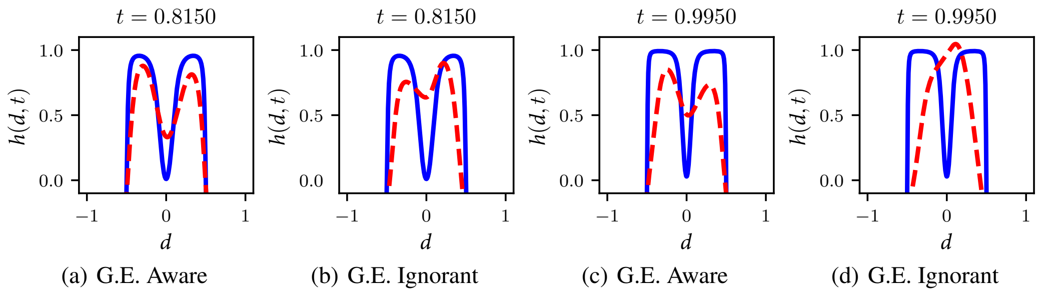


Figure 4: Extrapolation results by two neural networks (with identical architectures) that are aware or ignorant of the governing equation of the Allen–Cahn equation. The blue solid lines are reference solutions and the red dotted lines are extrapolation predictions. In all cases, better results are obtained when a neural network is aware of the governing equation, i.e., trained with L_G.

A IMPORTANCE OF LEARNING GOVERNING EQUATION

Here, we demonstrate the importance of learning governing equation in solving forward problems with an example, Allen–Cahn equation. The Allen–Cahn equation is a nonlinear reaction-diffusion problem, which describes the process of phase separation in alloys:

$$g(d,t) = h_t - 0.0001h_{dd} + 5h^3 - 5h = 0, d \in [-1,1],\ t \in [0,1], \tag{11}$$

with the initial condition $h(d,0) = d^2\cos(\pi d), \forall d \in [-1,1]$, and the periodic boundary conditions $h(-1,t) = h(1,t)$ and $h_d(-1,t) = h_d(1,t), \forall t \in [0,1]$. We note that $m = 1$ and $d_{\text{bc}} \in \{-1,1\}$ in this PDE. For computing its reference solutions, a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge–Kutta temporal integrator with time-step 10^{-6} is used.

To show the efficacy of the training method in Eqs. 2 through 5, we compare the method with the following naïve training method with the computed reference solutions:

$$\arg\min_{\boldsymbol{\theta}} \ L_I + L_B + L_R,$$

$$L_R \stackrel{\text{def}}{=} \frac{1}{N_R}\sum_{(d,t)} \big(f(d,t;\boldsymbol{\theta}) - h(d,t)\big)^2,$$

where L_R is to train $\boldsymbol{\theta}$ with the reference solutions of the Allen–Cahn equation with $h(d,t)$ $(t \leq 0.8)$. We note that the naïve model does not learn the governing equation but learn through the supervision with the reference solutions.

We also set $N_G = N_R$ and $t \leq 0.8$ to construct L_G for the fair comparison with the naïve model. We adopt the neural network architecture used in (Raissi et al., 2019) and train it with the two different training methods. As a result, one is aware of the governing equation because we use L_G and the other is ignorant of it because we use L_R instead of L_G. Figure 4 shows the extrapolation results for $t = \{0.8150, 0.9950\}$ obtained by using the two neural networks and we clearly see the governing-equation-aware neural network outperforms the other. In particular, the two figures at $t = 0.9950$ shows the efficacy of learning the governing equation: the prediction of the naïve model in Figure 4 (d) is not conforming to the underlying physical laws, considering that the Allen–Cahn equation is about the separation process of alloy. On the other hand, the model in Figure 4 (c) is aware of the existence of the valley around $x = 0$. This simple example demonstrates that the governing-equation-aware regression model generalizes much better for samples with unseen characteristics, e.g., extrapolation in the example. Thus, it is of our particular interest to make neural networks aware of governing equations in this work.

