbf{u}_1,\ r_1,\ I_1,\ \cdots,\ I_T)$ in $\mathcal{B}$.
  Sample a mini-batch of N episodes $I_0^i,\ \mathbf{u}_1^i,\ r_1^i,\ I_1^i,\ \cdots,\ I_T^i$ from $\mathcal{B}$.
  Construct histories $h_t^i = \left(o_1^i,\ u_1^i,\ \cdots,\ u_{t-1}^i,\ o_t^i\right)$.
  Compute target values for each sample episode $y_1^i,\ \cdots,\ y_T^i$ using recurrent target networks
  $y_t^i = r_t^i + \gamma min_{s=1,2} Q_{\theta_s'}\left(h_{t+1}^i, \pi_{\bar{\phi}}\left(h_{t+1}^i\right)\right)$
  Update critics $\theta_{s=1,2} \leftarrow min_{\theta_s} N^{-1}\sum\left(y_t^i - Q_{\theta_s}\left(h_t^i,\ u_t^i\right)|_{u_t^i=\pi_\phi\left(h_t^i\right)}\right)$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1}\sum \nabla_a Q_{\theta_1}\left(h_t^i,\ u_t^i\right)|_{u_t^i=\pi_\phi\left(h_t^i\right)} \nabla_\phi \pi_\phi(h_t^i)$
  end if
end for

3.2 Implementation

3.2.1 Datasets and Setup Configuration

- **Datasets**
  - We have used the dataset—BSD68 dataset, and preprocess the dataset in the following operation. Firstly, the dataset is downsampled, and then the sampled image is divided into 63*63 sub-images. In our project, 3584 images were generated for the dataset as the ground truth of the training data. The two most common noises in the original images are Gaussian noise and Poisson noise. Therefore, we randomly added Gaussian noise and Poisson noise in random proportion to the ground truth dataset to form the noise dataset. This operation ensures the authenticity of the training data since the ratio of Gaussian to Poisson noise added to each image is random.
  - With a total of 30000 episodes, the dataset is visualized at 300 episode intervals to compare the PSNR scores episode-wise while training our networks.
  - We evaluate that how well the method performs at different stages of restoring images using this dataset. It also compares this method with others using the same dataset, using metrics like PSNR and SSIM to see how it measures up. Essentially, this dataset is like a standard to see if this new deep reinforcement learning-based method really works well for image restoration.
  - The testing set of the dataset includes a total of 68 images for which we test our policy and value networks trained and plot the agent action map of all the testing images.

Figure 4: BSD-68 Dataset

- **Setup Configuration**
  - To start the project, we had to setup the configuration of the project. We included the following Python Frameworks and other tools for the same:
  - For training purposes, we used NVIDIA RTX A4000 GPU with 16 GB RAM present in Anvisa Lab, IIT Jodhpur. The NVIDIA RTX A4000 is a professional graphics card designed for design and visualization work. It is built on the NVIDIA Ampere architecture and combines 48 second-generation RT Cores, 192 third-generation Tensor Cores, and 6144 CUDA cores with 16 GB of GDDR6 memory. The card is known for its real-time ray tracing, AI-accelerated compute, and high-performance graphics capabilities. It is the most powerful single-slot GPU for professionals, making it suitable for a range of professional applications, including real-time 3D rendering, video editing, and computer-aided design (CAD) work.

Table 1: List Python Frameworks and Other Tools used

Framework or Tool	**Version**
Pandas	2.14
Numpy	1.26.2
Matplotlib	3.8.2
CUDA Toolkit	12.1
PyTorch	2.1.2+cu121
Torchvision	0.16.2+cu121
OpenCV-python	4.8.1.78
Chainer	5.0.0
Chainerrl	0.5.0
Pillow	10.1.0
Cupy-cuda12x	12.3.0
Protobuf	3.20.0

3.2.2 Batch-wise Dataloader

- We defined a MiniBatchLoader Class which primarily loads data, provides necessary augmentation and handle single image inference as well. It relies heavily on OpenCV (cv2) for image processing tasks like reading, resizing, and augmentation.
- It can convert RGB images to Grayscale internally and then the batch of images can be used for further processing.
- **Initialization**
  - The constructor initializes the MiniBatchLoader object with paths to training and testing data, along with an image directory path and a crop size.
  - We decided to set the size of training images (crop size) to be 1x70x70 but later testing size was set to be 1x248x248 to check the generalizability of the trained networks and to check whether the agents adapt to the continous learning space or not.

