Rate-In: Information-Driven Adaptive Dropout Rates for Improved Inference-Time Uncertainty Estimation

Supplementary Material

6. Appendix A: Reproducibility

6.1. Reproducibility Statement

The Rate-In algorithm code and implementation examples are available in the **GitHub repository**[1].

- **Environment and Data:** Dependencies are listed in requirements.txt. Datasets (PathMNIST, BloodMNIST, TissueMNIST) are from MedMNIST-V2 [42] and Liver CT and Prostate MRI datasets are from the Medical Segmentation Decathlon [2]. Download and preprocessing instructions are included in the original repositories.
- **Training and Evaluation:** Hyperparameters and training schedules are in supplement section 6.2.2. Pre-trained models for classification [42] and segmentation [20] are publicly available. Dropout baseline and evaluation metric scripts and formulations are in the repository and supplement section 6.2.3 and 6.2.4, respectively.
- **Hardware and Software:** Experiments used Python 3.9.12 and PyTorch 2.3.0 on an NVIDIA GeForce RTX 2080 Ti GPU. The code is hardware-agnostic but runtimes may vary on different setups.

6.2. Experiment Setting Supplement

6.2.1. Rate-In Configuration

To enable direct comparison with standard dropout baseline approaches, we initialized Rate-In's parameters as follows: The initial dropout rate p_0 was set equal to the dropout rate p used in the constant dropout benchmark approach. The objective information loss threshold ϵ was set to p, representing similar proportion of information to be preserved. For example, with a constant dropout rate of $p = 0.2$, we set $p^{(0)} = 0.2$ and $\epsilon = 0.2$, targeting 80% information preservation. We set δ to 0.01.

We defined information loss at layer (l) as the relative change in mutual information (MI) between input and post-dropout feature maps. Let $I^{(l)}_{\text{drop}}$ and $I^{(l)}_{\text{full}}$ represent MI in dropout and full models at layer l:

$$I^{(l)}_{\text{drop}} = \text{MI}(\mathbf{h}^{(0)}\text{in}, \mathbf{h}^{(l)}_{\text{drop}}\text{out})$$

$$I^{(l)}_{\text{full}} = \text{MI}(\mathbf{h}^{(0)}\text{in}, \mathbf{h}^{(l)}_{\text{full}}\text{out})$$

The information loss at layer l is:

$$\Delta I_l = \frac{I^{(l)}_{\text{drop}} - I^{(l)}_{\text{full}}}{I^{(l)}_{\text{full}}}$$

[1] https://github.com/code-supplement-25/rate-in/tree/main

Network Architectures and MI Calculation: For synthetic data, we used fully connected networks and calculated mutual information (MI) in two steps: first, computing batch-level MI between the input and each normalized hidden unit using 2D histograms, then averaging these values across all units. For real-world data using CNNs, we first resampled all feature maps to a uniform spatial resolution to maintain spatial consistency. We then calculated MI using adaptive estimators with entropy-equal bins [7].

6.2.2. Architectures of Neural Networks Used

Synthetic Data: The network was implemented in PyTorch with an architecture consisting of three linear layers ([1→50→50→1]) with ReLU activations between hidden layers. Training utilized Adam optimizer (lr=0.01) and mean squared error (MSE) loss over 1000 epochs, processing the entire dataset (N=100) in each iteration. For reproducibility, NumPy and PyTorch random seeds were set to 123. Dropout layers were placed after each hidden layer post-training (Figure 5).

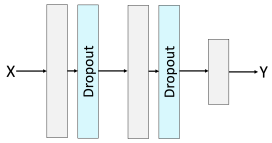


Figure 5. Basic layout of the regression network with dropout layers shown in blue.

Classification: ResNet-18 networks were employed using dataset-specific pre-trained weights and pre-processing protocols from MedMNIST-V2 [42]. The TissueMNIST, PathMNIST, and BloodMNIST datasets were provided as 28x28 voxel inputs. Dropout layers were placed after after each residual block in the pre-trained network (Figure 6).

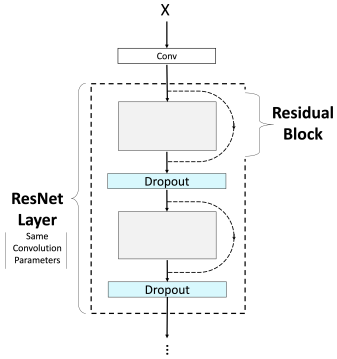


Figure 6. A single ResNet layer with dropout layers shown in blue. The ResNet-18 architecture comprises four such layers.

Segmentation: The segmentation pipeline employs nnU-Net, using task-specific pre-trained weights from the top-performing models in the Medical Segmentation Decathlon [2]. Each model utilizes a self-configuring segmentation pipeline based on the U-Net architecture with 2D convolutional layers [20]. Dropout layers were placed after each encoding and decoding step in the pre-trained network, except for the output step (Figure 7).

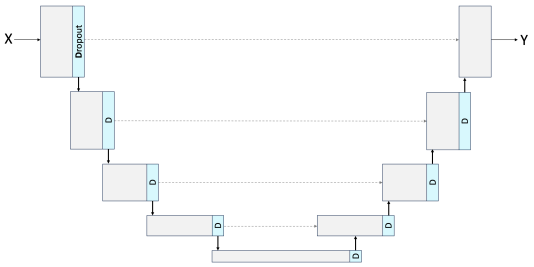


Figure 7. Basic layout of the U-Net network with dropout layers shown in blue.

6.2.3. Baseline Dropout Strategies

In the absence of widely established inference-time dropout strategies, we drew inspiration from training-time dropout approaches, adapting them for inference-time uncertainty estimation.