B PROOFS

Theorem B.1. *Given a machine learning task, let $\boldsymbol{\theta}^*$ and $\alpha^*_{i,j}$, for all i, j, be a cooperative equilibrium solution and governing equation (in terms of $L_T + \hat{L}_I + \hat{L}_G + R_G$) — in other words, we*

Table 6: The architecture of the network f

Layer	Design	Input Dim.	Output Dim.
1	Conv2d(filter size 3x3, stride 1, padding 1)	$6^2 \times 67$	$6^2 \times 67$
2	GroupNormalization(67 groups)	$6^2 \times 67$	$6^2 \times 67$
3	Conv2d(filter size 3x3, stride 1, padding 1)	$6^2 \times 67$	$6^2 \times 64$
4	GroupNormalizaiton(32 groups)	$6^2 \times 64$	$6^2 \times 64$
5	ReLU		

cannot minimize $L_T + \hat{L}_I + \hat{L}_G + R_G$ only by updating either of $\boldsymbol{\theta}^*$ or $\alpha^*_{i,j}$. By alternately solving the forward and the inverse problem, we can obtain $\boldsymbol{\theta}^*$ and $\alpha^*_{i,j}$, for all i, j.

Proof. We prove the theorem in the following sequence: i) we first prove that the forward problem is well-posed so that its solution uniquely exists, ii) the inverse problem can also be uniquely solved, and iii) we can obtain an equilibrium owing to the aforementioned uniquely-solvable characteristics.

Firstly, the forward problem is well-posed under the mild analytic condition of the following Eq. 12 — note that the following terms appear in the left-hand side of Eq. 1.

$$\alpha_{0,0} + \alpha_{1,0}f + \alpha_{2,0}f^2 + \alpha_{3,0}f^3 + \alpha_{0,1}f_d + \alpha_{1,1}ff_d + \alpha_{2,1}f^2f_d + \alpha_{3,1}f^3f_d + \alpha_{0,2}f_{dd} \\ + \alpha_{1,2}ff_{dd} + \alpha_{2,2}f^2f_{dd} + \alpha_{3,2}f^3f_{dd} + \alpha_{0,3}f_{ddd} + \alpha_{1,3}ff_{ddd} + \alpha_{2,3}f^2f_{ddd} + \alpha_{3,3}f^3f_{ddd} \tag{12}$$

For Eq. 12 to be analytic, f should be analytic w.r.t. d. All of f_d, f_{dd}, and f_{ddd} become analytic if f is analytic, and a composition of analytical functions is still analytic. Many neural network operators are analytic, e.g., softplus, fully-connected, exponential, and log, whereas some others are not, e.g., ReLU and absolute. Therefore, the analytical requirement can be fulfilled. If well-posed, the solution of the forward problem becomes a special case of the Cauchy problem and its solution uniquely exists.

Secondly, we prefer the most sparse governing equation that minimized the loss. Therefore, its solution can be uniquely defined and our training pursues it.

Lastly, let $\boldsymbol{\theta}^{(k)}$ and $\alpha^{(k)}_{i,j}$, for all i, j, be the solution and the governing equation obtained at k-th iteration of the algorithm. We quit the while loop when the sum of all the loss values converge and do not decrease in Alg. 1, which corresponds to the definition of the Nash equilibrium. Therefore, our algorithm always returns an equilibrium state. □

C Image Classification with MNIST and SVHN

We describe detailed experimental environments. Table 6 shows the detailed network architecture of f that we used for our experiments. The list of hyperparameters that we had considered for our experiments is as follows:

1. Train for 160 epochs with a batch size 128,
2. Use a MSE loss function for $\hat{L}_G, \hat{L}_I$ and a cross entropy loss function for L_T,
3. Use the standard PyTorch Adam optimizer for updating the governing equation g, $(d, t) \in H$, and the network f. On SVHN dataset, for the governing equation and (d, t) pairs, we use a weight decay of 1e-3. For MNIST, we update the governing equation and (d, t) pairs every epoch, and for SVHN, we update the governing equation and (d, t) pairs every 5 epochs.
4. $\boldsymbol{h}^{\text{task}}$ is a feature map in this case and its output size is in Table 6. We note that PR-Net has the same output size as that of ODE-Net. Refer to Section H about how we construct $\boldsymbol{h}^{\text{task}}$ with the set H of (d, t) pairs.

