Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

УДК 007.724.1
Инв. No

Работа допущена к защите
преподаватель
___________Д.Е.Моторин
«_____»___________2023.

Курсовая работа

SDE-Net: Equipping Deep Neural Networks with Uncertainty Estimates

Студент: Эспинола Ривера, Хольгер Элиас
1-й курс магистратура
Искусственый интеллект и машинное обучение

Санкт-Петербург
2023

1. Introduction and Related Works

1.1. Abstract

Purpose and motivation. This research is a purpose of a novel method to estimate an uncertainty in complex deep learning models.

Problem. The traditional methods to estimate the uncertainty in mathematical models like Bayesian and Non-Bayesian approaches don't have the necessary scalability that neural networks require to estimate the uncertainty, having large number of parameters and demanding a lot of computer power. They are not practical methods in this context. The big challenge is to find an appropriate and effective method to estimate the uncertainty in deep learning models.

Approach. To estimate uncertainty in deep neural networks, this research takes a dynamical system perspective. The implementation of Stochastic Differential Equation Network (SDE-Net) interpreting Deep Neural Network (DNN) as state evolution of stochastic dynamical system with Brownian motion term.

Results. This research takes experiments in tasks where uncertainty have a fundamental role. The SDE-Net outperforms the capacity to estimate uncertainty in comparison with currently methods.

Conclusion. The SDE-Net experimentally demonstrated to be more effective than traditional approaches to estimate uncertainty.

1.2. Read the introduction + related work

Introduction

Despite the great results achieved by Deep Neural Network models, many times these models make erroneous predictions with high confidence in scenarios where the evidence to support those predictions are not enough (overconfidence). The overconfidence usually produces inaccurate and unreliable results. To avoid this scenario, is necessary quantify the uncertainty and define what these models don't know.

Currently exists 2 approaches to estimate the uncertainty of models: Bayesian and non-Bayesian methods. In the case of Bayesian Methods, to quantify the uncertainty, these models define probability distributions for each parameter of model. Make these estimations and after that, make the inference are not viable way given the huge volume of parameters and the fact to these parameters don't have semantic interpretation. For the case of non-Bayesian methods, ensemble methods, requires train multiple NN with different initializations. Doing this implies a high computational cost. Adding to this, in other non-Bayesian methods presents a drawback of separating aleatoric and epistemic uncertainty.

The purpose of research considering an implementation of Stochastic Differential Equation Net (SDE-Net) to estimate uncertainty in Deep Neural Network (DNN) models.

The motivation to take this approach is sustained in the connection existing between neural networks and dynamical systems. In fact, we can see the DNN like a dynamical system. The forward passes in hidden layers can be interpreted like state of transformations in dynamical system defined by Neural Ordinary Differential Equation (Neural-ODE). Here, we have the deterministic model. To introduce the uncertainty in the model, the introduction of Brownian motion term, capture the epistemic uncertainty of these nets.

The implementation of the SDE-Net has 2 parts:
(1) drift-net that parametrizes a differential equation, taking the deterministic behavior of the model to fit the predictive function.
(2) diffusion-net that parametrizes the Brownian motion term to quantify epistemic uncertainty. Integrating these 2 parts in the network, the SDE-Net will provide the good capacity to do accurate predictions and estimate the uncertainty adding to this model, a deterministic part of model in a stochastic environment.

Related works

a. Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015, June). Weight uncertainty in neural network.

Bayesian approach to estimate uncertainty. This research purpose a backpropagation-algorithm for learning a probability distribution on the weights of a neural network. With this approach and using regularization methods for the weights, this research demonstrates that the learnt of uncertainty in the weights can be used to improve the generalization levels in non-linear regression problems and can be used to drive the exploration-exploitation trade-off in reinforcement learning. The purposed algorithm called Bayes by Backprop optimizes a well-defined objective function to learn a distribution of the weights of a neural network. In the problem of non-linear regression, the study found good and reasonable predictions in unseen data. The results of this regularization are comparable to the Dropout regularization. In the case of the reinforcement learning, was seen automatic learning of trade-off exploration-exploitation.

b. Hafner, D., Tran, D., Lillicrap, T., Irpan, A., & Davidson, J. (2020, August). Noise contrastive priors for functional uncertainty.

Bayesian approach to estimate uncertainty. In this research was proposed Noise Contrastive priors (NCP) to obtain reliable uncertainty estimates. The main idea of this approach is to train the model to take very good predictions in scenarios where have high uncertainty for data points outside of the training data. To do this, NCP are data priors that are enforced on both training inputs x and inputs x’ perturbed by a noise. Adding to this, priors in data space can easily capture properties such as periodicity or spatial invariance. NCP can prevent overfitting outside for data outside to training distribution. This approach fixes the currently problem in the models that generally uses independent Gaussian distribution, which give limited and even biased information for uncertainty.

c. Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles.

Non-Bayesian approach to estimate uncertainty. In this research, the study takes in account the problems in the requirements to estimate uncertainty in models like non-flexible training procedure and the huge computational cost, and makes an ensemble of NNs, and with this, reoriented the effort to have a good managing of hyperparameter tuning and with this, obtain a high predictive capacity for uncertainty estimates. The predictive uncertainty was made in test examples to know and unknow distributions in the context of classification and regression tasks. Two sources of uncertainty were captured: in probabilistic NN, notes the ambiguity in outputs (y) given the inputs (x). Adding to this, the combination of ensembles takes the averaging predictions over the multiple models which had different initializations, capturing the model uncertainty. With this work, was seen that non-Bayesian approach to estimate uncertainty can provide good predictive uncertainty estimates for the previously mentioned tasks.

d. Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations.

In this research was introduced a deep learning model called Ordinary Differential Equation Network (ODE-Network). This model parametrizes the derivates of the hidden states using a deep neural network and demonstrated that residual neural networks and ODE-Networks present very similar behavior, that suggest the behavior of dynamical systems in these nets. Under this ODE-Network, was implemented new models for time-series modelling, supervised learning and density estimation.

Limitations

All the methods discussed in the related works dealt the problem of the estimation of uncertainty. With the Bayesian approaches, occurs the problem of the huge computation necessary to scale these methods to more complex deep neural nets, which makes it intractable. In other way, the selection of prior gaussian distribution is a problem solved by NCP. In the case of the non-Bayesian approach, makes ensemble of multiple deep neural nets with huge volume of parameters is a limitation for the scalability of these methods for estimate uncertainty in more and more complex models in the currently situation where the trend of the models is to complexity increasing. In the case of the ODE-Network, is interesting the fact to stablish relationship between dynamic systems and deep neural nets, but this model considers just the deterministic part of the problem of the modeling, and this approach needs extension to problem of uncertainty under the models.

2. Model and Method

2.1.List of Annotations

x_t: is the hidden state at layer t
t: index of the hidden layer
$f(x_t, t)$: function of continuous dynamic in the system
dx_t: rate of change in hidden state at layer t
dt: rate of change of indexes of the hidden layer
$g(x_t, t)$: variance of the Brownian motion
W_t: standard Brownian motion
E: mathematic expectation
L(.): loss function – dependently of the task
P_{train}: probability distribution of training data
$P_{OOD:}$ out-of-distribution data
x_0': noisy inputs - inputs added with gaussian noise
Z_k: standard Gaussian random variable ($Z_k \sim \aleph(0,1)$)

2.2. Mathematical Model

The research purpose SDE-Net to estimate uncertainty, taking a stochastic dynamical system perspective and explicitly distinguishing the 2 sources of uncertainty.

The starting point for modeling the problem is take in account the model of transformation between layers in ResNet:

$$x_{t+1} = x_t + f(x_t, t) \tag{1}$$

Where:

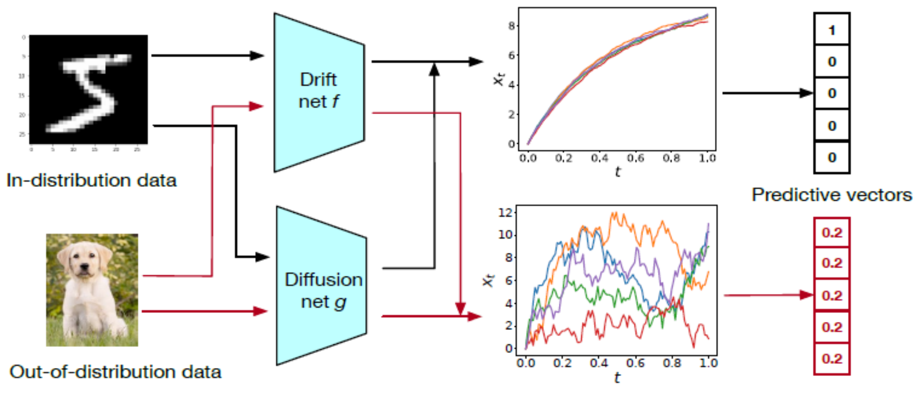
$$\lim_{x \to 0} \frac{x_{t+\Delta t} - x_t}{\Delta t} = \frac{dx_t}{dt} = f(x_t, t) \Leftrightarrow dx_t = f(x_t, t)dt \tag{2}$$

The Eq. (2) is considered the ODE in dynamical systems. However, this deterministic model doesn't model epistemic uncertainty. To capture the epistemic uncertainty, we add the Brownian motion term and the Stochastic Differential Equation is expressed as:

$$dx_t = \underbrace{f(x_t, t)}_{drift\ net} dt + \underbrace{g(x_t, t)}_{diffusion\ net} dW_t \tag{3}$$

With the variance of the Brownian motion, we can control the level of epistemic uncertainty. Scenarios where exists abundant training data and low epistemic uncertainty, the variance of the Brownian motion will be small. In the case when the training data is scarce and have high epistemic uncertainty, the variance of the Brownian motion will be high. This information is contained in $g(x_t, t)$.

2.3. SDE-Net Architecture



The SDE-Net have 2 separated neural networks:

Drift-Net: In Stochastic differential equation represented by Eq. (3), this net model the function *f*. Here, we have the control to system achieve good results. This model captures the aleatoric uncertainty.

Diffusion-Net: In Stochastic differential equation represented by Eq. (3), this net model the function *g*. Here, we have the diffusion of the system. Maybe deterministic and dominated by the drift for small variance of Brownian motion or chaotic and dominated by the diffusion for high variance of Brownian motion.