- **Constant Dropout** [38]: Dropout rates are fixed across all layers and MC iterations, represented as:

$$p_t^{(l)} = p \quad \forall\, l, t$$

  where l represents the layer and t the MC iteration.
- **Scheduled (Annealing) Dropout** [35]: Dropout rates are constant across layers but decrease linearly over MC iterations, starting with $p_0 = p$ and gradually reducing until dropout is fully disabled. This is given by:

$$p_t^{(l)} = p \cdot \left(1 - \frac{t-1}{T-1}\right) \quad \forall\, l, t$$

  where T is the total number of MC iterations.
- **Activation-Based Dropout**: Dropout rates vary across layers based on the activation diversity, measured by the coefficient of variation (CoV) of feature map values, while remaining constant across MC iterations. For each layer l and iteration t, the dropout rate $p_t^{(l)}$ is set as:

$$p_t^{(l)} = p \cdot \frac{\mathrm{CoV}(X^{(l)})}{\max_j \mathrm{CoV}(X^{(j)})} \quad \forall\, l, t$$

  where $X^{(l)}$ is the feature map for layer l in the full no-dropout model, and p is the maximum dropout rate applied to the layer with the highest CoV. To handle both positive and negative values in feature maps, CoV is calculated using the mean of absolute values for stability. Our Activation-Based Dropout method is inspired by training-time dropout approaches that adjust dropout rates based on activation patterns during training, as seen in variational dropout [22] and adaptive dropout [3].

6.2.4. Evaluation Metrics

Predictive Power: Diluting neural network models can impact their performance. We assessed the deviation from the full (no-dropout) model's prediction performance using the same evaluation metrics as those provided in the original public repository where the dataset was published.

- **Mean Squared Error (MSE):** For regression, MSE measures the average squared difference between predicted values and ground truth values, quantifying the error magnitude. The MSE is defined as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  where y_i represents the ground truth value, $\hat{y}_i$ is the predicted value, and n is the total number of data points. MSE provides a cohort-level evaluation by averaging the squared errors across all data instances.
  In this study, we used the implementation from `torchmetrics.MeanSquaredError`, with its default parameters.
- **Dice Similarity Coefficient (DSC):** For segmentation, DSC measures pixel-wise overlap between predicted binary mask and ground truth regions. The DSC is defined as:

$$\mathrm{DSC} = \frac{2|P \cap G|}{|P| + |G|}$$

  where P is the set of predicted pixels, G is the set of ground truth pixels, and $|\cdot|$ denotes the cardinality of the set. DSC was calculated at the pixel level and averaged across all data instances to provide a cohort-level evaluation.
  In this study, we used the implementation from `torchmetrics.classification.Dice`, with its default parameters.
- **Predictive Accuracy (ACC):** For classification, ACC measures the proportion of correctly classified instances over the total number of instances. The ACC is defined as:

$$\mathrm{ACC} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Instances}}$$

  In this study, we used the implementation from `sklearn.metrics.accuracy_score`.

Predictive Uncertainty: Uncertainty estimates were assessed as scores that classify model predictions into two outcomes: reject or do-not-reject, following [23, 30, 31]. To evaluate the alignment of uncertainty estimates with model prediction errors, we used:

- **Expected Calibration Error (ECE)**: For segmentation uncertainty, ECE measures calibration by aligning uncertainty scores with actual model errors [25]. The ECE is defined as:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{N} \left|\text{uncert}(B_m) - \text{err}(B_m)\right|$$

where M is the total number of bins, B_m is the set of pixels in bin m, $|B_m|$ is the number of pixels in B_m, N is the total number of pixels, $\text{uncert}(B_m)$ is the average predicted uncertainty score, and $\text{err}(B_m)$ is the actual model error within the bin. ECE was calculated at the pixel level and averaged across all data to provide a cohort-level evaluation.
In this study, we used the implementation from `torchmetrics.CalibrationError` with parameters `CalibrationError(task='binary', norm='l1', n_bins=15)` and all other parameters set to their default values.
- **Area Under the Accuracy-Rejection Curve (AUARC)**: For classification uncertainty, AUARC evaluates the trade-off between predictive accuracy and the fraction of predictions rejected based on uncertainty scores [15, 32]. The AUARC is defined as:

$$\text{AUARC} = \int_0^1 \text{ACC}(r)\, dr$$

where r is the rejection fraction, and $\text{ACC}(r)$ is the accuracy of the retained predictions at rejection fraction r. Uncertainty scores corresponded to the highest predicted class was used. Accuracy was calculated for rejection thresholds corresponding to each uncertainty score percentile, and the area under the curve was computed using `sklearn.metrics.auc`. Higher AUARC values indicate that the model maintains high accuracy while effectively rejecting uncertain predictions.
- **Boundary Uncertainty Consistency (BUC):** Accurate estimation of uncertainty at organ boundaries is critical in medical imaging, as these regions often exhibit high uncertainty due to anatomical variability, image noise, and ambiguous structures [27, 28, 43]. BUC quantifies the proportion of total uncertainty concentrated along boundaries compared to interior regions.