Table 7: The architecture of the network f

Layer	Design	Input Dim.	Output Dim.
1	Conv2d(1x1, stride 1)	$16^2 \times 67$	$16^2 \times 384$
2	BatchNorm2d	$16^2 \times 384$	$16^2 \times 384$
3	HSwish		
4	GroupConv2d(5x5, stride 1, groups 384)	$16^2 \times 384$	$16^2 \times 384$
5	BatchNorm2d	$16^2 \times 384$	$16^2 \times 384$
6	SE Block	$16^2 \times 384$	$16^2 \times 384$
7	HSwish		
8	Conv2d(1x1, stride 1)	$16^2 \times 384$	$16^2 \times 64$
9	BatchNorm2d	$16^2 \times 64$	$16^2 \times 64$

5. Utilize different learning rates for each dataset. For MNIST, we use a learning rate of 1e-3 to update the governing equation and (d, t) pairs and for SVHN, we use 3e-4 to update the governing equation and (d, t) pairs. For every datasets, we adjust the learning rate with a decay ratio of $\{0.1, 0.01, 0.001\}$ every 60, 80 and 140 epoch.

D Image Classification with Tiny Image Net

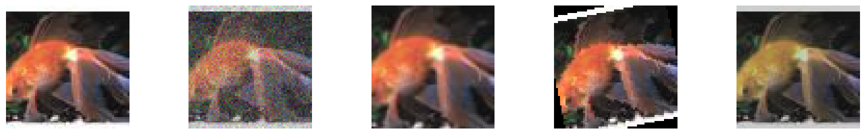
We describe detailed experimental environments. Table 7 shows the detailed network architecture of f that we used for our experiments. The list of hyperparameters that we had considered for our experiments is as follows:

1. Hswish and SE Block in Table 7 refers to hard-swish activation function and squeeze-and-excitation module used in (Howard et al., 2019), respectively.
2. Train for 150 epochs with a batch size of 64 and use early stopping.
3. Use a MSE loss function for $\hat{L}_G$, $\hat{L}_I$ and a cross entropy loss function with label smoothing 0.1 for L_T
4. Use three separate optimizers for updating the governing equation g, $(d, t) \in H$, and the network f. For the governing equation and (d, t) pairs, we use the standard Pytorch Adam optimizer with the L^1 regularization with a coefficient of $w = 2e-5$ and a weight decay of 2e-4, respectively. We update the governing equation and (d, t) pairs every 5 epochs. For the network f, we use the SGD optimizer with 0.9 momentum and apply a weight decay of 2e-4 to the learned weights in its convolutional and fully connected layers only.
5. $\boldsymbol{h}^{\text{task}}$ is a feature map in this case and its output size is in Table 7. We note that PR-Net has the same output size as that of ODE-Net. Refer to Section H about how we construct $\boldsymbol{h}^{\text{task}}$ with the set H of (d, t) pairs.
6. Utilize different learning rates for each optimizer. For learning the governing equation and (d, t) pairs, we use a learning rate of 1e-4. For training the network f, we gradually warm-up the learning rate for 5 epochs and use the cosine-annealing with the minimum learning rate set to 2e-4.
7. Use a dropout rate of 0.3 and batch-normalization layers with a momentum of 0.1.

E Adversarial Attack with Tiny ImageNet

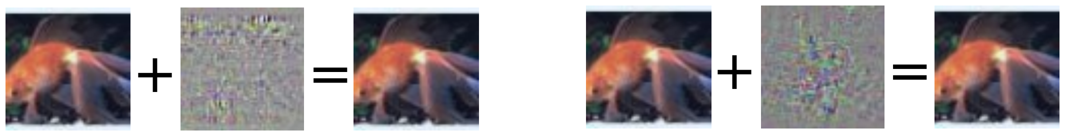
We describe detailed experimental environments for the reported adversarial attack experiments. We do not change the network architecture for these experiments. The list of hyperparameters of FGSM and PGD that we considered for our experiments is as follows:

1. For FGSM attack, we used a maximum perturbation of $\epsilon = \{0.5/255, 1/255, 3/255\}$.
2. For PGD attack, we employed a maximum perturbation of $\epsilon = \{0.5/255, 1/255, 3/255\}$ with 3 steps with a step-size of $\alpha = 1/255$.