Figure 5: A sample image from BSD-68 Dataset

- **Static Methods**
  - **path_label_generator**: Generates pairs of file paths by reading a text file (with paths) and joining them with a specified source path.
  - **count_paths**: Counts the number of paths in a given file.
  - **read_paths**: Uses path_label_generator to read paths from a text file and source path, returning a list of paths.
- **Loading Data**
  - **load_training_data** and **load_testing_data** methods take indices and call the load_data method with the respective dataset information (training/testing).
  - Loads images using OpenCV (cv2) based on provided indices and path information.
  - If augment is set to True, it performs data augmentation (random flipping, rotation) on the images and resizes them to a specified crop size.
  - Images are normalized (scaled between 0 and 1) and stored in an array (xs) structured to fit a mini-batch.
- **Conditions**
  - **augment=True**: When augmenting data for training.

- **mini_batch_size == 1**: For single image inference.
- Otherwise, it raises an error if the mini-batch size is not 1 during inference.

3.2.3 State Class

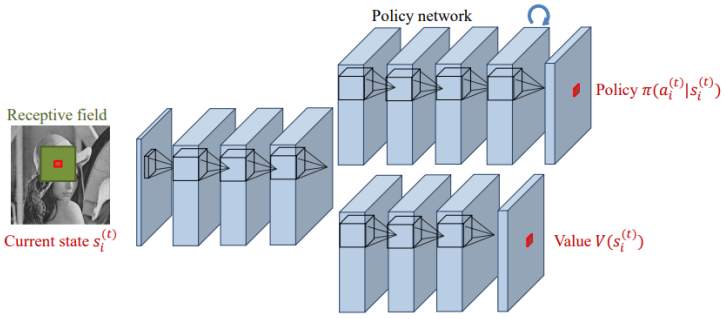
- This class represents an environment state that manages image manipulation based on specified actions.
- The class provides methods to reset the state, set a new image, and update the state based on actions taken in the environment.
- **Initialization** - Initializes a State object with an image of zeros with a specific size and a move range value.
- **reset**
  - Resets the state with a new image 'x' added to a value 'n'.
  - Constructs a tensor by concatenating the current image ('self.image') with a zero-filled array ('prev_state') along the second axis.
- **set**
  - Sets the state's image to a new input 'x'.
  - Updates the tensor's initial section to reflect the new image.
- **step**
  - Takes an action 'act' and an inner state.
  - Processes the action and manipulates the current image based on the action taken:
    * Scales the action values to a range and applies the movement to the image.
    * Applies various image manipulation filters (like Gaussian blur, bilateral filter, median blur, etc.) to different regions of the image based on the action taken ('act' values).
    * Updates the current image based on the actions performed using the filters.
    * Updates the tensor with the new modified image and inner state.

3.2.4 Policy and Value Networks using Dilated Convolutions

- This class represents a neural network architecture with convolutional, dilated convolutional, and Gate-Recurrent Unit (GRU) connections used for both policy and value estimation in reinforcement learning tasks. It computes both the policy (actions to take) and the value (expected future rewards) given an input `x`.
- **Constructor (`__init__`) method:**
  - It initializes the model and sets up its layers.
  - `n_actions` is the number of actions the model can output.
  - The model loads weights from a Caffe model (`CaffeFunction`) to initialize certain layers.
  - It creates multiple layers including:
    * Convolutional layers (`L.Convolution2D`)
    * `DilatedConvBlock` instances (`DilatedConvBlock`)

* Gate-Recurrent Unit (GRU) connections using convolutions (`conv7_*` connections).
* A softmax policy layer (`conv8_pi`) for outputting action probabilities.
* Additional dilated convolutional layers (`diconv5_V`, `diconv6_V`) and a final convolutional layer (`conv7_V`) for value estimation.