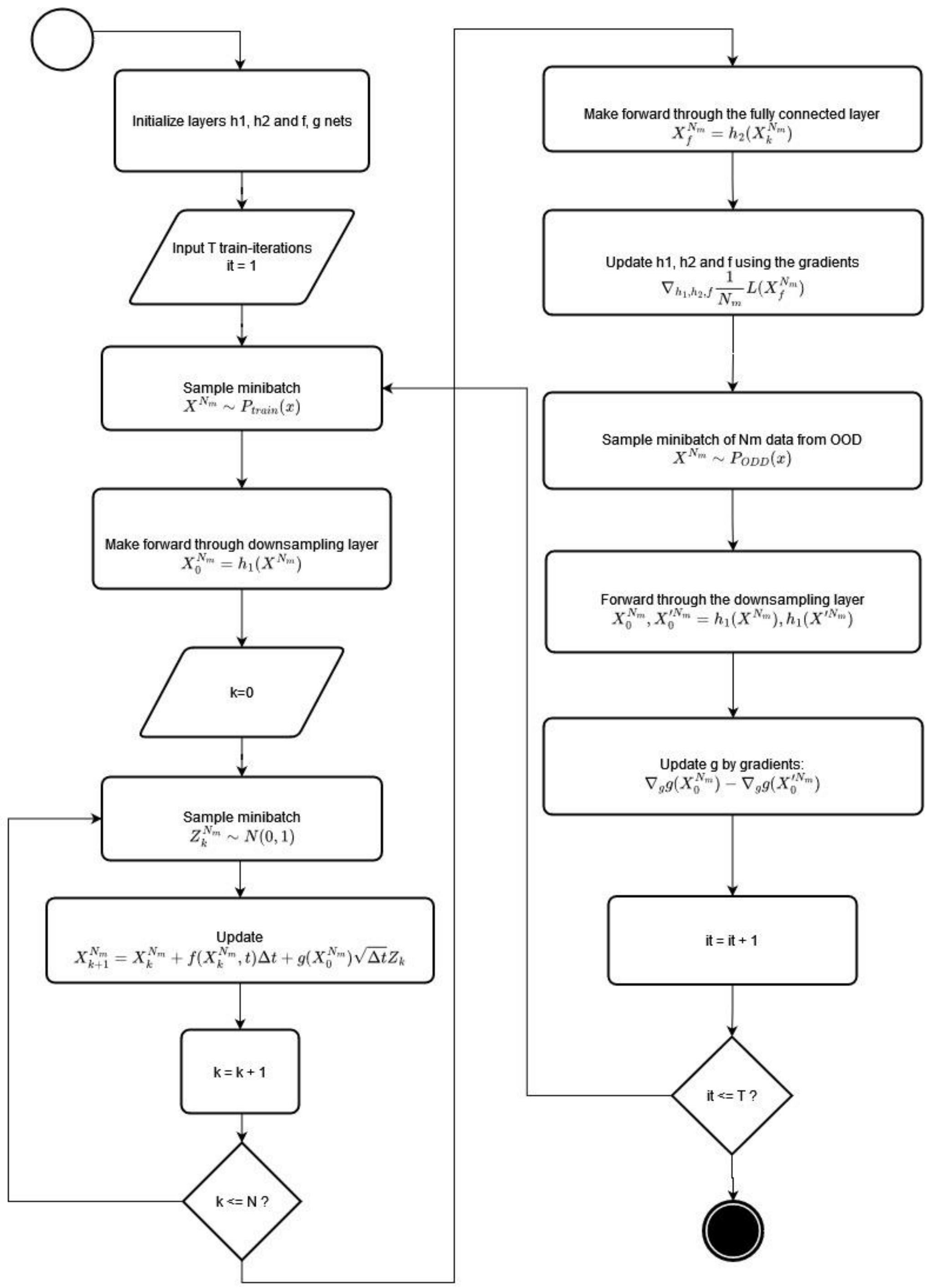
2.4. SDE-Net Training

Pseudocode for Training Algorithm of SDE-Net

Algorithm 1 Training of SDE-Net. h_1 is the downsampling layer; h_2 is the fully connected layer; f and g are the drift net and diffusion net; L is the loss function.

```
Initialize $h_1, f, g$ and $h_2$
for # training iterations do
    Sample minibatch of $N_M$ data from in-distribution: $\mathbf{X}^{N_M} \sim p_{\text{train}}(x)$
    Forward through the downsampling layer: $\mathbf{X}_0^{N_M} = h_1(\mathbf{X}^{N_M})$
    Forward through the SDE-Net block:
    for $k = 0$ to $N - 1$ do
        Sample $\mathbf{Z}_k^{N_M} \sim \mathcal{N}(0, \mathbf{I})$
        $\mathbf{X}_{k+1}^{N_M} = \mathbf{X}_k^{N_M} + f(\mathbf{X}_k^{N_M}, t)\Delta t + g(\mathbf{X}_0^{N_M})\sqrt{\Delta t}\mathbf{Z}_k$
    end for
    Forward through the fully connected layer: $\mathbf{X}_{\text{f}}^{N_M} = h_2(\mathbf{X}_k^{N_M})$
    Update $h_1, h_2$ and $f$ by $\nabla_{h_1,h_2,f}\frac{1}{N_M}L(\mathbf{X}_{\text{f}}^{N_M})$
    Sample minibatch of $N_M$ data from out-of-distribution: $\tilde{\mathbf{X}}^{N_M} \sim p_{\text{OOD}}(x)$
    Forward through the downsampling layer: $\mathbf{X}_0^{N_M}, \tilde{\mathbf{X}}_0^{N_M} = h_1(\mathbf{X}^{N_M}), h_1(\tilde{\mathbf{X}}^{N_M})$
    Update $g$ by $\nabla_g g(\mathbf{X}_0^{N_M}) - \nabla_g g(\tilde{\mathbf{X}}_0{}^{N_M})$
end for
```

Flux Diagram for Training Algorithm of SDE-Net


Initialize layers h1, h2 and f, g nets
Input T train-iterations it = 1
Sample minibatch $X^{N_m} \sim P_{train}(x)$
Make forward through downsampling layer $X_0^{N_m} = h_1(X^{N_m})$
k=0
Sample minibatch $Z_k^{N_m} \sim N(0,1)$
Update $X_{k+1}^{N_m} = X_k^{N_m} + f(X_k^{N_m}, t)\Delta t + g(X_0^{N_m})\sqrt{\Delta t} Z_k$
k = k + 1
k <= N ?
Make forward through the fully connected layer $X_f^{N_m} = h_2(X_k^{N_m})$
Update h1, h2 and f using the gradients $\nabla_{h_1,h_2,f} \frac{1}{N_m} L(X_f^{N_m})$
Sample minibatch of Nm data from OOD $X^{N_m} \sim P_{ODD}(x)$
Forward through the downsampling layer $X_0^{N_m}, X_0'^{N_m} = h_1(X^{N_m}), h_1(X'^{N_m})$
Update g by gradients: $\nabla_g g(X_0^{N_m}) - \nabla_g g(X_0'^{N_m})$
it = it + 1
it <= T ?

Objective Function

$$\min_{\theta_f} E_{x_0 \sim P_{train}} E(L(x_t)) + \min_{\theta_g} E_{x_0 \sim P_{train}} g(x_0, \theta_g) + \max_{\theta_g} E_{x_0' \sim POOD}\, g(x_0', \theta_g)$$

Adaptative step

$x_{k+1} = x_k + f(x_k, t, \theta_f)\Delta t + g(x_0, \theta_g)\sqrt{\Delta t} Z_k$, where $\Delta t = T / N$

3. Experimental Results

3.1. Schema of experiments

Was realized the next experiments:

a. Out-of-distribution detection
b. Misclassification detection
c. Adversarial sample detection
d. Active learning

3.2. Description of the datasets

For the experiments, was used 2 datasets:

1) MNIST: The MNIST database for classification of handwritten digits has a training set of 60 000 examples and a test set of 10 000 examples.

2) SVHN: Street view house numbers database for classification contain over 600k labelled real-world images of house numbers taken from Google Street view. This dataset is structured with 26 032 digits for testing, 73 257 digits for training and 531 131 as extra training data.

3) Year Prediction MSD: regression dataset for prediction of the release year of a song from 90 audio features. Training set have 463 715 examples and test set 51 630.

3.3. Description of the graphics

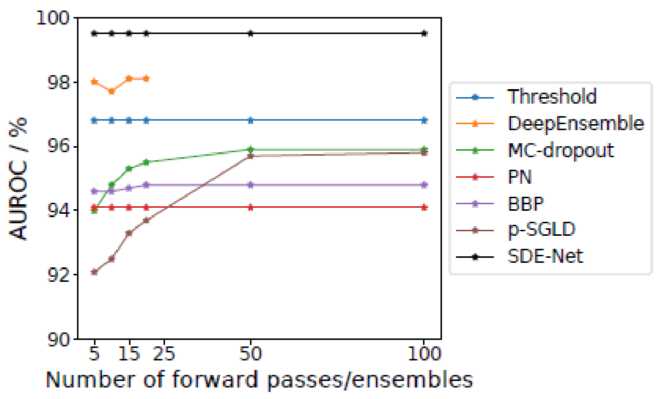


Figure 1. OOD Detection for classification

Effect number of forward passes/ ensembles on out-of-distribution (OOD) detection. Was used MNIST as ID data and SVHN as the OOD data. We can see the BNNs (MC-dropout, p-SGLD and BBP) requires more samples than SDE-Net to reach their peak performance at test time. In the case of the Deep Ensemble, the performance is saturated using 5 nets. Larger ensemble implies in losing of performance.

ID	OOD	Model	# Parameters	Classification accuracy	TNR at TPR 95%	AUROC	Detection accuracy	AUPR in	AUPR out
MNIST	SEMEION	Threshold	0.58M	99.5 ± 0.0	94.0 ± 1.4	98.3 ± 0.3	94.8 ± 0.7	99.7 ± 0.1	89.4 ± 1.1
		DeepEnsemble	$0.58M \times 5$	$\mathbf{99.6} \pm$ NA	$96.0 \pm$ NA	$98.8 \pm$ NA	$95.8 \pm$ NA	$99.8 \pm$ NA	$91.3 \pm$ NA
		MC-dropout	0.58M	99.5 ± 0.0	92.9 ± 1.6	97.6 ± 0.5	94.2 ± 0.7	99.6 ± 0.1	88.5 ± 1.7
		PN	0.58M	99.3 ± 0.1	93.4 ± 2.2	96.1 ± 1.2	94.5 ± 1.1	98.4 ± 0.7	88.5 ± 1.3
		BBP	1.02M	99.2 ± 0.3	75.0 ± 3.4	94.8 ± 1.2	90.4 ± 2.2	99.2 ± 0.3	76.0 ± 4.2
		p-SGLD	0.58M	99.3 ± 0.2	85.3 ± 2.3	89.1 ± 1.6	90.5 ± 1.3	93.6 ± 1.0	82.8 ± 2.2
		SDE-Net	0.28M	99.4 ± 0.1	$\mathbf{99.6} \pm 0.2$	$\mathbf{99.9} \pm 0.1$	$\mathbf{98.6} \pm 0.5$	$\mathbf{100.0} \pm 0$	$\mathbf{99.5} \pm 0.3$
MNIST	SVHN	Threshold	0.58M	99.5 ± 0.0	90.1 ± 2.3	96.8 ± 0.9	92.9 ± 1.1	90.0 ± 3.5	98.7 ± 0.3
		DeepEnsemble	$0.58M \times 5$	$\mathbf{99.6} \pm$ NA	$92.7 \pm$ NA	$98.0 \pm$ NA	$94.1 \pm$ NA	$94.5 \pm$ NA	$99.1 \pm$ NA
		MC-dropout	0.58M	99.5 ± 0.0	88.7 ± 0.6	95.9 ± 0.4	92.0 ± 0.3	87.6 ± 2.0	98.4 ± 0.1
		PN	0.58 M	99.3 ± 0.1	90.4 ± 2.8	94.1 ± 2.2	93.0 ± 1.4	73.2 ± 7.3	98.0 ± 0.6
		BBP	1.02M	99.2 ± 0.3	80.5 ± 3.2	96.0 ± 1.1	91.9 ± 0.9	92.6 ± 2.4	98.3 ± 0.4
		p-SGLD	0.58M	99.3 ± 0.2	94.5 ± 2.1	95.7 ± 1.3	95.0 ± 1.2	75.6 ± 5.2	98.7 ± 0.2
		SDE-Net	0.28M	99.4 ± 0.1	$\mathbf{97.8} \pm 1.1$	$\mathbf{99.5} \pm 0.2$	$\mathbf{97.0} \pm 0.2$	$\mathbf{98.6} \pm 0.6$	$\mathbf{99.8} \pm 0.1$
SVHN	CIFAR10	Threshold	0.58M	95.2 ± 0.1	66.1 ± 1.9	94.4 ± 0.4	89.8 ± 0.5	96.7 ± 0.2	84.6 ± 0.8
		DeepEnsemble	$0.58M \times 5$	$\mathbf{95.4} \pm$ NA	$66.5 \pm$ NA	$94.6 \pm$ NA	$90.1 \pm$ NA	$97.8 \pm$ NA	$84.8 \pm$ NA
		MC-dropout	0.58M	95.2 ± 0.1	66.9 ± 0.6	94.3 ± 0.1	89.8 ± 0.2	97.6 ± 0.1	84.8 ± 0.2
		PN	0.58M	95.0 ± 0.1	66.9 ± 2.0	89.9 ± 0.6	87.4 ± 0.6	92.5 ± 0.6	82.3 ± 0.9
		BBP	1.02M	93.3 ± 0.6	42.2 ± 1.2	90.4 ± 0.3	83.9 ± 0.4	96.4 ± 0.2	73.9 ± 0.5
		p-SGLD	0.58M	94.1 ± 0.5	63.5 ± 0.9	94.3 ± 0.4	87.8 ± 1.2	97.9 ± 0.2	83.9 ± 0.7
		SDE-Net	0.32 M	94.2 ± 0.2	$\mathbf{87.5} \pm 2.8$	$\mathbf{97.8} \pm 0.4$	$\mathbf{92.7} \pm 0.7$	$\mathbf{99.2} \pm 0.2$	$\mathbf{93.7} \pm 0.9$
SVHN	CIFAR100	Threshold	0.58M	95.2 ± 0.1	64.6 ± 1.9	93.8 ± 0.4	88.3 ± 0.4	97.0 ± 0.2	83.7 ± 0.8
		DeepEnsemble	$0.58M \times 5$	$\mathbf{95.4} \pm$ NA	$64.4 \pm$ NA	$93.9 \pm$ NA	$89.4 \pm$ NA	$97.4 \pm$ NA	$84.8 \pm$ NA
		MC-dropout	0.58M	95.2 ± 0.1	65.5 ± 1.1	93.7 ± 0.2	89.3 ± 0.3	97.1 ± 0.2	83.9 ± 0.4
		PN	0.58M	95.0 ± 0.1	65.8 ± 1.7	89.1 ± 0.8	86.6 ± 0.7	91.8 ± 0.8	81.6 ± 1.1
		BBP	1.02M	93.3 ± 0.6	42.4 ± 0.3	90.6 ± 0.2	84.3 ± 0.3	96.5 ± 0.1	75.2 ± 0.9
		p-SGLD	0.58M	94.1 ± 0.5	62.0 ± 0.5	91.3 ± 1.2	86.0 ± 0.2	93.1 ± 0.8	81.9 ± 1.3
		SDE-Net	0.32M	94.2 ± 0.2	$\mathbf{83.4} \pm 3.6$	$\mathbf{97.0} \pm 0.4$	$\mathbf{91.6} \pm 0.7$	$\mathbf{98.8} \pm 0.1$	$\mathbf{92.3} \pm 1.1$

Table 1. Classification and out-of-distribution detection results on MNIST and SVHN

In the case of the misclassification detection, the model uses the predictive uncertainty to identify test samples on which the model has misclassified.