(a) Original (b) Noise (c) Crop (d) Rotation (e) Jittering

Figure 5: Out-of-distribution examples



(a) FGSM (b) PGD

Figure 6: Adversarial attack examples. Goldfish with a confidence of 0.8931 is perturbed to torch with a confidence of 0.3546 in (a) and to candle with a confidence of 0.4810 in (b).

F Transfer Learning from Tiny ImageNet to Other Image Datasets

We describe detailed experimental environments for the reported transfer learning experiments. We do not change the network architecture for these experiments. We adopt the weights of the Tiny ImageNet pretrained model, and replace the last fully connected layer with a randomly initialized one that fits a target dataset. We then fine-tune all the layers of the pretrained model. All the target datasets are uniformly resized to 64 x 64. For data augmentation, we only used random horizontal flip for better reproducibility of our experiment. Note that same hyperparameters and training settings are employed for PR-Net, ODE-Net, and MobileNet V3. The list of hyperparameters that we considered for each target dataset is as follows:

1. To transfer from Tiny ImageNet to CIFAR100, CIFAR10, Food-101, we train for 80 epochs with a batch size of 64. We gradually warm-up the learning rate to 0.15 for 5 epochs and use the cosine-annealing with the minimum learning rate set to 2e-4. We utilize a dropout rate of 0.3 in fully connected layers and employ SGD optimizer with a momentum of 0.9 and a weight decay of 1e-4 applied to the learned weights in the convolutional and fully connected layers only.
2. To transfer from Tiny ImageNet to FGVC Aircraft and Cars, we train for 80 epochs with a batch size of 64. We gradually warm-up the learning rate to 0.15 for 5 epochs and use the cosine-annealing with the minimum learning rate set to 2e-4. We utilize a dropout rate of 0.3 in fully connected layers and employ SGD optimizer with a momentum of 0.9 and a weight decay of 5e-4 applied to the learned weights in the convolutional and fully connected layers only.
3. To transfer from Tiny ImageNet to DTD, we train for 150 epochs with a batch size of 64 and an initial learning rate of 0.001 that drops by a factor of 10 every 50 epoch. We employed SGD optimizer with a momentum of 0.9 and applied a weight decay of 5e-4 to the learned weights in the convolutional and fully connected layers only.

G Out-of-distribution and Adversarial Image Samples

We introduce a selected set of images that we produced for our robustness experiments. Figure 5 shows a set of image samples for the out-of-distribution image classification and Figure 6 shows a set of images perturbed by FGSM and PGD with $\epsilon = 3/255$. The original image is predicted as goldfish with a confidence of 0.8931 by PR-Net. The perturbed image by FGSM is predicted as

Table 8: Training overhead in terms of the GPU memory usage (megabytes) and the training time (seconds per iteration) in MNIST and SVHN

Name	# Params	MNIST		SVHN	
		Memory Usage	Training Time	Memory Usage	Training Time
ResNet	0.60M	2,359	**0.155**	2,363	**0.206**
RK-Net	0.22M	**819**	0.229	**823**	0.223
ODE-Net	0.22M	**819**	0.235	**823**	0.307
PR-Net	0.21M	836	0.289	841	0.227

Table 9: Training overhead in terms of the GPU memory usage (megabytes) and the training time (seconds per iteration) in Tiny ImageNet

Name	# Params	Width Multiplier	Tiny ImageNet	
			Memory Usage	Training Time
M.Net V3	1.21M	1	**4,989**	**0.038**
ODE-Net	1,36M	1	5,797	0.069
PR-Net	1.36M	1	7,583	0.077
M.Net V3	4.30M	2	**9,139**	**0.057**
ODE-Net	4.90M	2	9,977	0.211
PR-Net	4.56M	2	10,685	0.190

torch with a confidence of 0.3546 and the perturbed image by PGD is predicted as candle with a confidence of 0.4810.