- **`pi_and_v` method:**
  - This method defines the forward pass of the model.
  - It takes input `x`, applies a series of operations:
    * Initial convolution (`conv1`) followed by ReLU activation.
    * Multiple dilated convolutions (`diconv2`, `diconv3`, `diconv4`, `diconv5_pi`, `diconv6_pi`).
    * Calculation of intermediate variables (`x_t`, `h_t1`, `z_t`, `r_t`, `h_tilde_t`, `h_t`) using convolutions and sigmoid/tanh activations.
    * Output calculation for policy (`pout`) using `conv8_pi`.
    * Value estimation (`vout`) using dilated convolutional layers (`diconv5_V`, `diconv6_V`) and a final convolution (`conv7_V`).



Shared network				Policy network			
Conv+ReLU	Conv+ReLU	Conv+ReLU	Conv+ReLU	Conv+ReLU	Conv+ReLU	ConvGRU	Conv +Softmax
3x3, 1, 64	3x3, 2, 64	3x3, 3, 64	3x3, 4, 64	3x3, 3, 64	3x3, 2, 64	3x3, 1, 64	3x3, 1, $\|A\|$

Value network		
Conv+ReLU	Conv+ReLU	Conv
3x3, 3, 64	3x3, 2, 64	3x3, 1, 1

Figure 6: Network Architecture of Policy and Value Networks

3.2.5 PixelWise Asynchronous Advantage Actor Critic Class

- This class represents an A3C agent capable of asynchronous training and acting, gathering experiences, updating models, and handling episode terminations in reinforcement learning scenarios.
- **Initialization:**
  - It initializes the A3C agent with various hyperparameters like learning rate, discount factor, and others.

  - Sets up two models: `shared_model` (global) and `model` (thread-specific) using `deepcopy`.
  - Initializes optimizer and other variables used for tracking past actions, rewards, and states.

- `update` **Method:**
  - Updates the model based on accumulated experiences (rewards, actions, values).
  - Computes policy and value losses (`pi_loss`, `v_loss`), adjusts them based on coefficients and normalization.
  - Performs backpropagation, updates the shared model, and synchronizes parameters.

- `act_and_train` **Method:**
  - Performs an action in the environment, collecting experiences.
  - Updates the model when a certain number of experiences (`t_max`) is reached.
  - Samples an action based on policy (`pout`) and collects action log probabilities and entropy.
  - Updates internal counters and returns the selected action and inner state.

- `act` **Method:**
  - Similar to `act_and_train` but used for acting without training.
  - Samples an action based on policy (`pout`) and returns it along with the inner state.

- **Episode Handling:**
  - `stop_episode_and_train` and `stop_episode` manage episode endings, resetting states if necessary.

- **Other Methods:**
  - Includes methods for loading models and gathering statistics.

3.2.6 Train-Test Loop and Hyperparameters

- We initially instantiated a mini_batch_loader object from the MiniBatchLoader Class.
- Then we set the device to cuda to access the GPU using chainer.cuda module.
- Then we instantiated the current state object of the environment using the State Class.
- Further, we instantiated our model object which incorporates our policy and value networks and also the Adam optimizer using chainer.optimizers module.
- Moving forward, we instantiated our agent object using our PixelWiseA3C Class and set the agent's to GPU.
- **Function** `test`**:**

  This function conducts testing on the trained agent's performance using a given data loader and a pre-trained agent. It evaluates the agent's behavior and computes its performance metrics such as PSNR (Peak Signal-to-Noise Ratio) while generating denoised images. The steps involved are:
  - Iterating through the testing data in batches.

- Initializing the current state with the testing data and generating noise.
- Running the agent in a testing episode, collecting rewards, and computing the PSNR between original and denoised images.
- Saving the denoised images generated by the agent for visualization and evaluation purposes.

- **Training Loop (`for episode in range(1, N_EPISODES+1):`):**

  This loop trains the agent to denoise images using reinforcement learning. It iterates through episodes, each representing a sequence of actions and training steps. Each episode consists of:

  - Loading training data and generating noise for the images.
  - Initializing the current state with the noisy images and starting with zero rewards.
  - Running the agent for a set number of steps (`EPISODE_LEN`), collecting rewards, and updating its policy and value functions through `act_and_train` method.
  - Computing the rewards and updating the agent's policy based on the experience gained in the episode.
  - Periodically during training, the agent's performance is evaluated using the `test` function on a separate test dataset (`TEST_EPISODES`).
  - The agent's state is saved at specific intervals (`SNAPSHOT_EPISODES`) to maintain checkpoints.
  - Learning rate decay is applied to the optimizer throughout training (`optimizer.alpha` is adjusted).