Data	Model	AUROC	AUPR succ	AUPR err
MNIST	Threshold	94.3 ± 0.9	99.8 ± 0.1	31.9 ± 8.3
	DeepEnsemble	$\mathbf{97.5} \pm$ NA	$\mathbf{100.0} \pm$ NA	$41.4 \pm$ NA
	MC-dropout	95.8 ± 1.3	99.9 ± 0.0	33.0 ± 6.7
	PN	91.8 ± 0.7	99.8 ± 0.0	33.4 ± 4.6
	BBP	96.5 ± 2.1	$\mathbf{100.0} \pm 0.0$	35.4 ± 3.2
	P-SGLD	96.4 ± 1.7	$\mathbf{100.0} \pm 0.0$	$\mathbf{42.0} \pm 2.4$
	SDE-Net	96.8 ± 0.9	$\mathbf{100.0} \pm 0.0$	36.6 ± 4.6
SVHN	Threshold	90.1 ± 0.3	99.3 ± 0.0	42.8 ± 0.6
	DeepEnsemble	$91.0 \pm$ NA	$\mathbf{99.4} \pm$ NA	$46.5 \pm$ NA
	MC-dropout	90.4 ± 0.6	99.3 ± 0.0	45.0 ± 1.2
	PN	84.0 ± 0.4	98.2 ± 0.2	43.9 ± 1.1
	BBP	91.8 ± 0.2	99.1 ± 0.1	50.7 ± 0.9
	P-SGLD	$\mathbf{93.0} \pm 0.4$	$\mathbf{99.4} \pm 0.1$	48.6 ± 1.8
	SDE-Net	92.3 ± 0.5	$\mathbf{99.4} \pm 0.0$	$\mathbf{53.9} \pm 2.5$

Table 2. Misclassification detection performance on MNIST and SVHN

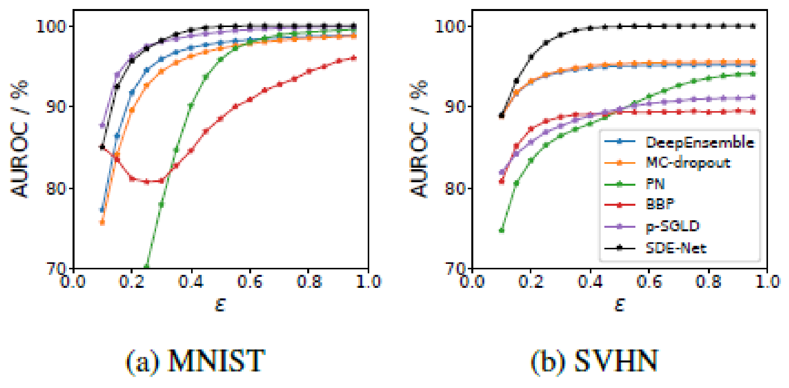


Figure 2. Performance of adversarial sample detection under FSGM attacks

Show the performance of different models when facing FGSM (Fast Gradient Sign Method) attacks. This experiment is important because FGSM highlight the vulnerability of machine learning models to adversarial examples through perturbations in the input data that is classified differently by the machine learning model than the original input was. In the experiment, we can see that SDE-Net is more robust model and resistant to this FGSM attacks in comparison with other models as the number of step size increases and increases the magnitude of the perturbation.

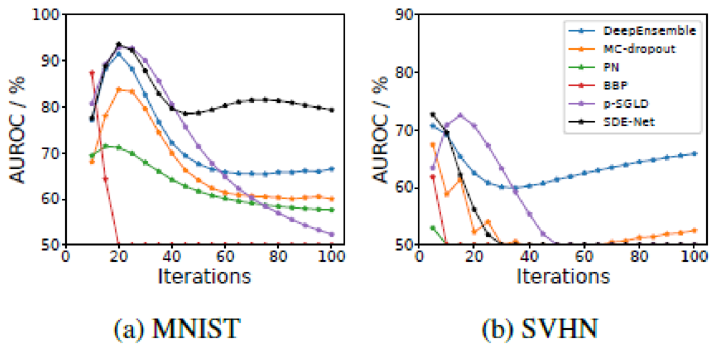


Figure 3. Performance of adversarial example detection under PGD attacks

Show the performance of different models when facing PGD (Projected Gradient Descent) attacks. This experiment considering the one of most powerful methods for generating robust adversarial examples. After multiple iterations of training under this PGD attack, for our models detect and classify correctly was much more difficult, and we can see good performance of SDE-Net in this scenario for MNIST dataset, but not enough in the case of SVHN, seeing a better performance of Deep Ensemble method and overconfidence in the SDE-Net and in the other models.

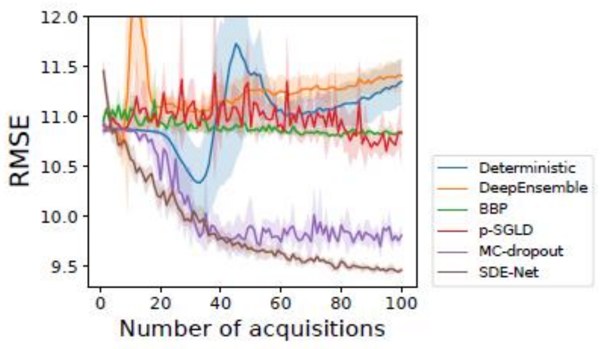
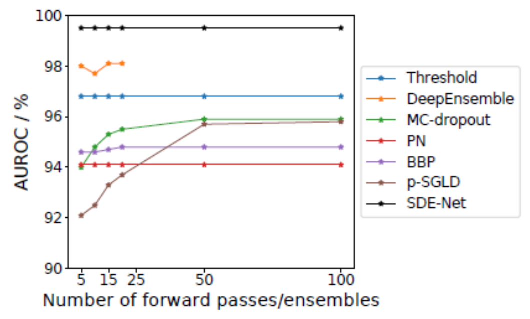
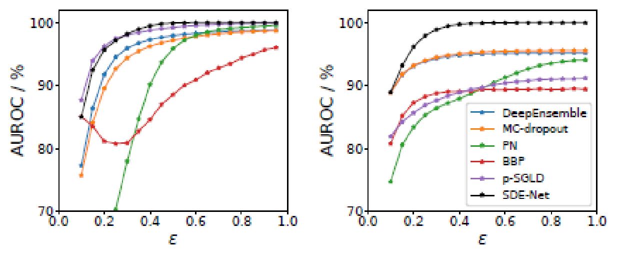


Figure 4. Performance of different models for active learning on Year Prediction MSD dataset

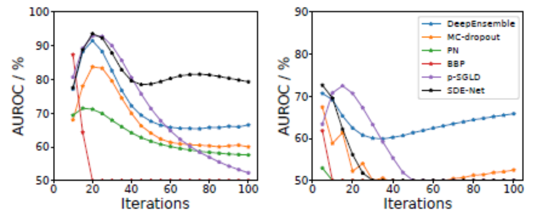
We can see in this active learning task, when much more labeled data the algorithm acquires, the RMSE (Root-Mean Square Error) consistently decreases and in the case of the SDE-Net, outperforms the other methods. Is interesting to note that the acquiring of new examples worsens the performance in the case of Deep Ensemble and Deterministic models, in others maintain and just the MC-dropout can achieve some comparative results respect SDE-Net.

3.4. Comparison between graphics

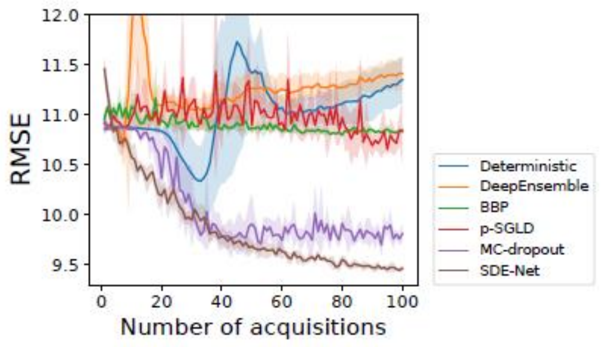




(a) MNIST (b) SVHN



(a) MNIST (b) SVHN



In the similarities between these experiments, we can see constantly that SDE-Net outperforms the other methods in the tasks where the uncertainty plays an important role. In other hand, the difference stays in each different task. In the first is focused in OOD detection, in the second in the reliable of the model under the FGSM attacks for adversarial sample detection, in the third, the same task but under PGD attacks, and in the last, the capacity of the algorithm improves its RMSE score when acquires new samples for training in the context of the regression task.

4. Software Tools

All the experiments were executed in GPU cluster tornado-k40 of Supercomputer Center of Polytech University.

The software tools used are:

- Programming language: Python in version 3.8.16
- Package manager: Anaconda 4.8 and pip 23.1.2
- Deep Learning framework: Pytorch
- Packages used: torch 2.0.1, torchvision 0.15.2, transformers 4.29.2

5. Execution of the Experiments

All the experiments were executed in the environment of Polytech's University Supercomputer center in cluster tornado-k40. The processes of training and test in the different tasks are divided by:

Classification Tasks:

a. Experiments for MNIST
    - Training and evaluating for Resnet
    - Training and evaluating for MC-dropout
    - Training and evaluating for SDE-Net
b. Experiments for SVHN
    - Training and evaluating for Resnet
    - Training and evaluating for MC-dropout
    - Training and evaluating for SDE-Net

Regression Task:

c. Experiments for Year MSD dataset
    - Training and evaluating for MC-dropout
    - Training and evaluating for SDE-Net

The details for the process of execution for each experiment are detailed in the annexes.

6. Discussion

With the implementation of the SDE-Net, was observed 3 main benefits:
(1) The aleatoric uncertainty and the epistemic uncertainty was explicitly separated in the model. The aleatoric uncertainty is naturally in the noise of data, and epistemic uncertainty was expressed by the introduction of the Brownian motion term in the model.
(2) It's not necessary define model priors and infer posterior probability distributions and it's straight-forward to implement.
(3) Available to use to classification and regression tasks.

7. Conclusion

(1) The model of SDE-Net can separate the different sources of uncertainty in comparison with existing non-Bayesian methods and in more straight-forward way in comparison with Bayesian methods.
(2) Experimentally was demonstrated that SDE-net outperforms state-of-art techniques the quantification of uncertainty and achieves robustness under different scenarios where uncertainty plays an important role.
(3) This research successfully established the connection between stochastic dynamical systems and neural networks for uncertainty quantification.
(4) In future directions, this approach has the potential to solve the problem of overconfidence under uncertainty scenarios in models which stay present in different deep learning applications.

8. References

[1] Kong, L., Sun, J., & Zhang, C. (2020). Sde-net: Equipping deep neural networks with uncertainty estimates. arXiv preprint arXiv:2008.10546.

[2] Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015, June). Weight uncertainty in neural network. In International conference on machine learning (pp. 1613-1622). PMLR.

[3] Hafner, D., Tran, D., Lillicrap, T., Irpan, A., & Davidson, J. (2020, August). Noise contrastive priors for functional uncertainty. In Uncertainty in Artificial Intelligence (pp. 905-914). PMLR.

[4] Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30.

[5] Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. Advances in neural information processing systems, 31.