H Discretizing Feature Map Dimensions for Efficient Processing

One more advantage of using PDEs is that we can discretize some dimensions[2]. Given a feature map size of $d_1 \times d_2 \times d_3$, one can design a neural network that outputs each scalar element for $d \in \mathbb{R}^3$ and $t \in [0, T]$. However, this approach incurs a large number of queries, i.e., $d_1 \times d_2 \times d_3$ queries, to reconstruct the feature map. To increase the efficiency in our experiments, we discretize the last dimension and let the network f outputs a matrix of $d_1 \times d_2$ for each discretized dimension of d_3, in which case $d \in \mathbb{R}^2$. Therefore, we have d_3 matrices (i.e., channels), each of which has a size of $d_1 \times d_2$. To further increase the efficiency, we let all the elements in the same position of the matrices share the same (d, t) pair where $d \in \mathbb{R}^2$ (See Figure 7 for the case of MNIST and SVHN as an example). In our case, we append three more channels to the input feature map $\boldsymbol{h}(0)$, each channel of which contains the index values of d_1, d_2, and t, respectively. When $t = 0$ and $d_1, d_2 = \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$, therefore, our network $f(\boldsymbol{h}(0), d, t; \boldsymbol{\theta})$ should output its initial condition $\boldsymbol{h}(0)$ to minimize $\hat{L}_I$ — note that we normalize d_1 and d_2. To minimize L_T, we construct the output feature map $\boldsymbol{h}^{task}$ with the various (d, t) pairs in H.

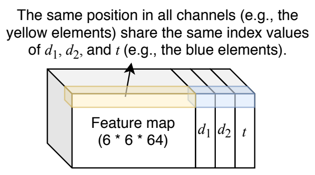
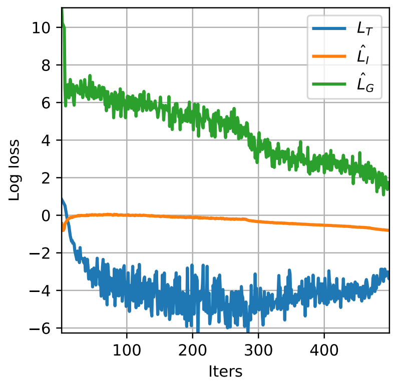

The same position in all channels (e.g., the yellow elements) share the same index values of d_1, d_2, and t (e.g., the blue elements).
Feature map (6 * 6 * 64)
d_1
d_2
t

Figure 7: An illustration for MNIST/SVHN on how to increase the processing efficiency by discretizing the last dimension and share (d, t) pairs.

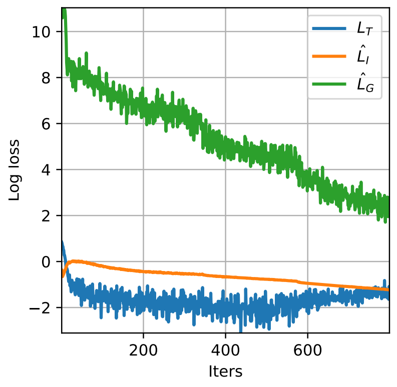
I Training Overhead

Our proposed PR-Net has several parts to be considered during its training process, e.g., governing equation. Due to these additional parts, our proposed method requires more resources in comparison with other baselines. However, training happens only once and after deployment, PR-Net shows more efficient behaviors, e.g. shorter forward-pass inference time. In this section, we compare the time and space overhead for MNIST, SVHN, and Tiny ImageNet.

[2]In fact, a PDE reduces to a system of ODEs after discretizing all spatial dimensions and maintaining only one time variable, i.e, there is one ODE for each discretized dimension and a system of such ODEs can approximate the original PDE. In this perspective, neural ODEs can be seen as that i) the hidden vector dimensions are discretized and ii) the time variable is maintained.



(a) Training curve in MNIST



(b) Training curve in SVHN

Figure 8: The curves of log loss values decrease as training goes on in MNIST and SVHN.