$$\text{BUC} = \frac{\text{Mean}(\text{uncert}_{\text{Boundary}})}{\text{Mean}(\text{uncert}_{\text{Boundary}}) + \text{Mean}(\text{uncert}_{\text{Interior}})},$$

where $\text{Mean}(\text{uncert}_{\text{Boundary}})$ and $\text{Mean}(\text{uncert}_{\text{Interior}})$ represent the average uncertainties in boundary and interior regions, respectively. The boundary region in our experiments was defined as a 5-pixel-wide band surrounding the edges of the segmented objects.
BUC highlights the model's ability to assign higher uncertainty to critical boundary areas, ensuring robust uncertainty evaluation essential for clinical decision-making.
- **Interval Efficiency Ratio (IER):** Measures the trade-off between predictive interval width and its coverage probability (PICP), quantifying the efficiency of uncertainty estimation. The interval width represents the average range of the prediction intervals and is calculated as:

$$\text{Uncertainty Interval Width} = \frac{1}{N} \sum_{i=1}^{N} 2 \cdot Z \cdot \sigma_i$$

where Z represents the Z-score corresponding to the desired confidence level (e.g., $Z = 1.96$ for 95% confidence intervals), and σ_i denotes the predicted standard deviation, calculated as the standard deviation of the Monte Carlo (MC) predictions at each data point i.
The PICP quantifies the proportion of true values y_{true} that fall within the predicted intervals and is defined as:

$$\text{PICP} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left[y_{\text{true},i} \in (\mu_i - Z \cdot \sigma_i, \mu_i + Z \cdot \sigma_i)\right]$$

where $\mathbb{I}$ is the indicator function, μ_i is the predicted mean, and N is the number of samples.
The ratio of interval width to PICP evaluates the trade-off between the tightness of prediction intervals and the coverage of true values, with lower ratios indicating more efficient uncertainty estimation.

7. Appendix B: Additional Results

7.1. Synthetic Data

Figures 8 and 9 evaluate Rate-In dropout's performance. Figure 8 compares uncertainty intervals from constant dropout ($p = 0.10$, red) and Rate-In dropout ($\epsilon = 0.10$, blue) across five noise levels ($\sigma = 0.1 - 0.5$), showing Rate-In's more precise and stable bounds under increasing noise. Figure 9 examine performance at fixed noise levels ($\sigma = 0.01, 0.10, 0.50$) across varying training set sizes, showing uncertainty interval efficiency ratios.

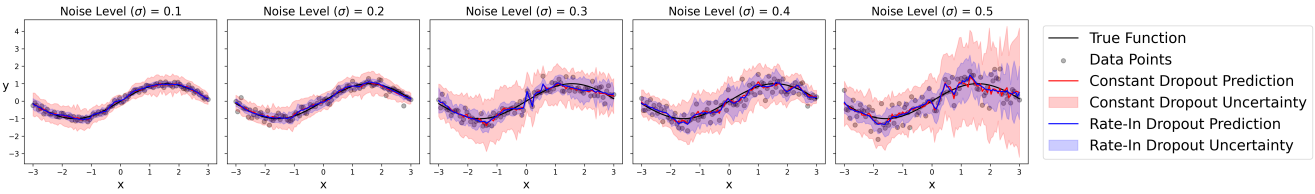
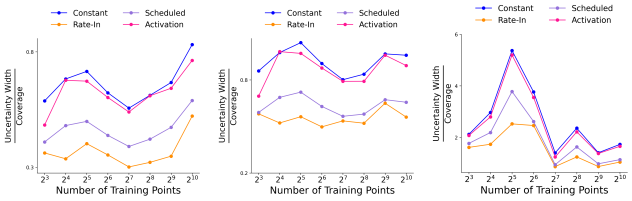


Figure 8. Rate-In dropout yields narrower, more stable uncertainty intervals under increasing noise levels ($\sigma = 0.1 - 0.5$), compared to constant dropout. Black: true function and training data; Red: constant dropout ($p = 0.10$); Blue: Rate-In dropout ($\epsilon = 0.10$).



(a) $\sigma = 0.01$ (b) $\sigma = 0.10$ (c) $\sigma = 0.50$

Figure 9. **Rate-In is more efficient in uncertainty estimation.** The ratio of uncertainty interval width to 95% coverage for varying number of training points at fixed noise levels. Lower ratios indicate more efficient uncertainty estimation. (a) $\sigma = 0.01$, (b) $\sigma = 0.10$, and (c) $\sigma = 0.50$.

Figure 10 analyzes the convergence behavior of Rate-In across varying initial dropout rates under different information loss thresholds ($\epsilon = 0.10, 0.30, 0.50$), with $N_{max} = 100$ and $\sigma = 0.50$. The results demonstrate that Rate-In converges to consistent dropout rates, independent of the initial dropout rate values.

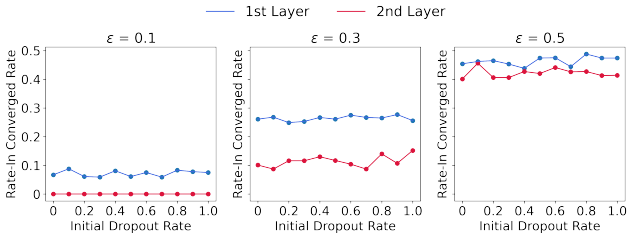


Figure 10. Rate-In final dropout rate (y-axis) versus initial dropout rate (x-axis) for different information loss thresholds ($\epsilon = 0.10$, 0.30, 0.50). Parameters: $N_{max} = 100$, $\sigma = 0.50$.

7.2. Classification

Figure 11 presents the performance of Rate-In on the PathMNIST dataset, demonstrating its ability to maintain high accuracy and better uncertainty calibration compared to baseline methods. Notably, Rate-In sustains strong performance across varying levels of information loss.

7.3. Segmentation

Table 4 presents the comparison between Rate-In and all baseline dropout approaches across segmentation tasks.