[6] PyTorch. (n.d.). PyTorch Documentation. Retrieved May 28, 2023, from https://pytorch.org/docs/stable/index.html

9. Annexes

Drift neural network

```
class Drift(nn.Module):

    def __init__(self, dim):
        super(Drift, self).__init__()
        self.norm1 = norm(dim)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm2 = norm(dim)
        self.conv2 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm3 = norm(dim)

    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.norm3(out)
        return out
```

Diffusion neural network for MNIST

```
class Diffusion(nn.Module):
    def __init__(self, dim_in, dim_out):
        super(Diffusion, self).__init__()
        self.norm1 = norm(dim_in)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm2 = norm(dim_in)
        self.conv2 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.fc = nn.Sequential(norm(dim_out), nn.ReLU(inplace=True),
nn.AdaptiveAvgPool2d((1, 1)), Flatten(),
                                nn.Linear(dim_out, 1), nn.Sigmoid())
    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.fc(out)
        return out
```

Diffusion neural network for SVHN

```
class Diffusion(nn.Module):
    def __init__(self, dim_in, dim_out):
        super(Diffusion, self).__init__()
        self.norm1 = norm(dim_in)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm2 = norm(dim_in)
        self.conv2 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.norm3 = norm(dim_in)
        self.conv3 = ConcatConv2d(dim_in, dim_out, 3, 1, 1)
        self.fc = nn.Sequential(norm(dim_out), nn.ReLU(inplace=True),
nn.AdaptiveAvgPool2d((1, 1)), Flatten(),
                                nn.Linear(dim_out, 1), nn.Sigmoid())
    def forward(self, t, x):
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.norm3(out)
        out = self.relu(out)
        out = self.conv3(t, out)
        out = self.fc(out)
        return out
```

SDE-Net

```
class SDENet(nn.Module):
    def __init__(self, layer_depth, num_classes=10, dim = 64):
        super(SDENet, self).__init__()
        self.layer_depth = layer_depth
        self.downsampling_layers = nn.Sequential(
            nn.Conv2d(3, dim, 3, 1),
            norm(dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(dim, dim, 4, 2, 1),
            norm(dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(dim, dim, 4, 2, 1),
        )
        self.drift = Drift(dim)
        self.diffusion = Diffusion(dim, dim)
        self.fc_layers = nn.Sequential(norm(dim), nn.ReLU(inplace=True),
nn.AdaptiveAvgPool2d((1, 1)), Flatten(),
                                       nn.Linear(dim, 10))
        self.deltat = 6./self.layer_depth
        self.apply(init_params)
        self.sigma = 50

    def forward(self, x, training_diffusion=False):
        out = self.downsampling_layers(x)
        if not training_diffusion:
            t = 0
            diffusion_term = self.sigma*self.diffusion(t, out)
            diffusion_term = torch.unsqueeze(diffusion_term, 2)
            diffusion_term = torch.unsqueeze(diffusion_term, 3)
            for i in range(self.layer_depth):
                t = 6*(float(i))/self.layer_depth
                out = out + self.drift(t, out)*self.deltat +
diffusion_term*math.sqrt(self.deltat)*torch.randn_like(out).to(x)
            final_out = self.fc_layers(out)
        else:
            t = 0
            final_out = self.diffusion(t, out.detach())

        return final_out
```

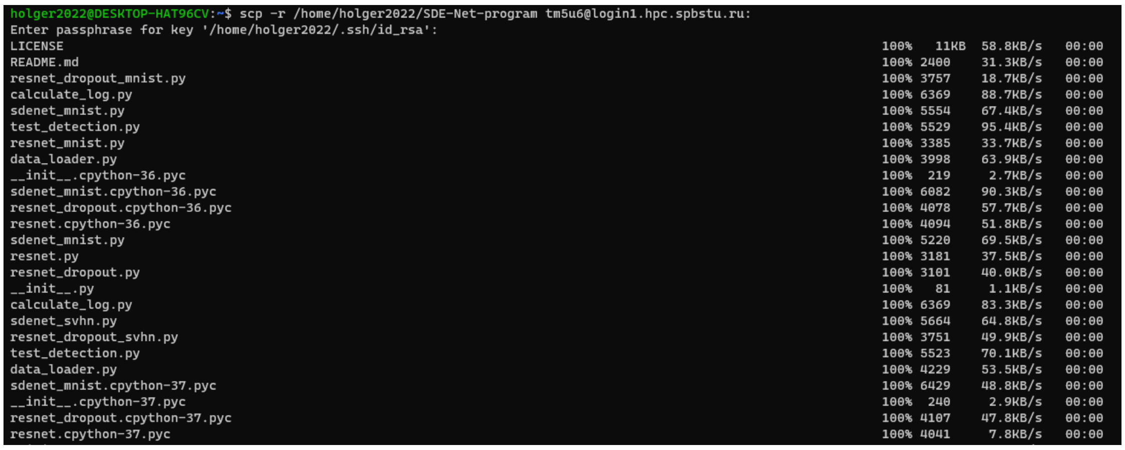
Experimental Results

- Prepare the GPU Environment

1) Path of the files used to build the experiments

```
holger2022@DESKTOP-HAT96CV:~$ tree /home/holger2022/SDE-Net-program
/home/holger2022/SDE-Net-program
├── LICENSE
├── MNIST
│   ├── calculate_log.py
│   ├── data_loader.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-36.pyc
│   │   │   ├── resnet.cpython-36.pyc
│   │   │   ├── resnet_dropout.cpython-36.pyc
│   │   │   └── sdenet_mnist.cpython-36.pyc
│   │   ├── resnet.py
│   │   ├── resnet_dropout.py
│   │   └── sdenet_mnist.py
│   ├── resnet_dropout_mnist.py
│   ├── resnet_mnist.py
│   ├── sdenet_mnist.py
│   └── test_detection.py
├── README.md
├── SVHN
│   ├── calculate_log.py
│   ├── data_loader.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-36.pyc
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── resnet.cpython-36.pyc
│   │   │   ├── resnet.cpython-37.pyc
│   │   │   ├── resnet_dropout.cpython-36.pyc
│   │   │   ├── resnet_dropout.cpython-37.pyc
│   │   │   ├── sdenet.cpython-36.pyc
│   │   │   ├── sdenet_mnist.cpython-36.pyc
│   │   │   └── sdenet_mnist.cpython-37.pyc
│   │   ├── resnet.py
│   │   ├── resnet_dropout.py
│   │   └── sdenet.py
│   ├── resnet_dropout_svhn.py
│   ├── resnet_svhn.py
│   ├── sdenet_svhn.py
│   └── test_detection.py
├── YearMSD
│   ├── DNN_mc.py
│   ├── SDE_regression.py
│   ├── calculate_log.py
│   ├── data_loader.py
│   ├── models
│   │   ├── DNN.py
│   │   ├── DNN_mcdropout.py
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── DNN.cpython-36.pyc
│   │   │   ├── __init__.cpython-36.pyc
│   │   │   └── sdenet.cpython-36.pyc
│   │   └── sdenet.py
│   ├── test_detection_mc.py
│   └── test_detection_sde.py
└── figure
    └── illustration.png
```

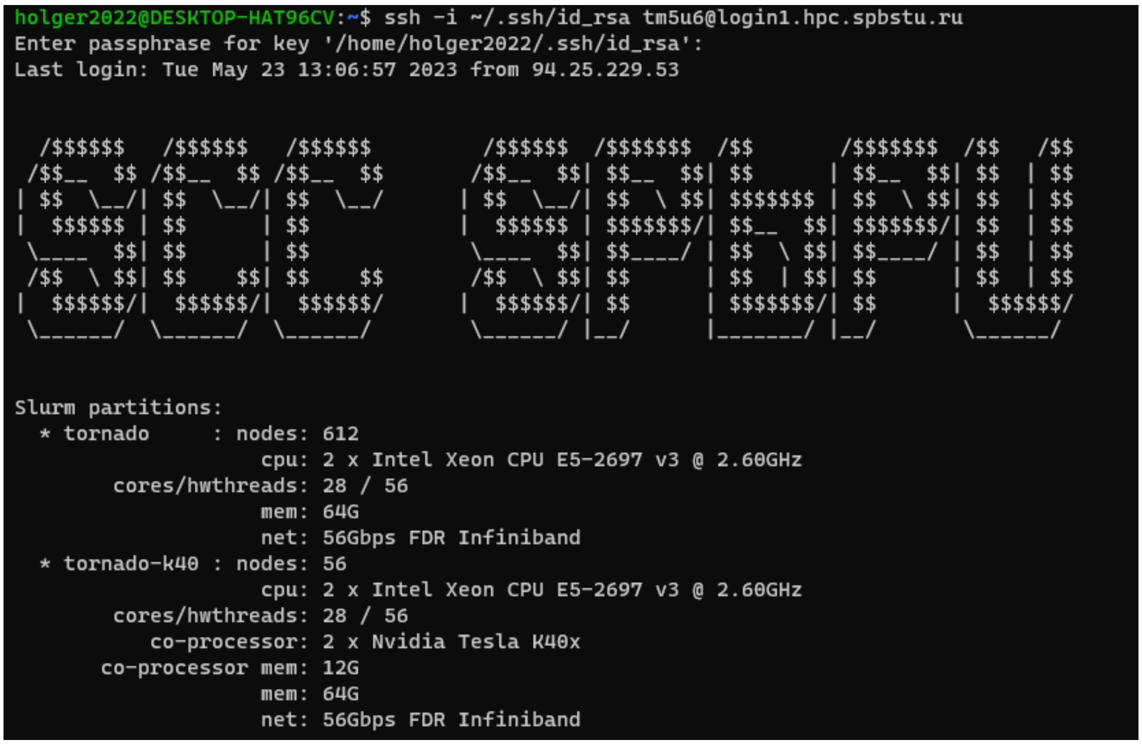
2) Copy files to storage of Polytech’s University Supercomputer Center



```
holger2022@DESKTOP-HAT96CV:~$ scp -r /home/holger2022/SDE-Net-program tm5u6@login1.hpc.spbstu.ru:
Enter passphrase for key '/home/holger2022/.ssh/id_rsa':
LICENSE                                                                          100%   11KB  58.8KB/s   00:00
README.md                                                                        100% 2400    31.3KB/s   00:00
resnet_dropout_mnist.py                                                          100% 3757    18.7KB/s   00:00
calculate_log.py                                                                 100% 6369    88.7KB/s   00:00
sdenet_mnist.py                                                                  100% 5554    67.4KB/s   00:00
test_detection.py                                                                100% 5529    95.4KB/s   00:00
resnet_mnist.py                                                                  100% 3385    33.7KB/s   00:00
data_loader.py                                                                   100% 3998    63.9KB/s   00:00
__init__.cpython-36.pyc                                                          100%  219     2.7KB/s   00:00
sdenet_mnist.cpython-36.pyc                                                      100% 6082    90.3KB/s   00:00
resnet_dropout.cpython-36.pyc                                                    100% 4078    57.7KB/s   00:00
resnet.cpython-36.pyc                                                            100% 4094    51.8KB/s   00:00
sdenet_mnist.py                                                                  100% 5220    69.5KB/s   00:00
resnet.py                                                                        100% 3181    37.5KB/s   00:00
resnet_dropout.py                                                                100% 3101    40.0KB/s   00:00
__init__.py                                                                      100%   81     1.1KB/s   00:00
calculate_log.py                                                                 100% 6369    83.3KB/s   00:00
sdenet_svhn.py                                                                   100% 5664    64.8KB/s   00:00
resnet_dropout_svhn.py                                                           100% 3751    49.9KB/s   00:00
test_detection.py                                                                100% 5523    70.1KB/s   00:00
data_loader.py                                                                   100% 4229    53.5KB/s   00:00
sdenet_mnist.cpython-37.pyc                                                      100% 6429    48.8KB/s   00:00
__init__.cpython-37.pyc                                                          100%  240     2.9KB/s   00:00
resnet_dropout.cpython-37.pyc                                                    100% 4107    47.8KB/s   00:00
resnet.cpython-37.pyc                                                            100% 4041     7.8KB/s   00:00
```

Figure 1. Linux command to copy all files and directories of project to supercomputer