Table 10: Image classification datasets used in our experiments

Dataset	**# Classes**	**Size (Train / Test)**	**Evaluation Metrics**
MNIST	10	60,000 / 10,000	Top-1, Top-5, Mean & Std. Per-Class
SVHN	10	73,257 / 26,032	Top-1, Top-5, Mean & Std. Per-Class
Tiny ImageNet	200	100,000 / 10,000	Top-1, Top-5, Mean & Std. Per-Class
CIFAR 100	100	50,000 / 10,000	Top-1, Top-5, Mean & Std. Per-Class
CIFAR 10	10	50,000 / 10,000	Top-1, Top-5, Mean & Std. Per-Class
FGVC Aircraft	70	6,667 / 3,333	Top-1, Top-5, Mean & Std. Per-Class
Food-101	101	75,750 / 25,250	Top-1, Top-5, Mean & Std. Per-Class
Describable Textures (DTD)	47	3,760 / 1,880	Top-1, Top-5, Mean & Std. Per-Class
Stanford Cars	196	8,144 / 8,041	Top-1, Top-5, Mean & Std. Per-Class

Table 8 summarizes the training overhead in MNIST and SVHN. ResNet requires the largest amount of GPU memory but takes the smallest time per iteration. ODE-Net's training time per iteration is not as small as that of ResNet because it needs to solve integral problems. PR-Net has more factors to consider in a training iteration and requires more memory than ODE-Net in almost all cases. However, ODE-Net requires the longest time per iteration in SVHN because its adaptive step-size solver needs many steps to solve the reverse-mode integral problems to calculate gradients with the adjoint sensitivity method (Chen et al., 2018). It is worth noting that RK-Net, which has the same architecture as ODE-Net but use the standard backpropagation, takes much lesser time than ODE-Net.

For Tiny ImageNet, we summarize in Table 9. As expected, PR-Net require the largest amount of memory for its more complicated training loss definitions than those of baselines. However, ODE-Net requires the longest time per iteration when the width multiplier is set to 2. This phenomenon also happened for MNIST and SVHN. The reverse-mode integral of the adjoint sensitivity method has a space complexity of $\mathcal{O}(1)$ but in any case it needs to solve an integral problem, which incurs additional time complexity.

Figure 8 illustrates the curves of $L_T, \hat{L}_I, \hat{L}_G$ in MNIST and SVHN. Both L_T and $\hat{L}_I$ are easier to train than $\hat{L}_G$. The governing equation loss $\hat{L}_G$ typically starts with a very large value and decreases slowly as training goes on. In comparison with $\hat{L}_G$, the task loss L_T decreases much faster, which shows the difficulty of learning a physical dynamics (i.e., governing equation) governing the classification procedures.

Table 11: Image classification in Tiny ImageNet. We show the mean and the standard deviation of per-class accuracy.

Name	M.Net V3	ODE-Net	PR-Net	M.Net V3	ODE-Net	PR-Net
Width Multiplier	1	1	1	2	2	2
Mobile Blocks	4	3	3	4	3	3
ODE Blocks	N/A	1	N/A	N/A	1	N/A
PDE Blocks	N/A	N/A	1	N/A	N/A	1
Mean Accuracy	0.5809	0.5547	**0.5972**	0.6076	0.5672	**0.6157**
Std. Dev. Accuracy	0.1584	0.1628	**0.1473**	0.1570	0.1618	**0.1496**
# Params	1.21M	1.36M	1.36M	4.30M	4.90M	4.56M
Inference Time	**4.14**	5.26	5.23	**5.21**	8.3	6.25
Out-of-distribution Robustness (Mean Accuracy)						
Gaussian Noise	0.4495	0.4165	**0.4685**	0.4757	0.4474	**0.4878**
Random Crop & Resize	0.4636	0.4305	**0.4841**	0.4814	0.4419	**0.4965**
Random Rotation	0.3961	0.3667	**0.4267**	0.4256	0.3901	**0.4381**
Color Jittering	0.4206	0.3812	**0.4429**	0.4555	0.4108	**0.4693**
Out-of-distribution Robustness (Std. Dev. Accuracy)						
Gaussian Noise	0.1747	0.1697	**0.1610**	0.1710	0.1754	**0.1674**
Random Crop & Resize	0.1768	0.1824	**0.1731**	0.1786	0.1862	**0.1801**
Random Rotation	0.1623	0.1690	**0.1606**	0.1719	0.1759	**0.1664**
Color Jittering	0.1505	0.1495	**0.1462**	0.1534	0.1491	**0.1535**

Table 12: Adversarial attacks in Tiny ImageNet. We show the mean and the standard deviation of per-class accuracy.