- **Hyperparameters**

Table 2: Hyperparameters Used

Hyperparameter	**Value**
Learning Rate	0.001
Train Batch Size	64
Test Batch Size	1
N_episodes	30000
N_episodes_test	5
Snapshot_episodes	300
Gamma	0.95
Mean	0
Sigma	15
N_actions	9
Move Range	3
GPU ID	0
Crop Size	248

3.2.7 Interpretability and Explainability using Agent Action Maps

- Interpretability in reinforcement learning for denoising entails visualizing agent actions and their impact on noisy images. By mapping action probabilities and rewards, we reveal where and why the agent makes specific denoising decisions. Analyzing network architectures helps understand learned

patterns. Translating actions into human-readable steps and creating interactive visual tools enhances understanding. Documentation of findings and validation with experts ensures alignment with domain knowledge. Through these methods, interpretability provides insights into how the agent navigates image denoising, fostering transparency and comprehension in its decision-making process.

- Here, we showcase the actions taken by our multi-agents for 248x248 images in the test dataset across five episodes. The recorded actions for each episode are stored in the 'agent_actions.txt' file.
  The actions correspond to specific image processing filters:
  - **0:** Gaussian Blur with a 5x5 filter size and sigma = 0.5
  - **1:** Bilateral Filter with a 5x5 filter size, sigmaColor = 0.1, and sigmaSpace = 5
  - **2:** Median Blur with a 5x5 filter size
  - **3:** Gaussian Blur with a 5x5 filter size and sigma = 1.5
  - **4:** Bilateral Filter with a 5x5 filter size, sigmaColor = 1.0, and sigmaSpace = 5
  - **5:** Box Filter with a 5x5 filter size
- For each state out of the total 5 episodes, each test image has 248x248 pixels leading to the same number of agent actions in the continuous space.
- Therefore, for each state we have an agent matrix with dimensions 248x248 with values from the set 0, 1, 2, 3, 4, 5 specifying the type of filter selected by each agent from the toolbox mentioned above.
- Then, we created a color-coded representation of the agent's actions by mapping each action to a specific color defined in the action_color_map dictionary.

$$\text{action\_color\_map} = \{0 : \text{'gray'}, 1 : \text{'brown'}, 2 : \text{'aqua'}, 3 : \text{'navy'}, 4 : \text{'violet'}, 5 : \text{'red'}\}$$

- Looping through each pixel of the 248x248 image, it assigns a color to represent the corresponding action using predefined RGB values for different colors such as gray, brown, aqua, navy, violet, and red based on the action color mapping.

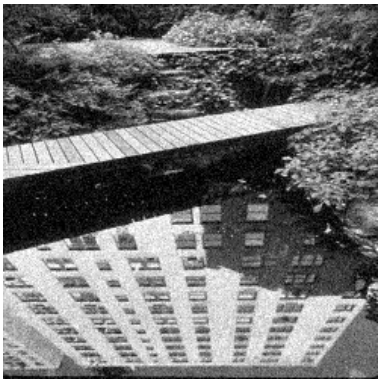


Figure 7: Input Image with added noise

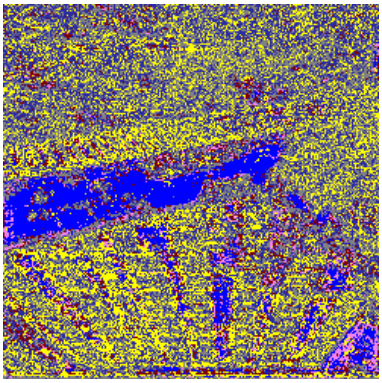


Figure 8: Agent Acton Map for Episode 5

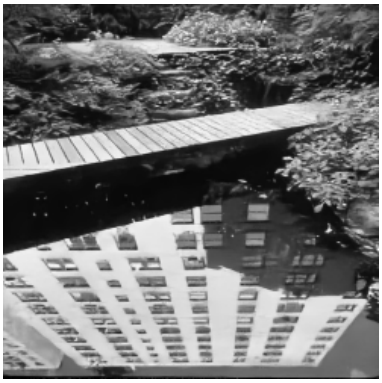


Figure 9: Output Denoised Image after trained agent actions

Figure 10: Agent Action Map for a sample image

4 Results and Discussion

4.1 Episode-wise training results on BSD-68 Dataset

- Following are the results on the test BSD-68 dataset while training the policy and value networks for 30000 episodes.