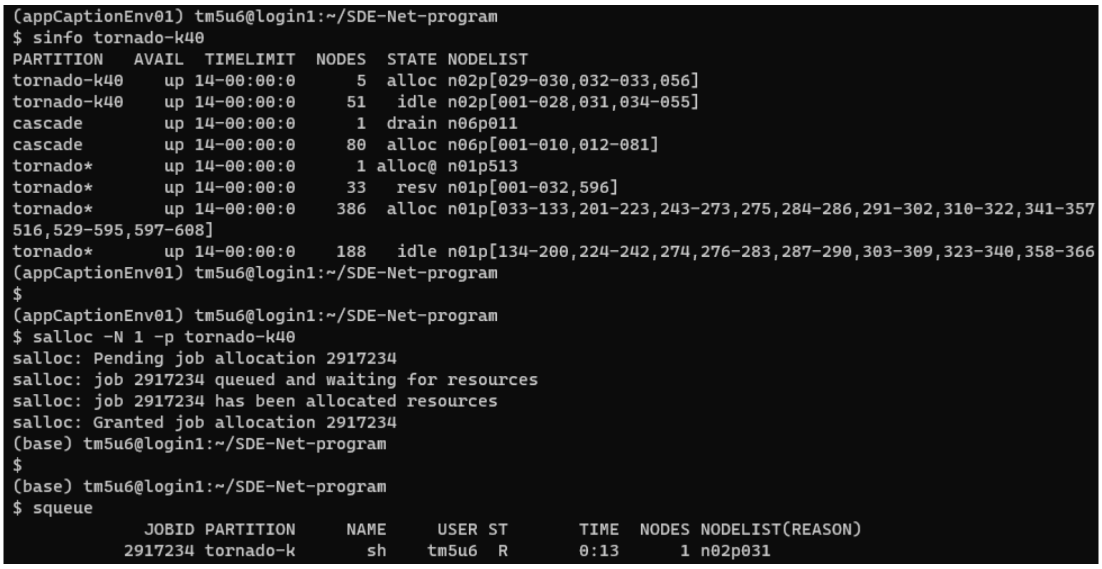
3) Connection via SSH with the supercomputer



```
holger2022@DESKTOP-HAT96CV:~$ ssh -i ~/.ssh/id_rsa tm5u6@login1.hpc.spbstu.ru
Enter passphrase for key '/home/holger2022/.ssh/id_rsa':
Last login: Tue May 23 13:06:57 2023 from 94.25.229.53


  /$$$$$$   /$$$$$$   /$$$$$$           /$$$$$$  /$$$$$$$  /$$       /$$$$$$$  /$$   /$$
 /$$__  $$ /$$__  $$ /$$__  $$         /$$__  $$| $$__  $$| $$      | $$__  $$| $$  | $$
| $$  \__/| $$  \__/| $$  \__/        | $$  \__/| $$  \ $$| $$$$$$$ | $$  \ $$| $$  | $$
|  $$$$$$ | $$      | $$              |  $$$$$$ | $$$$$$$/| $$__  $$| $$$$$$$/| $$  | $$
 \____  $$| $$      | $$               \____  $$| $$____/ | $$  \ $$| $$____/ | $$  | $$
 /$$  \ $$| $$    $$| $$    $$         /$$  \ $$| $$      | $$  | $$| $$      | $$  | $$
|  $$$$$$/|  $$$$$$/|  $$$$$$/        |  $$$$$$/| $$      | $$$$$$$/| $$      |  $$$$$$/
 \______/  \______/  \______/          \______/ |__/      |_______/ |__/       \______/


Slurm partitions:
  * tornado     : nodes: 612
                    cpu: 2 x Intel Xeon CPU E5-2697 v3 @ 2.60GHz
        cores/hwthreads: 28 / 56
                    mem: 64G
                    net: 56Gbps FDR Infiniband
  * tornado-k40 : nodes: 56
                    cpu: 2 x Intel Xeon CPU E5-2697 v3 @ 2.60GHz
        cores/hwthreads: 28 / 56
           co-processor: 2 x Nvidia Tesla K40x
       co-processor mem: 12G
                    mem: 64G
                    net: 56Gbps FDR Infiniband
```

Figure 2. Linux command to make SSH connection with SCC SPBPU

4) Allocate 1 node for GPU-cluster in SCC



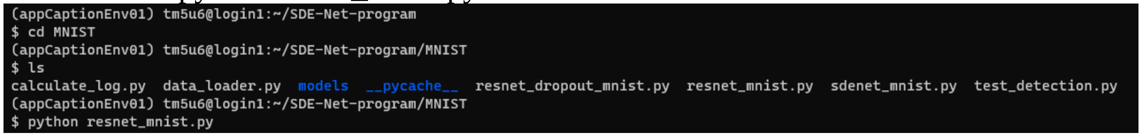
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program
$ sinfo tornado-k40
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
tornado-k40    up 14-00:00:0      5  alloc n02p[029-030,032-033,056]
tornado-k40    up 14-00:00:0     51   idle n02p[001-028,031,034-055]
cascade        up 14-00:00:0      1  drain n06p011
cascade        up 14-00:00:0     80  alloc n06p[001-010,012-081]
tornado*       up 14-00:00:0      1 alloc@ n01p513
tornado*       up 14-00:00:0     33   resv n01p[001-032,596]
tornado*       up 14-00:00:0    386  alloc n01p[033-133,201-223,243-273,275,284-286,291-302,310-322,341-357
516,529-595,597-608]
tornado*       up 14-00:00:0    188   idle n01p[134-200,224-242,274,276-283,287-290,303-309,323-340,358-366
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program
$
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program
$ salloc -N 1 -p tornado-k40
salloc: Pending job allocation 2917234
salloc: job 2917234 queued and waiting for resources
salloc: job 2917234 has been allocated resources
salloc: Granted job allocation 2917234
(base) tm5u6@login1:~/SDE-Net-program
$
(base) tm5u6@login1:~/SDE-Net-program
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2917234 tornado-k       sh    tm5u6  R       0:13      1 n02p031
```

Figure 3. Allocation of node in GPU Tesla k-40

EXPERIMENTS WITH MNIST

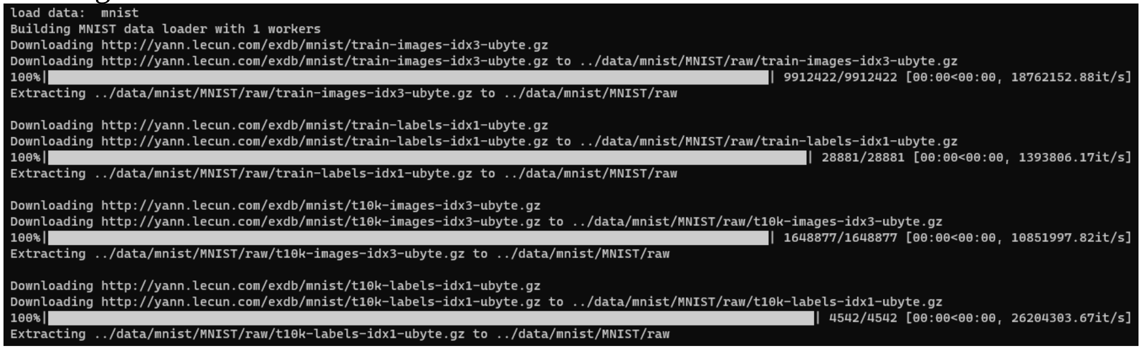
- Experiment with MNIST – Training Resnet

Command: $ python resnet_mnist.py



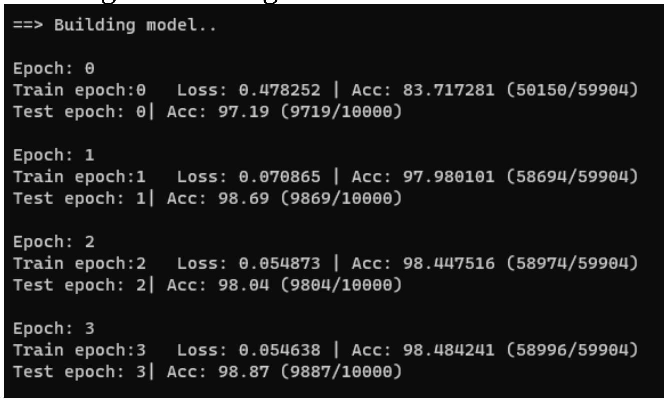
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program
$ cd MNIST
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ ls
calculate_log.py  data_loader.py  models  __pycache__  resnet_dropout_mnist.py  resnet_mnist.py  sdenet_mnist.py  test_detection.py
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ python resnet_mnist.py
```

Loading of dataset



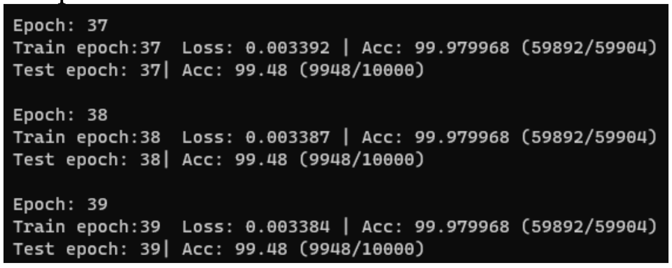
```
load data:  mnist
Building MNIST data loader with 1 workers
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/mnist/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████████████████████████████████████████████| 9912422/9912422 [00:00<00:00, 18762152.88it/s]
Extracting ../data/mnist/MNIST/raw/train-images-idx3-ubyte.gz to ../data/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/mnist/MNIST/raw/train-labels-idx1-ubyte.gz
100%|█████████████████████████████████████████████████████| 28881/28881 [00:00<00:00, 1393806.17it/s]
Extracting ../data/mnist/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████████████████████████████████████████████| 1648877/1648877 [00:00<00:00, 10851997.82it/s]
Extracting ../data/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|█████████████████████████████████████████████████████| 4542/4542 [00:00<00:00, 26204303.67it/s]
Extracting ../data/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/mnist/MNIST/raw
```

Building and training of the model



```
==> Building model..

Epoch: 0
Train epoch:0   Loss: 0.478252 | Acc: 83.717281 (50150/59904)
Test epoch: 0| Acc: 97.19 (9719/10000)

Epoch: 1
Train epoch:1   Loss: 0.070865 | Acc: 97.980101 (58694/59904)
Test epoch: 1| Acc: 98.69 (9869/10000)

Epoch: 2
Train epoch:2   Loss: 0.054873 | Acc: 98.447516 (58974/59904)
Test epoch: 2| Acc: 98.04 (9804/10000)

Epoch: 3
Train epoch:3   Loss: 0.054638 | Acc: 98.484241 (58996/59904)
Test epoch: 3| Acc: 98.87 (9887/10000)
```

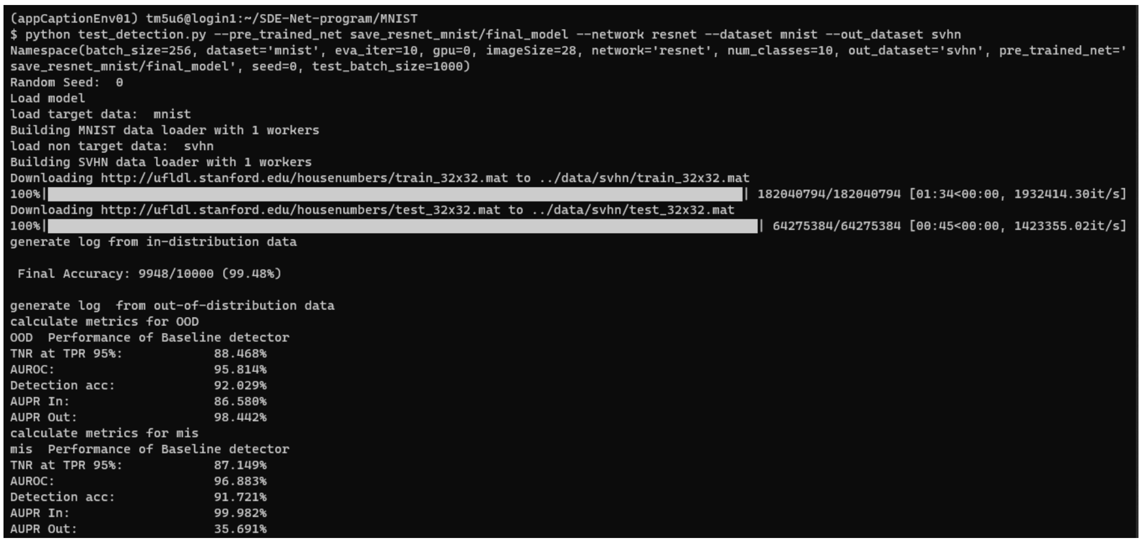
40 epochs after…



```
Epoch: 37
Train epoch:37  Loss: 0.003392 | Acc: 99.979968 (59892/59904)
Test epoch: 37| Acc: 99.48 (9948/10000)

Epoch: 38
Train epoch:38  Loss: 0.003387 | Acc: 99.979968 (59892/59904)
Test epoch: 38| Acc: 99.48 (9948/10000)

Epoch: 39
Train epoch:39  Loss: 0.003384 | Acc: 99.979968 (59892/59904)
Test epoch: 39| Acc: 99.48 (9948/10000)
```