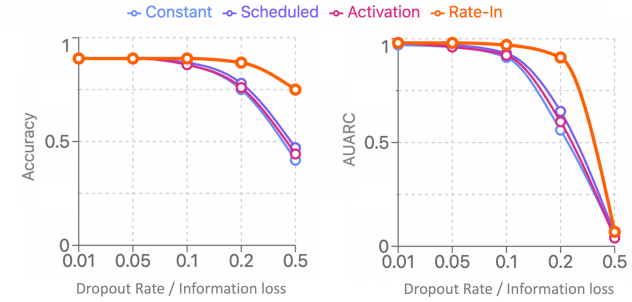


Figure 11. **Rate-In maintains high classification accuracy even at aggressive dropout rates where conventional methods significantly degrade.** Classification accuracy on PathMNIST dataset comparing different dropout strategies across increasing dropout rates. While traditional methods show sharp accuracy decline, Rate-In preserves performance by adaptively adjusting dropout patterns based on local feature importance, demonstrating its robustness for reliable medical image classification.

7.4. Out-of-Distribution (OOD) Tasks

Table 5 reports the mean accuracy (AUARC) across all corruption types in the MedMNIST-C datasets for Rate-In and all benchmark methods.

7.4.1. Natural Scene Images

We employed the ImageNet dataset from the ImageNet Object Localization Challenge [19], which consists of 50,000 natural scene images spanning 1,000 categories. For our experiments, we used the PyTorch implementations of MobileNet-V3-Large, VGG-16, and ResNet-50, along with their pre-trained IMAGENET1K_V1 weights [34]. Dropout layers were introduced after each convolutional layer in all three networks.

Figure 12 illustrates the information loss patterns induced by dropout across all layers of the three neural network architectures. Dropout led to substantial information loss in early layers, with a 50% reduction for rates between 0.05 and 0.15. ResNet-50 showed the greatest variability across layers.

Table 4. Comparison of Rate-In and baseline dropout approaches for uncertainty estimation and segmentation accuracy across anatomical zones in MRI and CT modalities, evaluated using DSC, ECE, and BUC metrics at varying dropout rates. * Best baseline dropout method is defined by the highest average DSC across dropout rates per task. The overall **best-performing approach across metrics is in bold**

Zone	Modality	Dropout Method	DSC	DSC (Full Model, % change)			ECE $\times 10^{-3}$			BUC		
			Full Model	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10
Peripheral	MRI	**Rate-In**	0.682	+0.29%	+2.79%	+2.35%	4.60	**4.26**	**4.58**	**0.67**	**0.63**	**0.61**
		Constant*		+2.05%	+2.59%	+2.05%	**4.40**	5.547	7.85	0.54	0.48	0.45
		Scheduled		+1.06%	+2.84%	+2.51%	4.44	5.14	7.24	0.55	0.47	0.45
		Activation		+1.17%	+2.79%	+2.64%	4.62	4.40	5.01	0.58	0.53	0.50
Transitional	MRI	**Rate-In**	0.892	-0.34%	-0.78%	-0.67%	**3.97**	**4.03**	**4.86**	**0.87**	**0.84**	**0.80**
		Constant*		+0.11%	-0.78%	-0.78%	4.25	6.60	10.73	0.70	0.62	0.57
		Scheduled		-0.16%	-0.69%	-0.74%	4.30	6.15	9.64	0.71	0.63	0.58
		Activation		-0.18%	-0.71%	-0.69%	4.13	4.55	5.76	0.78	0.71	0.67
Liver	CT	**Rate-In**	0.955	+1.05%	+1.05%	+0.94%	**6.73**	**5.54**	**4.65**	**0.93**	**0.93**	**0.92**
		Constant		-0.00%	-0.01%	-0.03%	6.80	5.80	4.90	0.92	0.91	0.91
		Scheduled*		+0.00%	-0.00%	-0.00%	6.90	5.80	5.00	0.92	0.91	0.91
		Activation		-0.00%	-0.03%	-0.04%	7.00	6.20	5.40	0.92	0.91	0.91
Tumor	CT	**Rate-In**	0.579	+1.21%	-0.52%	-2.59%	**1.78**	**1.78**	**1.88**	**0.61**	0.53	0.48
		Constant		-0.12%	-2.14%	-2.61%	1.80	1.80	1.90	0.59	0.52	0.49
		Scheduled		+0.05%	-1.96%	-2.62%	1.80	1.80	2.00	0.59	0.52	0.49
		Activation*		+0.17%	-1.90%	-1.73%	1.80	1.80	1.90	0.60	**0.55**	**0.53**

Table 5. Rate-In robustness evaluation across MedMNIST-C corruption types. The table compares Rate-In with benchmark methods across three datasets, showing superior performance at varying dropout rates (p,ϵ=0.05-0.20), with average ACC(AUARC) highlighted in the 'Avg.' column.