Episode 300 | Episode 3000 | Episode 6000 | Episode 12000 | Episode 18000 | Episode 24000 | Episode 30000

Figure 11: Episode-wise training

Episode	Total Train Rewards
1	-188.98345
300	132.70823
3000	146.28993
6000	150.58558
12000	152.38966
18000	152.51031
24000	156.90663
30000	153.66375

Table 3: Increasing Training Rewards

Total Test Reward	PSNR
149.75215	29.77506

Table 4: Total Test Rewards and PSNR for Test Images

4.2 Testing results for 5 episodes and sigma=15

Sample 1 - Input

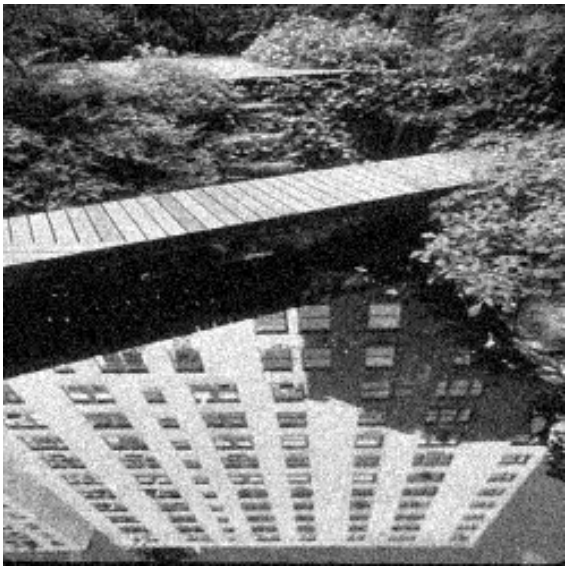
Sample 1 - Output

Sample 2 - Input

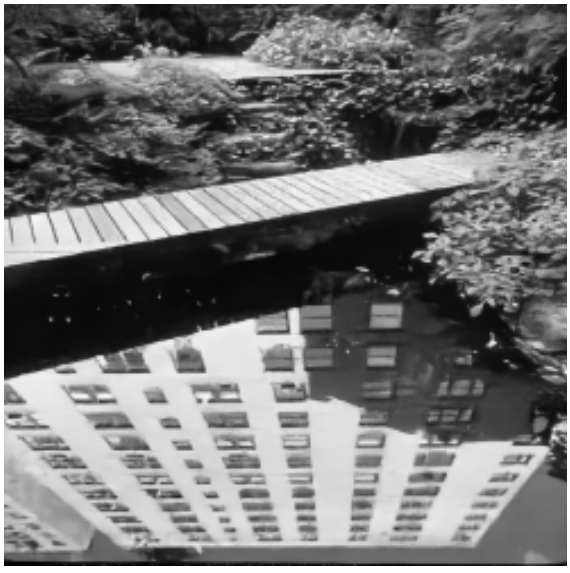
Sample 2 - Output

Sample 3 - Input

Sample 3 - Output



Sample 4 - Input



Sample 4 - Output

4.3 Results on custom images to check for multiple noises

Total Test Reward	PSNR
207.80816	36.85772

Table 5: Total Test Rewards and PSNR for Custom Images

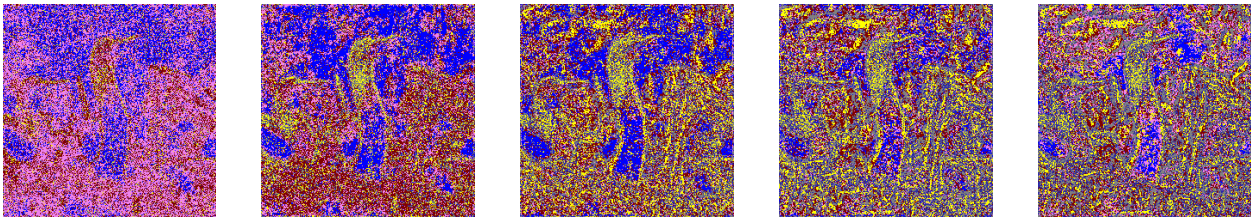
Custom sample 1 - Input

Custom sample 1 - Output

Custom sample 2 - Input

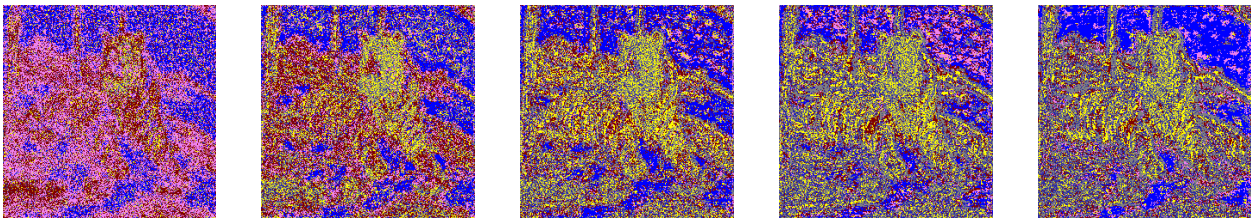
Custom sample 2 - Output

4.4 Interpretable Results using Agent Action maps



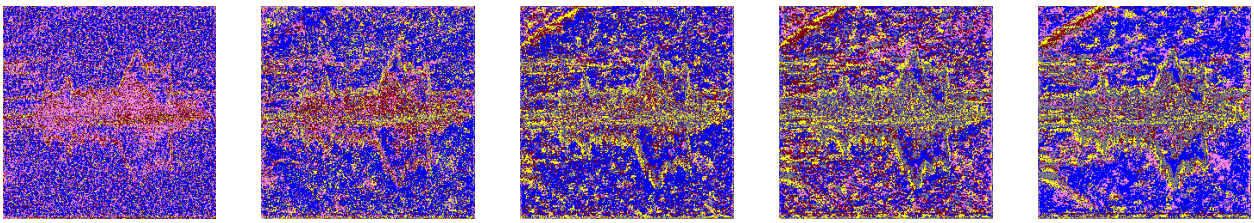
t=0 t=1 t=2 t=3 t=4

Figure 12: Agent Action Map Sample - 1



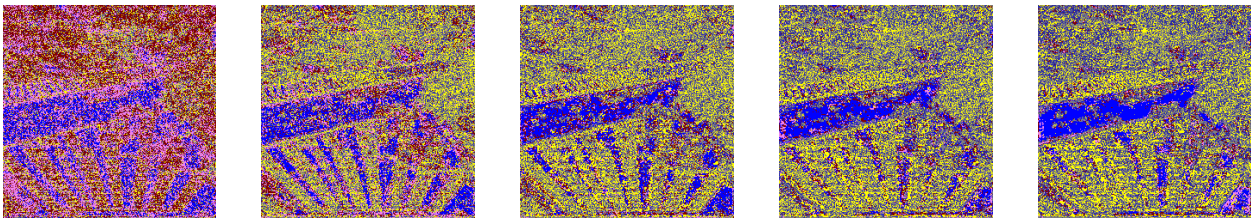
t=0 t=1 t=2 t=3 t=4

Figure 13: Agent Action Map Sample - 2



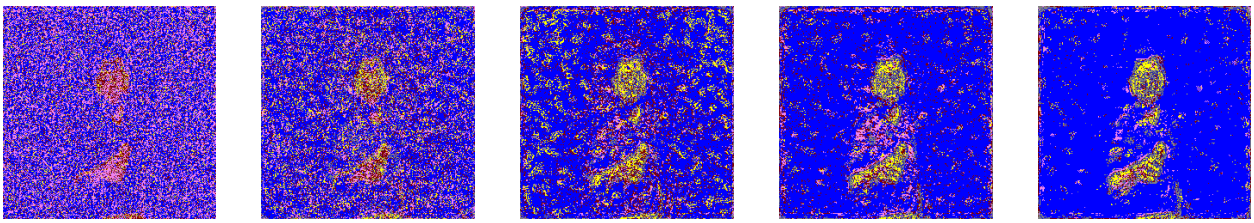
t=0 t=1 t=2 t=3 t=4

Figure 14: Agent Action Map Sample - 3



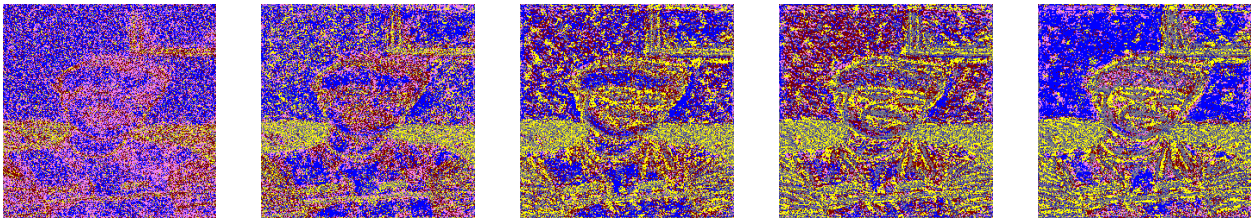
t=0 t=1 t=2 t=3 t=4