- Experiment with MNIST - Evaluating Resnet

Command: $ python test_detection.py --pre_trained_net save_resnet_mnist/final_model --network resnet --dataset mnist --out_dataset svhn



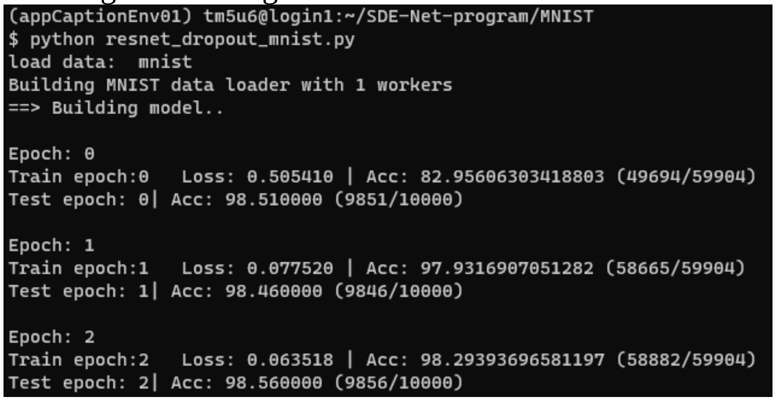
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ python test_detection.py --pre_trained_net save_resnet_mnist/final_model --network resnet --dataset mnist --out_dataset svhn
Namespace(batch_size=256, dataset='mnist', eva_iter=10, gpu=0, imageSize=28, network='resnet', num_classes=10, out_dataset='svhn', pre_trained_net='
save_resnet_mnist/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  mnist
Building MNIST data loader with 1 workers
load non target data:  svhn
Building SVHN data loader with 1 workers
Downloading http://ufldl.stanford.edu/housenumbers/train_32x32.mat to ../data/svhn/train_32x32.mat
100%|█████████████████████████████████████████████████████████████| 182040794/182040794 [01:34<00:00, 1932414.30it/s]
Downloading http://ufldl.stanford.edu/housenumbers/test_32x32.mat to ../data/svhn/test_32x32.mat
100%|██████████████████████████████████████████████████████████████| 64275384/64275384 [00:45<00:00, 1423355.02it/s]
generate log from in-distribution data

 Final Accuracy: 9948/10000 (99.48%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:            88.468%
AUROC:                     95.814%
Detection acc:             92.029%
AUPR In:                   86.580%
AUPR Out:                  98.442%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:            87.149%
AUROC:                     96.883%
Detection acc:             91.721%
AUPR In:                   99.982%
AUPR Out:                  35.691%
```

Figure 4. Final Performance results for Resnet in MNIST

- Experiment with MNIST – Training MC-dropout

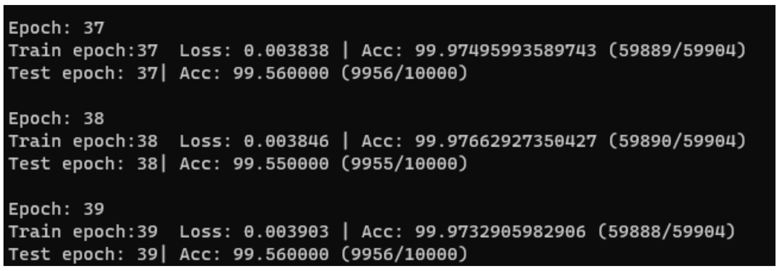
Command: $ python resnet_dropout_mnist.py

Building and training the model



```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ python resnet_dropout_mnist.py
load data:  mnist
Building MNIST data loader with 1 workers
==> Building model..

Epoch: 0
Train epoch:0   Loss: 0.505410 | Acc: 82.95606303418803 (49694/59904)
Test epoch: 0| Acc: 98.510000 (9851/10000)

Epoch: 1
Train epoch:1   Loss: 0.077520 | Acc: 97.9316907051282 (58665/59904)
Test epoch: 1| Acc: 98.460000 (9846/10000)

Epoch: 2
Train epoch:2   Loss: 0.063518 | Acc: 98.29393696581197 (58882/59904)
Test epoch: 2| Acc: 98.560000 (9856/10000)
```

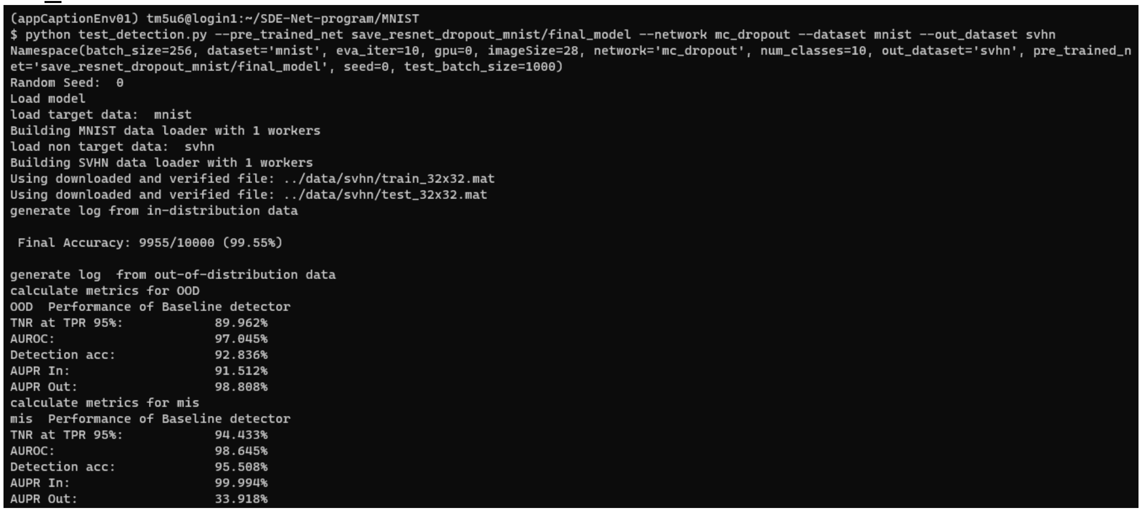
40 epochs after…



```
Epoch: 37
Train epoch:37  Loss: 0.003838 | Acc: 99.97495993589743 (59889/59904)
Test epoch: 37| Acc: 99.560000 (9956/10000)

Epoch: 38
Train epoch:38  Loss: 0.003846 | Acc: 99.97662927350427 (59890/59904)
Test epoch: 38| Acc: 99.550000 (9955/10000)

Epoch: 39
Train epoch:39  Loss: 0.003903 | Acc: 99.9732905982906 (59888/59904)
Test epoch: 39| Acc: 99.560000 (9956/10000)
```

- Experiment with MNIST – Evaluating MC-Dropout

Command: $ python test_detection.py --pre_trained_net save_resnet_dropout_mnist/final_model --network mc_dropout --dataset mnist --out_dataset svhn

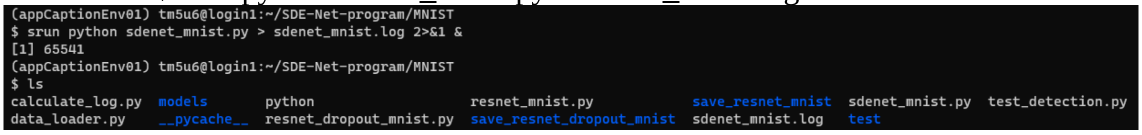


```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ python test_detection.py --pre_trained_net save_resnet_dropout_mnist/final_model --network mc_dropout --dataset mnist --out_dataset svhn
Namespace(batch_size=256, dataset='mnist', eva_iter=10, gpu=0, imageSize=28, network='mc_dropout', num_classes=10, out_dataset='svhn', pre_trained_n
et='save_resnet_dropout_mnist/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  mnist
Building MNIST data loader with 1 workers
load non target data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
generate log from in-distribution data

 Final Accuracy: 9955/10000 (99.55%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:            89.962%
AUROC:                     97.045%
Detection acc:             92.836%
AUPR In:                   91.512%
AUPR Out:                  98.808%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:            94.433%
AUROC:                     98.645%
Detection acc:             95.508%
AUPR In:                   99.994%
AUPR Out:                  33.918%
```

Figure 5. Final Performance results for MC-Dropout in MNIST

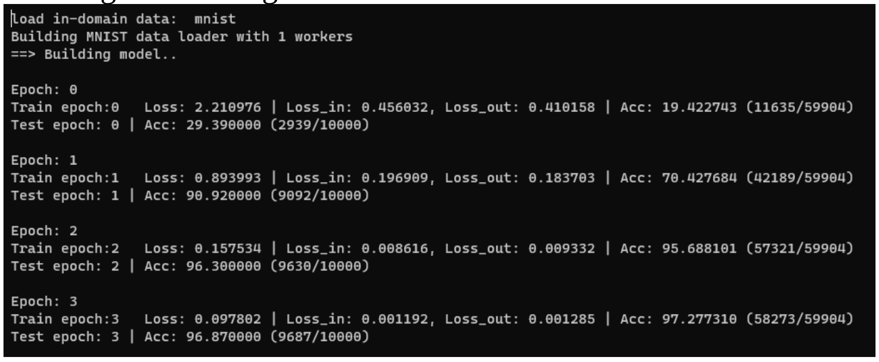
- Experiment with MNIST – Training the SDE-Net

Command: $ srun python sdenet_mnist.py > sdenet_mnist.log 2>&1 &



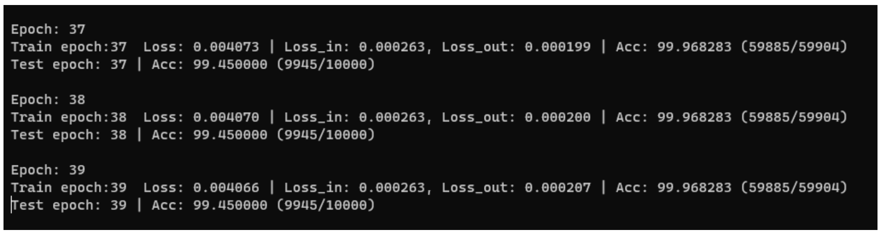
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ srun python sdenet_mnist.py > sdenet_mnist.log 2>&1 &
[1] 65541
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ ls
calculate_log.py  models       python                  resnet_mnist.py            save_resnet_mnist  sdenet_mnist.py  test_detection.py
data_loader.py    __pycache__  resnet_dropout_mnist.py  save_resnet_dropout_mnist  sdenet_mnist.log   test
```

Building and training the model



```
load in-domain data:  mnist
Building MNIST data loader with 1 workers
==> Building model..

Epoch: 0
Train epoch:0   Loss: 2.210976 | Loss_in: 0.456032, Loss_out: 0.410158 | Acc: 19.422743 (11635/59904)
Test epoch: 0 | Acc: 29.390000 (2939/10000)

Epoch: 1
Train epoch:1   Loss: 0.893993 | Loss_in: 0.196909, Loss_out: 0.183703 | Acc: 70.427684 (42189/59904)
Test epoch: 1 | Acc: 90.920000 (9092/10000)

Epoch: 2
Train epoch:2   Loss: 0.157534 | Loss_in: 0.008616, Loss_out: 0.009332 | Acc: 95.688101 (57321/59904)
Test epoch: 2 | Acc: 96.300000 (9630/10000)

Epoch: 3
Train epoch:3   Loss: 0.097802 | Loss_in: 0.001192, Loss_out: 0.001285 | Acc: 97.277310 (58273/59904)
Test epoch: 3 | Acc: 96.870000 (9687/10000)
```