Dataset	p, ϵ	**Method**	**Avg.**	**bright. down**	**bright. up**	**bubble**	**contrast down**	**contrast up**	**defocus blur**	**jpeg compr.**	**motion blur**	**pixelate**	**saturate**
BloodMNIST	ACC Full Model		.77	.87	.88	.18	.88	.92	.69	.87	.70	.91	.83
	0.05	Rate-In	**.77**(.87)	.86(.96)	**.84**(.94)	.19(.17)	.91(.97)	**.91**(.98)	.70(.89)	**.90**(.97)	**.70**(.88)	**.92**(.98)	**.81**(.95)
		Constant	.76(.87)	.86(.97)	.71(.88)	.19(.20)	**.94**(.98)	.87(.98)	**.74**(.89)	.87(.97)	.69(.89)	.90(.98)	.79(.93)
		Scheduled	.76(.87)	.86(.97)	.77(.90)	.19(.19)	**.94**(.98)	.89(.98)	.72(.88)	.88(.97)	.68(.88)	.90(.98)	.80(.93)
		Activation	**.77**(.87)	.86(.97)	.77(.90)	.19(.20)	**.94**(.98)	.89(.98)	.72(.89)	.88(.97)	**.70**(.88)	**.92**(.98)	**.81**(.94)
	0.10	Rate-In	**.75**(.86)	**.84**(.96)	**.72**(.89)	**.20**(.18)	.94(.98)	**.86**(.98)	.72(.88)	**.87**(.97)	.67(.88)	**.90**(.98)	**.78**(.93)
		Constant	.70(.83)	.80(.95)	.45(.67)	.18(.21)	**.96**(.98)	.82(.95)	**.73**(.86)	.85(.97)	.71(.88)	.81(.96)	.73(.89)
		Scheduled	.73(.84)	.83(.96)	.56(.72)	.19(.22)	**.96**(.99)	.84(.96)	**.73**(.87)	.85(.97)	**.72**(.88)	.86(.97)	.76(.89)
		Activation	.72(.85)	.81(.96)	.55(.74)	.18(.21)	.95(.99)	**.86**(.97)	.72(.87)	.85(.97)	.69(.88)	.87(.97)	.76(.91)
	0.20	Rate-In	**.66**(.79)	**.73**(.89)	**.35**(.52)	**.17**(.20)	**.95**(.98)	**.72**(.87)	**.73**(.86)	**.76**(.92)	**.68**(.85)	**.78**(.94)	**.71**(.86)
		Constant	.45(.55)	.63(.80)	.17(.23)	.12(.17)	.72(.89)	.28(.37)	.64(.71)	.38(.49)	.58(.66)	.54(.64)	.46(.58)
		Scheduled	.56(.66)	.72(.87)	.19(.29)	.12(.17)	.87(.94)	.48(.58)	.66(.77)	.65(.74)	.63(.77)	.70(.80)	.61(.70)
		Activation	.59(.73)	.72(.88)	.19(.28)	.12(.17)	.87(.96)	.62(.79)	.67(.84)	.69(.85)	.67(.83)	.74(.90)	.66(.80)
PathMNIST	ACC Full Model		.54	.44	.42	.67	.51	.79	.34	.25	.53	.78	.71
	0.05	Rate-In	**.52**(.62)	**.43**(.38)	**.39**(.53)	**.63**(.81)	.53(.61)	**.75**(.82)	**.33**(.37)	**.24**(.38)	**.52**(.63)	**.75**(.91)	**.63**(.78)
		Constant	.47(.58)	.42(.35)	.34(.48)	.53(.71)	**.54**(.61)	.69(.78)	.23(.35)	.21(.35)	.50(.60)	.70(.85)	.59(.70)
		Scheduled	.49(.59)	**.43**(.35)	.35(.48)	.58(.73)	**.54**(.60)	.72(.78)	.24(.35)	.22(.36)	.51(.62)	.72(.87)	**.63**(.73)
		Activation	.49(.58)	.42(.34)	.35(.48)	.56(.72)	**.54**(.61)	.70(.78)	.23(.36)	.21(.36)	**.52**(.60)	.71(.86)	.62(.72)
	0.10	Rate-In	**.50**(.59)	**.41**(.35)	**.35**(.51)	**.59**(.76)	**.55**(.59)	**.71**(.79)	**.29**(.37)	**.22**(.36)	**.52**(.60)	**.72**(.88)	**.63**(.74)
		Constant	.37(.47)	.36(.27)	.27(.36)	.38(.47)	.49(.60)	.54(.60)	.16(.28)	.20(.32)	.38(.50)	.55(.70)	.38(.55)
		Scheduled	.39(.49)	.39(.28)	.28(.39)	.41(.52)	.49(.62)	.58(.65)	.18(.30)	.19(.33)	.39(.53)	.58(.74)	.46(.57)
		Activation	.38(.48)	.38(.28)	.28(.38)	.41(.49)	.49(.60)	.57(.63)	.17(.31)	.19(.34)	.39(.53)	.55(.72)	.40(.56)
	0.20	Rate-In	**.39**(.49)	**.31**(.26)	**.29**(.41)	**.40**(.54)	**.49**(.58)	**.52**(.61)	**.24**(.35)	.16(.29)	**.50**(.58)	**.54**(.72)	**.42**(.54)
		Constant	.21(.26)	.13(.17)	.16(.14)	.23(.18)	.30(.49)	.25(.23)	.16(.23)	.19(.25)	.19(.28)	.32(.29)	.18(.29)
		Scheduled	.25(.31)	.20(.19)	.17(.19)	**.24**(.24)	.38(.55)	.31(.33)	.16(.25)	**.22**(.31)	.25(.34)	**.33**(.33)	**.26**(.33)
		Activation	.24(.29)	.19(.17)	.19(.18)	.22(.22)	.35(.52)	.29(.26)	.17(.25)	.21(.33)	.25(.34)	.32(.29)	.21(.31)
TissueMNIST	ACC Full Model		.65	.65	.72	—	.70	.70	—	.55	—	.56	—
	0.05	Rate-In	**.63**(.77)	**.60**(.78)	**.70**(.85)	—	**.72**(.86)	**.64**(.77)	—	**.58**(.61)	—	**.56**(.74)	—
		Constant	.50(.54)	.45(.42)	.57(.70)	—	.59(.73)	.47(.50)	—	.42(.42)	—	.48(.49)	—
		Scheduled	.51(.60)	.46(.51)	.59(.75)	—	.59(.73)	.49(.54)	—	.43(.48)	—	.51(.56)	—
		Activation	.50(.56)	.46(.46)	.57(.72)	—	.60(.74)	.46(.49)	—	.43(.43)	—	.49(.52)	—
	0.10	Rate-In	**.59**(.71)	**.58**(.71)	**.66**(.80)	—	**.67**(.84)	**.58**(.69)	—	**.48**(.52)	—	**.56**(.69)	—
		Constant	.30(.27)	.29(.21)	.41(.44)	—	.36(.33)	.28(.23)	—	.22(.19)	—	.24(.25)	—
		Scheduled	.34(.35)	.30(.25)	.43(.49)	—	.42(.45)	.32(.33)	—	.28(.32)	—	.29(.29)	—
		Activation	.30(.29)	.29(.22)	.42(.43)	—	.35(.35)	.30(.24)	—	.23(.23)	—	.24(.30)	—
	0.20	Rate-In	**.45**(.48)	**.46**(.47)	**.45**(.50)	—	**.56**(.68)	**.37**(.39)	—	**.41**(.34)	—	**.43**(.51)	—
		Constant	.11(.06)	.11(.08)	.11(.05)	—	.12(.06)	.10(.08)	—	.13(.08)	—	.08(.03)	—
		Scheduled	.12(.07)	.12(.09)	.15(.10)	—	.13(.07)	.11(.09)	—	.13(.07)	—	.08(.02)	—
		Activation	.11(.07)	.10(.08)	.12(.06)	—	.13(.07)	.11(.09)	—	.13(.08)	—	.08(.02)	—