Attack Method	M.Net V3	ODE-Net	PR-Net	M.Net V3	ODE-Net	PR-Net
	Mean Accuracy			Std. Dev. Accuracy		
FGSM($\epsilon = 0.5/255$)	0.3860	0.3656	**0.4041**	0.1778	0.1716	**0.1685**
FGSM($\epsilon = 1/255$)	0.2304	0.2287	**0.2499**	0.1631	0.1639	**0.1561**
FGSM($\epsilon = 3/255$)	0.0452	**0.0464**	0.0369	0.0775	0.0791	**0.0653**
PGD ($\epsilon = 0.5/255$)	0.3733	0.3525	**0.3910**	0.1774	0.1726	**0.1661**
PGD ($\epsilon = 1/255$)	0.1902	0.1908	**0.2133**	0.1488	0.1553	**0.1467**
PGD ($\epsilon = 3/255$)	0.0218	**0.0235**	0.017	0.0506	0.0558	**0.0480**

J ADDITIONAL EXPERIMENTAL RESULTS – PER-CLASS ACCURACY

All datasets we used are summarized in Table 10. Since some datasets have many classes (e.g., 200 classes in Tiny ImageNet), we introduce the mean and standard deviation of per-class accuracy for each dataset. In this section, we use only the top-1 accuracy to calculate the mean and standard deviation of per-class accuracy — we did not use the per-class accuracy in the main paper. We note that lower (resp. larger) values are preferred for the standard deviation (resp. for the mean).

In Table 11, we summarize the mean and the standard deviation of per-class accuracy for our Tiny ImageNet classification and out-of-distribution robustness experiments. In the Tiny ImageNet classification experiment, PR-Net shows the smallest standard deviation in all cases, which means that it achieved more uniform per-class accuracy than other baselines. In some cases, ODE-Net fails to show more uniform per-class accuracy than MobileNet V3. We could observe similar patterns for the standard deviation in the out-of-distribution robustness experiment.

For our adversarial attack experiment, we summarize the mean and the standard deviation of per-class accuracy in Table 12. PR-Net shows smaller standard deviations than other baselines. Sometimes, ODE-Net also shows good performance.

The datasets we used for our transfer learning experiments also have many classes and some of them are not balanced, e.g., Aircraft, DTD, and Cars. In those unbalanced datasets, the mean of per-class accuracy is different from the mean accuracy in Table 5. We summarize their means and standard deviations of per-class accuracy in Table 13. As reported, PR-Net shows smaller standard deviation values than baselines in many cases. For MNIST and SVHN, all methods have good per-class accuracy distribution patterns.

Table 13: Transfer learning in Tiny ImageNet. We show the mean and the standard deviation of per-class accuracy.

Dataset	M.Net V3	ODE-Net	PR-Net	M.Net V3	ODE-Net	PR-Net
	Mean Accuracy			Std. Dev. Accuracy		
CIFAR100	0.7676	0.7460	**0.7750**	**0.1139**	0.1142	0.1146
CIFAR10	0.9403	0.9280	**0.9417**	0.0301	0.04	**0.029**
Aircraft	0.5922	0.5704	**0.6364**	**0.1889**	0.1975	0.1909
Food-101	0.7317	0.7128	**0.7366**	0.1156	0.1199	**0.1135**
DTD	0.4819	0.5016	**0.5154**	0.1546	0.1683	**0.1515**
Cars	**0.6322**	0.5576	0.6294	0.1358	**0.127**	0.1360

K FEATURE MAP ANALYSES

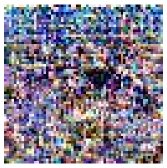
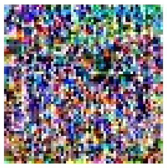
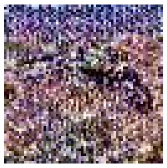
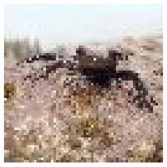
We also analyzed the feature maps created by MobileNet V3, ODE-Net, and PR-Net in Figure 9. For this, we use the method of image representation inversion which i) is to find an image whose representation best matches a given representation vector, and ii) also had been used in (Engstrom et al., 2019a) to check the quality of feature maps. According to (Engstrom et al., 2019a), robust representations are approximately invertible. For MoblieNet V3, ODE-Net, and PR-Net, we reconstruct the target image using the representation vector produced at the third Mobile block of each model and strictly follow the inversion method used in (Mahendran & Vedaldi, 2015) after downloading the program codes in the respected github repository[3]. As shown in Figure 9, our PR-Net shows the best inversion quality in many cases.