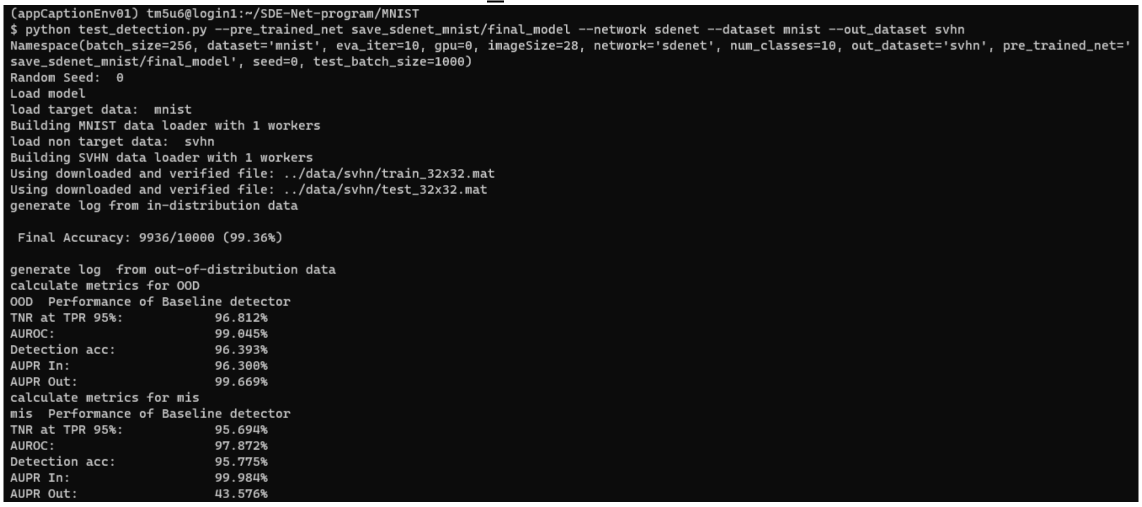
40 epochs after…



```
Epoch: 37
Train epoch:37  Loss: 0.004073 | Loss_in: 0.000263, Loss_out: 0.000199 | Acc: 99.968283 (59885/59904)
Test epoch: 37 | Acc: 99.450000 (9945/10000)

Epoch: 38
Train epoch:38  Loss: 0.004070 | Loss_in: 0.000263, Loss_out: 0.000200 | Acc: 99.968283 (59885/59904)
Test epoch: 38 | Acc: 99.450000 (9945/10000)

Epoch: 39
Train epoch:39  Loss: 0.004066 | Loss_in: 0.000263, Loss_out: 0.000207 | Acc: 99.968283 (59885/59904)
Test epoch: 39 | Acc: 99.450000 (9945/10000)
```

- Experiment with MNIST – Evaluating SDE-Net

Command: $ python test_detection.py --pre_trained_net save_sdenet_mnist/final_model --network sdenet --dataset mnist --out_dataset svhn



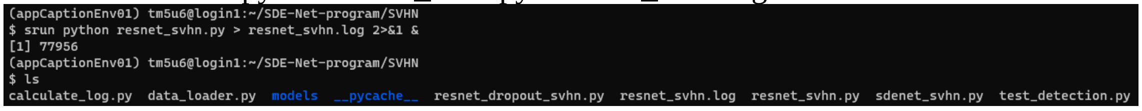
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/MNIST
$ python test_detection.py --pre_trained_net save_sdenet_mnist/final_model --network sdenet --dataset mnist --out_dataset svhn
Namespace(batch_size=256, dataset='mnist', eva_iter=10, gpu=0, imageSize=28, network='sdenet', num_classes=10, out_dataset='svhn', pre_trained_net='
save_sdenet_mnist/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  mnist
Building MNIST data loader with 1 workers
load non target data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
generate log from in-distribution data

 Final Accuracy: 9936/10000 (99.36%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:             96.812%
AUROC:                      99.045%
Detection acc:              96.393%
AUPR In:                    96.300%
AUPR Out:                   99.669%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:             95.694%
AUROC:                      97.872%
Detection acc:              95.775%
AUPR In:                    99.984%
AUPR Out:                   43.576%
```

Fig 6. Final performance of SDE-Net in MNIST

EXPERIMENTS WITH SVHN

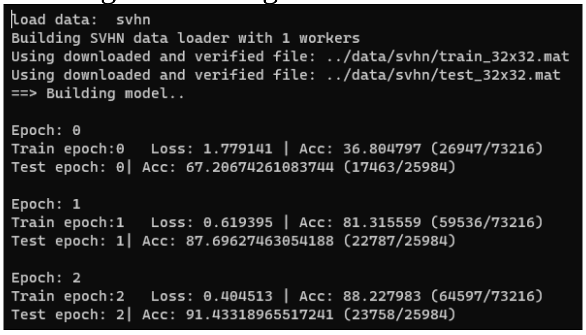
- Experiment with SVHN – Training the Resnet

Command: $ srun python resnet_svhn.py > resnet_svhn.log 2>&1 &



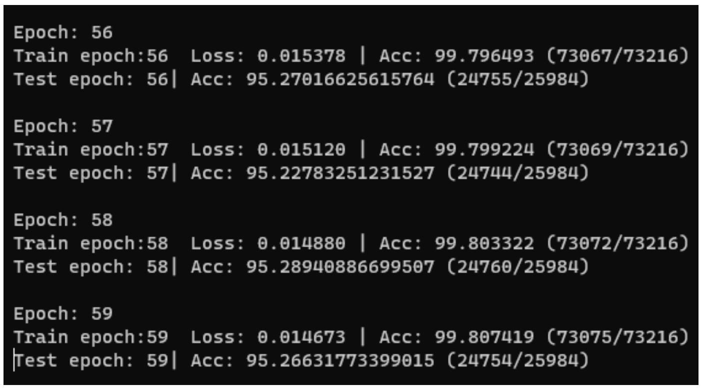
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ srun python resnet_svhn.py > resnet_svhn.log 2>&1 &
[1] 77956
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ ls
calculate_log.py  data_loader.py  models  __pycache__  resnet_dropout_svhn.py  resnet_svhn.log  resnet_svhn.py  sdenet_svhn.py  test_detection.py
```

Building and training the model



```
load data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
==> Building model..

Epoch: 0
Train epoch:0   Loss: 1.779141 | Acc: 36.804797 (26947/73216)
Test epoch: 0| Acc: 67.20674261083744 (17463/25984)

Epoch: 1
Train epoch:1   Loss: 0.619395 | Acc: 81.315559 (59536/73216)
Test epoch: 1| Acc: 87.69627463054188 (22787/25984)

Epoch: 2
Train epoch:2   Loss: 0.404513 | Acc: 88.227983 (64597/73216)
Test epoch: 2| Acc: 91.43318965517241 (23758/25984)
```

After 60 epochs…

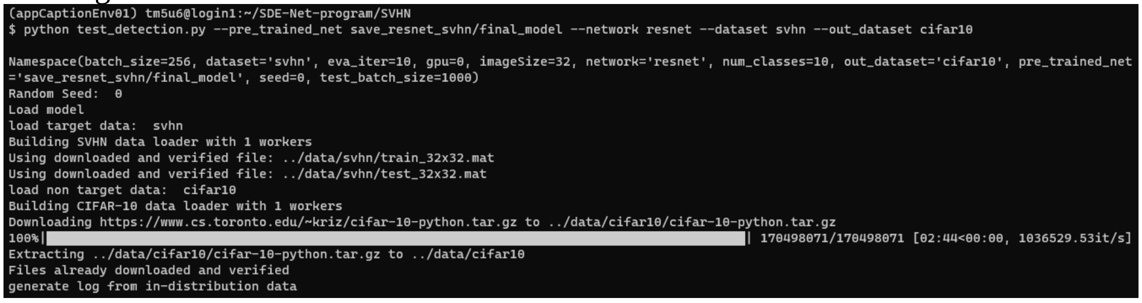


```
Epoch: 56
Train epoch:56  Loss: 0.015378 | Acc: 99.796493 (73067/73216)
Test epoch: 56| Acc: 95.27016625615764 (24755/25984)

Epoch: 57
Train epoch:57  Loss: 0.015120 | Acc: 99.799224 (73069/73216)
Test epoch: 57| Acc: 95.22783251231527 (24744/25984)

Epoch: 58
Train epoch:58  Loss: 0.014880 | Acc: 99.803322 (73072/73216)
Test epoch: 58| Acc: 95.28940886699507 (24760/25984)

Epoch: 59
Train epoch:59  Loss: 0.014673 | Acc: 99.807419 (73075/73216)
Test epoch: 59| Acc: 95.26631773399015 (24754/25984)
```

- Experiment with SVHN – Evaluating Resnet

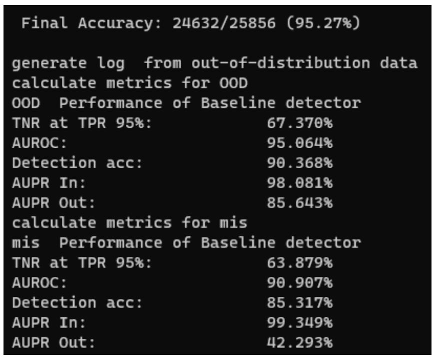
Command: $ python test_detection.py --pre_trained_net save_resnet_svhn/final_model --network resnet --dataset svhn --out_dataset cifar10

Loading CIFAR-10 dataset



```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ python test_detection.py --pre_trained_net save_resnet_svhn/final_model --network resnet --dataset svhn --out_dataset cifar10

Namespace(batch_size=256, dataset='svhn', eva_iter=10, gpu=0, imageSize=32, network='resnet', num_classes=10, out_dataset='cifar10', pre_trained_net
='save_resnet_svhn/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
load non target data:  cifar10
Building CIFAR-10 data loader with 1 workers
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ../data/cifar10/cifar-10-python.tar.gz
100%|████████████████████████████████████████████████████████████████| 170498071/170498071 [02:44<00:00, 1036529.53it/s]
Extracting ../data/cifar10/cifar-10-python.tar.gz to ../data/cifar10
Files already downloaded and verified
generate log from in-distribution data
```

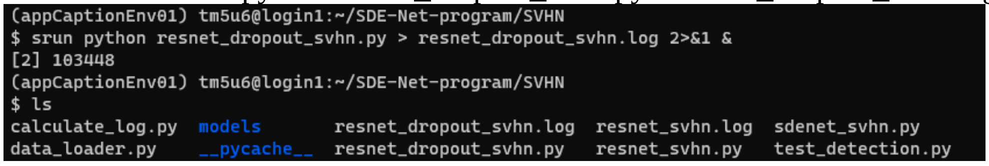
Evaluation



```
 Final Accuracy: 24632/25856 (95.27%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:            67.370%
AUROC:                     95.064%
Detection acc:             90.368%
AUPR In:                   98.081%
AUPR Out:                  85.643%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:            63.879%
AUROC:                     90.907%
Detection acc:             85.317%
AUPR In:                   99.349%
AUPR Out:                  42.293%
```

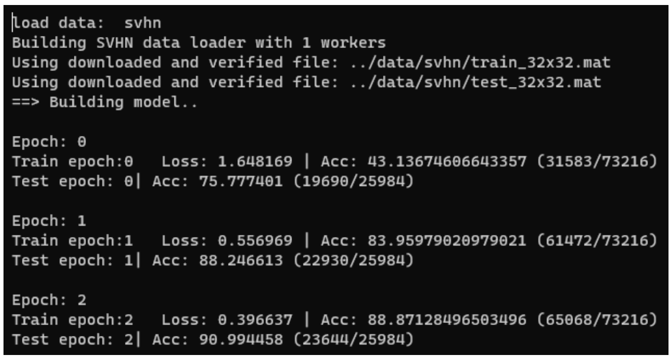
- Experiment with SVHN – Training the MC-dropout

Command: $ srun python resnet_dropout_svhn.py > resnet_dropout_svhn.log 2>&1 &



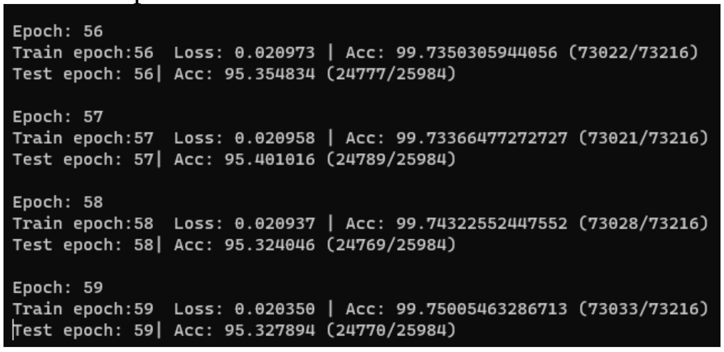
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ srun python resnet_dropout_svhn.py > resnet_dropout_svhn.log 2>&1 &
[2] 103448
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ ls
calculate_log.py  models       resnet_dropout_svhn.log  resnet_svhn.log  sdenet_svhn.py
data_loader.py    __pycache__  resnet_dropout_svhn.py   resnet_svhn.py   test_detection.py
```

Building and training the model



```
load data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
==> Building model..