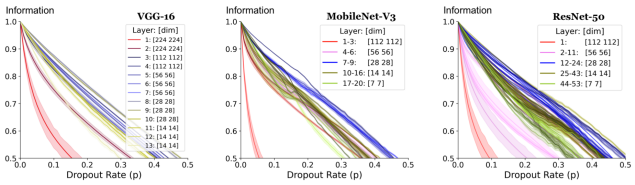


Figure 12. Information loss patterns across VGG-16, MobileNet, and ResNet architectures using ImageNet data.

7.5. Computational Complexity Analysis

The Rate-In algorithm may introduce an additional computational overhead during inference, resulting in increased latency from input availability to prediction. Section 3.3.2 outlines strategies to mitigate this inference-time overhead.

The pseudocode in Listing 1 illustrates the `AdaptiveInformationDropout` class, which implements the optimization process for the Rate-In algorithm. The computational complexity of the dropout rate optimization is $O(N_{\max} \cdot (n + f))$, where $N_{\max}$ is the maximum number of optimization iterations, n is the number of input instances, and f is the complexity of the information loss calculation, which may depend on n.

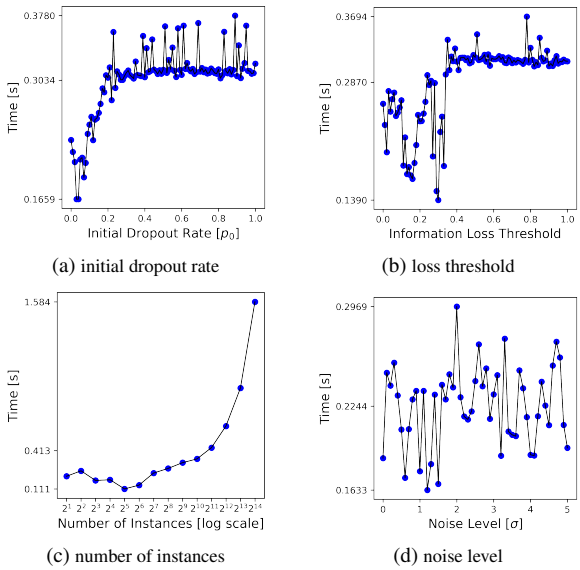
The optimization process dominates the complexity, iterating $N_{\max}$ times with $O(n)$ dropout operations and $O(f)$ calculations per iteration. The computational complexity of each dropout operation, while initially appearing as $O(n)$, further breaks down to $O(d)$, where d is the number of elements in each input instance. Thus, the true complexity of a single dropout operation is $O(n \cdot d)$.

Once the dropout rates are determined, they remain fixed during Monte Carlo inference and, therefore, do not introduce additional complexity compared to regular constant dropout approaches.

Listing 1. AdaptiveInformationDropout Class implementation pseudocode for the Rate-In algorithm

```
class
    AdaptiveInformationDropout(torch.nn.Module):
    def __init__(self, ...):
        # O(1)
        ...

    def _apply_dropout(self, x: torch.Tensor,
        rate: float) -> torch.Tensor:
        # O(n), n = number of elements in x
        return torch.nn.functional.dropout(x,
            p=rate, training=self.training)

    def _optimize_dropout_rate(self, x:
        torch.Tensor) -> float:
        # O(N_max * (n + f))
        # N_max = max_iterations, n = elements in
            x, f = complexity of
            calc_information_loss
        for iteration in
            range(config.max_iterations): #
            O(N_max)
            post_dropout =
                self._apply_dropout(pre_dropout,
                current_rate) # O(n)
            info_loss =
                self.calc_information_loss(...) #
                O(f)
            # Update dropout rate: O(1)
            ...

    def forward(self, x: torch.Tensor) ->
        torch.Tensor:
        if self.training:
            # O(N_max * (n + f) + n) = O(N_max * (n
                + f))
            optimized_rate =
                self._optimize_dropout_rate(x) #
                O(m * (n + f))
            return self._apply_dropout(x,
                optimized_rate) # O(n)
        return x # O(1) if not training

# Overall forward pass complexity: O(N_max * (n
    + f))
```

Synthetic Data: Figures 13 illustrate the time (in seconds) required to apply Rate-In to the synthetic regression task described in Section 4.1. We analyzed how Rate-In execution time varies as a function of four parameters: initial dropout rate (p_0), noise level (σ), loss threshold (ϵ), and number of data instances. For each analysis, we varied one parameter while keeping the others fixed. The maximum number of Rate-In iterations (N_{max}) was set to 30, with $\delta = 0.001$ and a learning rate of 0.9 across all experiments.