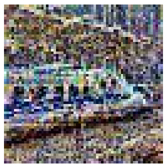
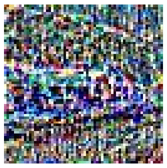
Figures 10 and 11 visualizes the feature maps of ResNet, ODE-Net, and PR-Net for MNIST and SVHN using t-SNE algorithm. In terms of human visual perception, they all look similar. We further employed the silhouette score to evaluate the quality of clusters on t-SNE embeddings, where the number of clusters is the number of classes. PR-Net shows the best clustering outcomes by classes in Table 14, e.g., a silhouette score of 0.4959 for ResNet in MNIST vs. 0.4991 for ODE-Net vs. 0.5079 for PR-Net.

Table 14: The silhouette score of clustering feature maps in MNIST and SVHN

Name	MNIST	SVHN
ResNet	0.49594527	0.42278063
RK-Net	0.5053296	0.42842203
ODE-Net	0.4991746	0.42694366
PR-Net	**0.5079406**	**0.43123975**

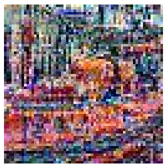
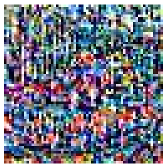
[3]`https://github.com/utkuozbulak/pytorch-cnn-visualizations`


(a) M.Net V3

(b) ODE-Net

(c) PR-Net

(d) Original


(e) M.Net V3

(f) ODE-Net

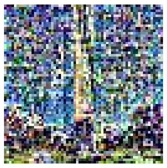
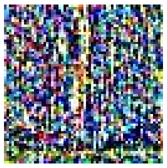
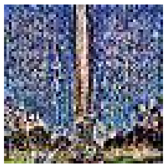
(g) PR-Net

(h) Original

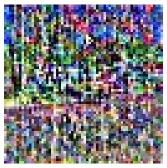
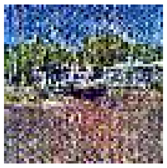

(i) M.Net V3

(j) ODE-Net

(k) PR-Net

(l) Original


(m) M.Net V3

(n) ODE-Net

(o) PR-Net

(p) Original

(q) M.Net V3

(r) ODE-Net

(s) PR-Net

(t) Original

Figure 9: The visualization of image representation inversion in Tiny ImageNet

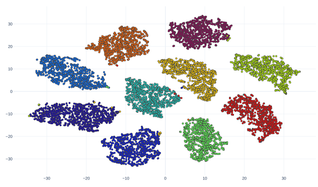
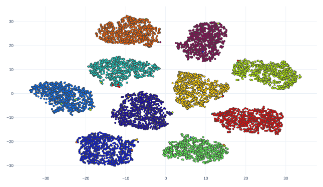
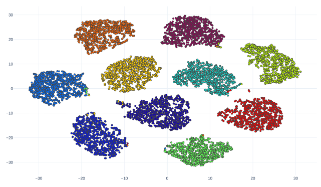
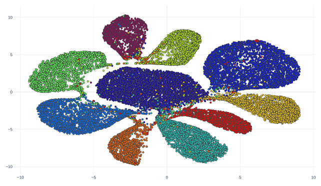
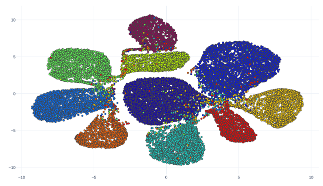

(a) ResNet

(b) ODE-Net

(c) PR-Net

Figure 10: The visualization of feature maps in MNIST. We use t-SNE to project the feature maps onto a 2-dimensional space.


(a) ResNet


(b) ODE-Net

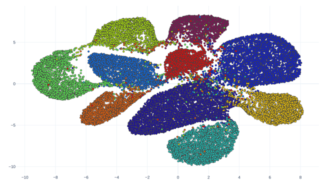

(c) PR-Net

Figure 11: The visualization of feature maps in SVHN. We use t-SNE to project the feature maps onto a 2-dimensional space.