Epoch: 0
Train epoch:0   Loss: 1.648169 | Acc: 43.13674606643357 (31583/73216)
Test epoch: 0| Acc: 75.777401 (19690/25984)

Epoch: 1
Train epoch:1   Loss: 0.556969 | Acc: 83.95979020979021 (61472/73216)
Test epoch: 1| Acc: 88.246613 (22930/25984)

Epoch: 2
Train epoch:2   Loss: 0.396637 | Acc: 88.87128496503496 (65068/73216)
Test epoch: 2| Acc: 90.994458 (23644/25984)
```

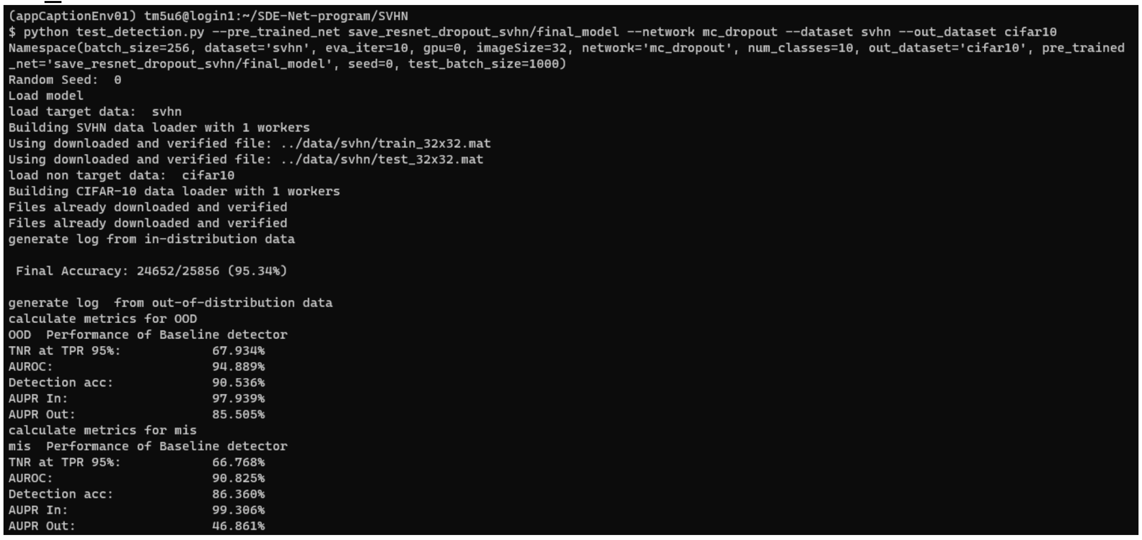
After 60 epochs…



```
Epoch: 56
Train epoch:56  Loss: 0.020973 | Acc: 99.7350305944056 (73022/73216)
Test epoch: 56| Acc: 95.354834 (24777/25984)

Epoch: 57
Train epoch:57  Loss: 0.020958 | Acc: 99.73366477272727 (73021/73216)
Test epoch: 57| Acc: 95.401016 (24789/25984)

Epoch: 58
Train epoch:58  Loss: 0.020937 | Acc: 99.74322552447552 (73028/73216)
Test epoch: 58| Acc: 95.324046 (24769/25984)

Epoch: 59
Train epoch:59  Loss: 0.020350 | Acc: 99.75005463286713 (73033/73216)
Test epoch: 59| Acc: 95.327894 (24770/25984)
```

- Experiment with SVHN – Evaluating MC-dropout

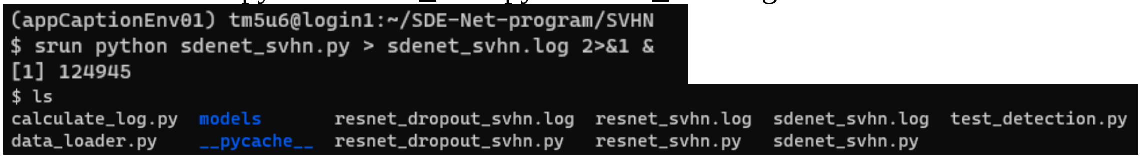
Command: $ python test_detection.py --pre_trained_net save_resnet_dropout_svhn/final_model --network mc_dropout --dataset svhn --out_dataset cifar10



```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ python test_detection.py --pre_trained_net save_resnet_dropout_svhn/final_model --network mc_dropout --dataset svhn --out_dataset cifar10
Namespace(batch_size=256, dataset='svhn', eva_iter=10, gpu=0, imageSize=32, network='mc_dropout', num_classes=10, out_dataset='cifar10', pre_trained
_net='save_resnet_dropout_svhn/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
load non target data:  cifar10
Building CIFAR-10 data loader with 1 workers
Files already downloaded and verified
Files already downloaded and verified
generate log from in-distribution data

 Final Accuracy: 24652/25856 (95.34%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:             67.934%
AUROC:                      94.889%
Detection acc:              90.536%
AUPR In:                    97.939%
AUPR Out:                   85.505%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:             66.768%
AUROC:                      90.825%
Detection acc:              86.360%
AUPR In:                    99.306%
AUPR Out:                   46.861%
```

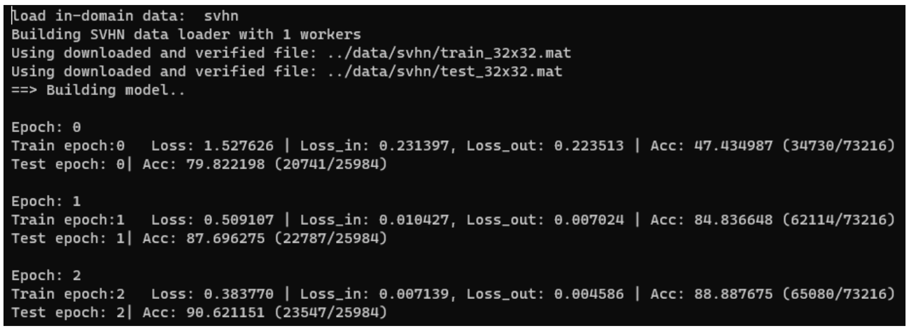
- Experiment with SVHN – Training the SDE-Net

Command: $ srun python sdenet_svhn.py > sdenet_svhn.log 2>&1 &



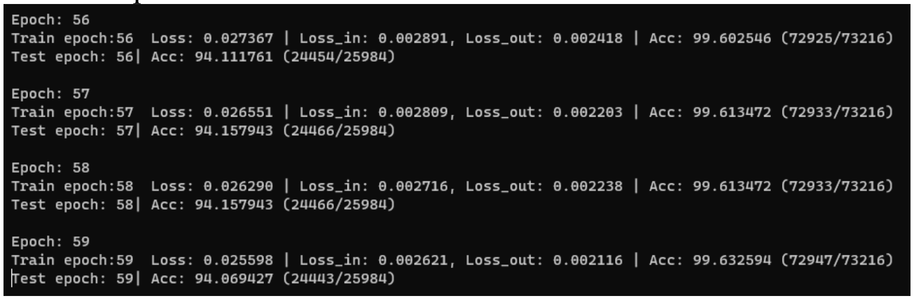
```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ srun python sdenet_svhn.py > sdenet_svhn.log 2>&1 &
[1] 124945
$ ls
calculate_log.py  models       resnet_dropout_svhn.log  resnet_svhn.log  sdenet_svhn.log  test_detection.py
data_loader.py    __pycache__  resnet_dropout_svhn.py   resnet_svhn.py   sdenet_svhn.py
```

Building and training the model



```
Load in-domain data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
==> Building model..

Epoch: 0
Train epoch:0   Loss: 1.527626 | Loss_in: 0.231397, Loss_out: 0.223513 | Acc: 47.434987 (34730/73216)
Test epoch: 0| Acc: 79.822198 (20741/25984)

Epoch: 1
Train epoch:1   Loss: 0.509107 | Loss_in: 0.010427, Loss_out: 0.007024 | Acc: 84.836648 (62114/73216)
Test epoch: 1| Acc: 87.696275 (22787/25984)

Epoch: 2
Train epoch:2   Loss: 0.383770 | Loss_in: 0.007139, Loss_out: 0.004586 | Acc: 88.887675 (65080/73216)
Test epoch: 2| Acc: 90.621151 (23547/25984)
```

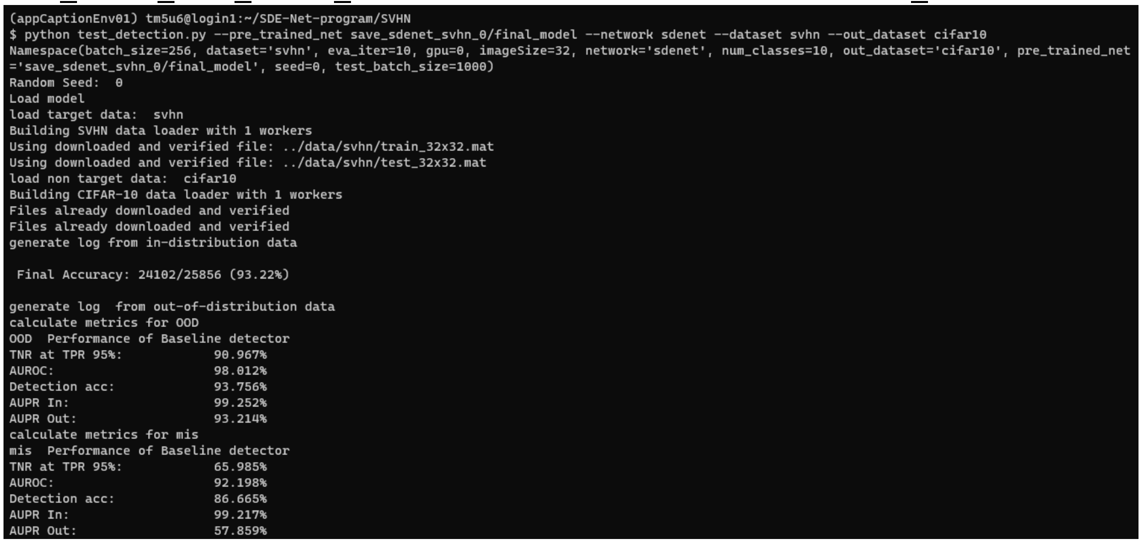
After 60 epochs…



```
Epoch: 56
Train epoch:56  Loss: 0.027367 | Loss_in: 0.002891, Loss_out: 0.002418 | Acc: 99.602546 (72925/73216)
Test epoch: 56| Acc: 94.111761 (24454/25984)

Epoch: 57
Train epoch:57  Loss: 0.026551 | Loss_in: 0.002809, Loss_out: 0.002203 | Acc: 99.613472 (72933/73216)
Test epoch: 57| Acc: 94.157943 (24466/25984)

Epoch: 58
Train epoch:58  Loss: 0.026290 | Loss_in: 0.002716, Loss_out: 0.002238 | Acc: 99.613472 (72933/73216)
Test epoch: 58| Acc: 94.157943 (24466/25984)

Epoch: 59
Train epoch:59  Loss: 0.025598 | Loss_in: 0.002621, Loss_out: 0.002116 | Acc: 99.632594 (72947/73216)
Test epoch: 59| Acc: 94.069427 (24443/25984)
```

- Experiment with SVHN – Evaluating SDE-Net

Command: $ python test_detection.py --pre_trained_net save_sdenet_svhn_0/final_model --network sdenet --dataset svhn --out_dataset cifar10



```
(appCaptionEnv01) tm5u6@login1:~/SDE-Net-program/SVHN
$ python test_detection.py --pre_trained_net save_sdenet_svhn_0/final_model --network sdenet --dataset svhn --out_dataset cifar10
Namespace(batch_size=256, dataset='svhn', eva_iter=10, gpu=0, imageSize=32, network='sdenet', num_classes=10, out_dataset='cifar10', pre_trained_net
='save_sdenet_svhn_0/final_model', seed=0, test_batch_size=1000)
Random Seed:  0
Load model
load target data:  svhn
Building SVHN data loader with 1 workers
Using downloaded and verified file: ../data/svhn/train_32x32.mat
Using downloaded and verified file: ../data/svhn/test_32x32.mat
load non target data:  cifar10
Building CIFAR-10 data loader with 1 workers
Files already downloaded and verified
Files already downloaded and verified
generate log from in-distribution data

 Final Accuracy: 24102/25856 (93.22%)

generate log  from out-of-distribution data
calculate metrics for OOD
OOD  Performance of Baseline detector
TNR at TPR 95%:            90.967%
AUROC:                     98.012%
Detection acc:             93.756%
AUPR In:                   99.252%
AUPR Out:                  93.214%
calculate metrics for mis
mis  Performance of Baseline detector
TNR at TPR 95%:            65.985%
AUROC:                     92.198%
Detection acc:             86.665%
AUPR In:                   99.217%
AUPR Out:                  57.859%
```