(a) initial dropout rate (b) loss threshold

(c) number of instances (d) noise level

Figure 13. **Rate-In execution time analysis for synthetic regression task**. Plots show optimization process duration (in seconds) as a function of: (a) initial dropout rate, (b) information loss threshold, (c) number of inference instances, and (d) noise level.

Rate-In running time plateaus with increased initial dropout rates and information loss probabilities, while scaling linearly with training examples (note log-scale x-axis). Noise levels show minimal impact on execution time. The optimization process occasionally failed to find valid dropout rates in deeper layers, particularly in the second hidden layer. This occurred when the cumulative information loss from earlier layers made it mathematically impossible to achieve the target information loss threshold ϵ, even with a dropout rate of zero. We defined convergence failure as the inability to reduce the objective function below the threshold ϵ after N_{max} iterations, despite reaching the minimum possible dropout rate. Implementation of early stopping could mitigate computational costs.

Medical Data: Tables 6 and 7 report the mean duration (with standard deviation) in seconds for Rate-In processing of individual instances in real-world classification and segmentation tasks for different values of information loss objectives. Initial dropout rate was set to be equal to the information loss objectives ($p_0 = \epsilon$), while all other parameters remained unchanged. The tables also include worst-case scenarios.

Table 6. Mean (± std) and worst-case processing times (in seconds) of MI-based Rate-In for classification tasks.

Dataset	**Metric**	$\varepsilon = 0.05$	$\varepsilon = 0.10$	$\varepsilon = 0.20$
TissueMNIST	Mean (Std)	0.42 (0.07)	0.43 (0.03)	0.39 (0.12)
	Worst-case	0.50	0.45	0.48
PathMNIST	Mean (Std)	0.65 (0.01)	0.67 (0.02)	0.56 (0.12)
	Worst-case	0.66	0.68	0.65
BloodMNIST	Mean (Std)	0.68 (0.01)	0.64 (0.01)	0.48 (0.12)
	Worst-case	0.70	0.65	0.57

Table 7. Mean (± std) and worst-case processing times (in seconds) of MI-based Rate-In for segmentation tasks.

Dataset	**Metric**	$\varepsilon = 0.01$	$\varepsilon = 0.05$	$\varepsilon = 0.10$
Prostate	Mean (Std)	15.48 (5.48)	22.48 (7.48)	28.8 (9.55)
	Worst-case	27.43	40.06	48.72
Liver	Mean (Std)	10.08 (9.59)	28.62 (16.62)	36.11 (18.74)
	Worst-case	56.89	77.4	95.76

7.6. Structural Similarity Index (SSIM)

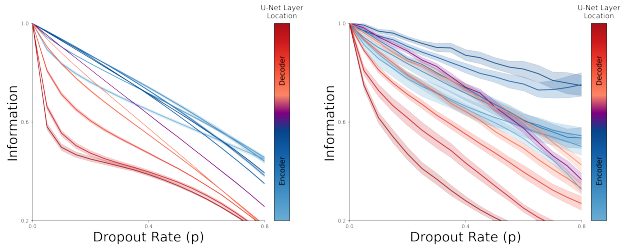
The Rate-In algorithm enables domain-specific information loss measurements through user-defined metrics. While mutual information quantifies statistical dependencies affected by dropout noise, the SSIM Index [41] can help evaluate spatial relationships in feature maps. This property makes SSIM relevant for CNNs, as it measures the preservation of local structures and patterns that these networks extract through their convolutional layers.

We used SSIM to measure information loss between pre- and post-dropout feature maps. This approach allows for simpler implementation as it only requires local comparisons within each layer, without needing access to the full model or input data. Based on earlier experiments showing non-negative SSIM values under dropout noise, the information loss at layer l is defined as: $\Delta I_l = 1 - \text{SSIM}(\mathbf{h}^{(l)}_{\text{drop}}\text{in}, \mathbf{h}^{(l)}_{\text{drop}}\text{out})$

The performance comparison between SSIM-based and MI-based Rate-In across segmentation tasks is presented in Table 9, with corresponding processing times for the SSIM approach shown in Table 8. To ensure consistent SSIM computation across different network layers, we normalized all feature maps to the [0,1] range. We set data_range=1 in the torchmetrics SSIM implementation to match this normalized range, allowing meaningful comparison of structural similarities regardless of the original feature map magnitudes. Figure 14 shows SSIM (left) and MI (right) dropout-induced information loss across layers in the U-Net prostate segmentation network.

Table 8. Mean (± std) and worst-case processing times (in seconds) of SSIM-based Rate-In for segmentation tasks.