Figure 15: Agent Action Map Sample - 4



t=0 t=1 t=2 t=3 t=4

Figure 16: Agent Action Map Custom Sample - 1



t=0 t=1 t=2 t=3 t=4

Figure 17: Agent Action Map Custom Sample - 2

5 Conclusion

In conclusion, through our B.Tech project, we have presented a deep reinforcement learning-based method for multi-noise image restoration. The proposed method uses a POMDP framework and pixel-wise denoising operations to address the challenges of multi-noise image restoration. The method also introduces a novel exploration scheme to avoid overfitting and produce interpretable results. The experimental results demonstrate that the proposed method outperforms existing approaches in terms of PSNR and SSIM metrics on the BSD-68 dataset. The method achieves state-of-the-art performance in multi-noise image restoration and provides a promising direction for future research in this field.

References

[1] Zhang, J., Zhang, Q., Zhao, X. et al. *Boosting denoisers with reinforcement learning for image restoration.* **Soft Comput 26 (2022)**, 3261–3272.

[2] Agustsson E, Timofte R (2017) Ntire 2017 *Challenge on single image super-resolution: dataset and study.* **Proceedings of the IEEE conference on computer vision and pattern recognition workshops**, 127-135.

[3] Anaya J, Barbu A (2018) *RENOIR-a dataset for real low-light image noise reduction.* **J Vis Commun Image Represent**, 51:144-154.

[4] Bertsekas DP, Tsitsiklis JN (1995) *Neuro-dynamic programming: an overview.* **Proceedings of 1995 34th IEEE conference on decision and control.**, 560-564.