Dataset	**Metric**	$\varepsilon = 0.01$	$\varepsilon = 0.05$	$\varepsilon = 0.10$
Prostate	Mean (Std)	9.5 (1.53)	28.29 (4.59)	46.72 (7.23)
	Worst-case	13.86	44.07	71.9
Liver	Mean (Std)	11.39 (12.59)	13.08 (4.71)	17.13 (7.62)
	Worst-case	71.46	22.88	33.13



(a) SSIM vs. Dropout Rate (b) MI vs. Dropout Rate

Figure 14. Dropout-induced information loss in the U-Net segmentation network as measured by SSIM (left) and MI (right) across different dropout rates and layers.

8. Appendix C: Further Discussion

Dropout, beyond just graph manipulation. Rather than viewing dropout as merely a tool for graph manipulation, Rate-In reinterprets it as a method for controlled noise injection. This perspective shift allows us to examine how dropout affects network representations in task-specific contexts. Rate-In effectively acts as a 'noise translator', converting random dropout noise into meaningful, task-specific variations in feature maps. Similar to how test-time

Table 9. Medical image segmentation performance of Rate-In across anatomical regions in MRI and CT: comparing segmentation accuracy (DSC) and uncertainty calibration (ECE, BUC) using MI and SSIM-based information metrics at varying dropout rates.

Zone	Modality	Method	DSC	DSC (Full Model, % change)			ECE $\times 10^{-3}$			BUC		
			Full Model	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10	p,ϵ=0.01	p,ϵ=0.05	p,ϵ=0.10
Peripheral	MRI	Rate-In (MI)	0.682	+0.29%	+2.79%	+2.35%	4.60	**4.26**	**4.58**	**0.67**	**0.63**	**0.61**
		Rate-In (SSIM)		+3.05%	+2.59%	+1.82%	**4.57**	4.52	5.54	0.61	0.54	0.51
Transitional	MRI	Rate-In (MI)	0.892	-0.34%	-0.78%	-0.67%	3.97	**4.03**	**4.86**	**0.87**	**0.84**	**0.80**
		Rate-In (SSIM)		-0.72%	-0.76%	-0.87%	**3.95**	4.78	6.72	0.81	0.72	0.67
Liver	CT	Rate-In (MI)	0.955	+1.05%	+1.05%	+0.94%	**6.73**	5.54	4.65	0.93	0.93	0.92
		Rate-In (SSIM)		+1.82%	+1.16%	+1.18%	6.82	**5.49**	**4.47**	0.93	0.93	0.92
Tumor	CT	Rate-In (MI)	0.579	+1.21%	-0.52%	-2.59%	**1.78**	1.78	1.88	0.61	0.53	0.48
		Rate-In (SSIM)		+1.32%	+1.29%	+1.29%	1.81	**1.76**	**1.74**	**0.64**	**0.60**	**0.55**

augmentation adds meaningful noise to inputs, Rate-In dynamically manages noise levels in feature maps to achieve more accurate uncertainty estimates.

Rate-In, enhancing dropout post-training. Nearly a decade after its introduction, classical Bernoulli dropout is still among the most popular regularization techniques in deep learning, outlasting many newer alternatives. Rate-In builds upon this foundation by introducing dynamic rate adjustment without modifying the underlying dropout mechanism, making it easy for users to adopt Rate-In without altering their existing workflows.

Similar dropout rates, different effects: layer position matters. Our experiments revealed that dropout's impact varies significantly across network layers, even when feature map dimensions are similar. In semantic segmentation tasks, we found that decoder layers showed higher sensitivity to dropout rate changes compared to encoder layers. This variation may be attributed to MRI image characteristics - uniform dark backgrounds in early layers mitigate dropout's impact, while intensity variations in later layers amplify it. In addition, decoder layers have fewer opportunities to compensate for dropout noise, emphasizing the importance of layer-specific rate adjustment.

Sequential dropout adjustments: preserving inter-layer rate dependency The Rate-In algorithm operates sequentially during the forward pass of a pre-trained network. At each dropout layer, a feedback loop adjusts the dropout rate to align with the information loss objectives specific to that layer. After adjustment, the modified information propagates through the network to the next dropout layer, where the process repeats. This approach enables control over information loss across the entire network, establishing a dropout rate policy that maintains inter-layer dependency.

Additional complexity in inference, but can be handled offline. Rate-In introduces additional computation at inference. While this can be challenging for some real-time applications, a few potential solutions can be considered. Rate-In can be applied offline to a dataset, such as the training or validation set, to establish population-level dropout rates that approximate individual case needs, assuming data distribution alignment. These rates can then serve as starting points for further tuning during inference or be sampled from the population rate distribution for each layer.

8.1. Limitations

MI might not fully capture vision task nuances. In our experiments, we used mutual information (MI) to quantify functional information loss from dropout. While MI is a common choice, it may not capture all aspects of information critical in computer vision tasks. Supplementing MI with metrics like the Structural Similarity Index (SSIM) could provide a more comprehensive assessment, covering different dimensions of information quality and enabling more precise dropout rate adjustments to address various types of potential information loss.

Loss objectives may need empirical validation. Rate-In's feedback loop relies on an empirically defined information loss objective. Our experiments suggest that keeping information loss below 10% preserves functional information while retaining dropout's benefits. Currently, each dropout layer applies a local loss function, resulting in distinct dropout rates per layer. The current implementation uses independent loss functions for each dropout layer, leading to layer-specific dropout rates. One potential modification would be to use a network-wide information loss constraint - a 'dropout budget' shared across layers.

Dropout does not cover all types of noise. In convolutional layers, dropout adds noise only within small, localized regions defined by the convolution kernel. As a result,

it may not capture broader noise patterns such as frequency artifacts, intensity shifts, or large-scale texture variations that span entire images. This limitation restricts dropout's effectiveness in addressing global noise types that are more uniformly spread across feature maps.