[5] Bowling M, Veloso M (2001) *Rational and convergent learning in stochastic games.* **International joint conference on artificial intelligence. Lawrence Erlbaum Associates Ltd**, 1021-1026.

[6] Buades A, Coll B, Morel J-M (2005) *A non-local algorithm for image denoising.* **2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)**, 60-65.

[7] Burger HC, Schuler CJ, Harmeling S (2012) *Image denoising: Can plain neural networks compete with BM3D?* **2012 IEEE conference on computer vision and pattern recognition.**, 2392-2399.

[8] Buşoniu L, Babuška R, De Schutter B (2010) *Multi-agent reinforcement learning: an overview.* **Innovations in multi-agent systems and applications-1.**, 183-221.

[9] Cao Q, Lin L, Shi Y (2017) *Attention-aware face hallucination via deep reinforcement learning.* **Proceedings of the IEEE conference on computer vision and pattern recognition**, 690-698.

[10] Chen Y, Yu W, Pock T (2015) *On learning optimized reaction diffusion processes for effective image restoration.* **Proceedings of the IEEE conference on computer vision and pattern recognition**, 5261-5269.

[11] Chen W, Wilson J, Tyree S (2015) *Compressing neural networks with the hashing trick.* **International conference on machine learning**, 2285-2294.

[12] Dabov K, Foi A, Egiazarian K (2007) *Image denoising by sparse 3-D transform-domain collaborative filtering.* **IEEE Trans Image Process**, 16:2080-2095.