REGULARIZING IMAGE CLASSIFICATION NEURAL NETWORKS WITH PARTIAL DIFFERENTIAL EQUATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Differential equations can be used to design neural networks. For instance, neural ordinary differential equations (neural ODEs) can be considered as a continuous generalization of residual networks. In this work, we present a novel partial differential equation (PDE)-based approach for image classification, where we learn both a PDE's governing equation for image classification and its solution approximated by our neural network. In other words, the knowledge contained in the learned governing equation can be injected into the neural network which approximates the PDE solution function. Owing to the recent advancement of learning PDEs, the presented novel concept, called PR-Net, can be implemented. Our method shows comparable (or better) accuracy and robustness for various datasets and tasks in comparison with neural ODEs and Isometric MobileNet V3. Thanks to the efficient nature of PR-Net, it is suitable to be deployed in resource-scarce environments, e.g., deployed instead of MobileNet.

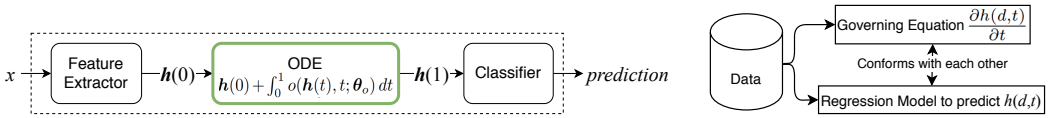
1 INTRODUCTION

It has been discovered that combining neural networks and differential equations is possible by several independent research groups (Weinan, 2017; Ruthotto & Haber, 2019; Lu et al., 2018; Ciccone et al., 2018; Chen et al., 2018; Gholami et al., 2019). For instance, the seminal neural ordinary differential equation (neural ODE) research work, which considers the general architecture in Figure 1 (a), is to learn a neural network approximating $\frac{\partial \boldsymbol{h}(t)}{\partial t}$, where $\boldsymbol{h}(t)$ is a hidden vector at layer (or time) t (Chen et al., 2018). As such, a neural network is described by a system of ODEs, each ODE of which describes a dynamics of a hidden element. While neural ODEs have many preferred characteristics, they also have limitations as follows:

1. Neural ODEs can interpret t as a continuous variable and we can have hidden vectors at any layer (or time) l by $\boldsymbol{h}(l) = \boldsymbol{h}(0) + \int_0^l o(\boldsymbol{h}(t), t; \boldsymbol{\theta}_o)\, dt$, where $o(\boldsymbol{h}(t), t; \boldsymbol{\theta}_o) = \frac{\partial \boldsymbol{h}(t)}{\partial t}$ is a neural network parameterized by $\boldsymbol{\theta}_o$.
2. Neural ODEs sometimes have smaller numbers of parameters than those of other conventional neural network designs, e.g., (Pinckaers & Litjens, 2019).
3. Neural ODEs' forward-pass inference can take a long time in solving integral problems, e.g., a forward-pass time of 37.6 seconds of ODE-Net vs. 9.8 seconds of our method in Table 2. Several countermeasures have been proposed to enhance the inference time, but it is unavoidable to solve integral problems (Zhuang et al., 2020; Finlay et al., 2020; Daulbaev et al., 2020).

To tackle the limitation, we propose the concept of partial differential equation (PDE)-regularized neural network (PR-Net) to directly learn a hidden element, denoted $h(d, t)$ at layer (or time) $t \in [0, T]$ and dimension $d \in \mathbb{R}^m$. Under general contexts, a PDE consists of i) an initial condition at $t = 0$, ii) a boundary condition at a boundary location of the spatial domain $\mathbb{R}^m$, and iii) a governing equation describing $\frac{\partial h(d,t)}{\partial t}$. As such, learning a PDE from data can be reduced to a regression-like problem to predict $h(d, t)$ that meets its initial/boundary conditions and governing equation.

In training our proposed PR-Net, $\boldsymbol{h}(0)$ is provided by an earlier feature extraction layer, which is the same as neural ODEs. However, an appropriate governing equation is unknown for downstream



(a) The general architecture of neural ODEs (b) The proposed PR-Net concept

Figure 1: Our PR-Net avoids solving integral problems by learning a regression model that conforms with a learned governing equation.

machine learning tasks. Therefore, we propose to train a regression model for predicting $h(d,t)$ and its governing equation simultaneously (see Figure 1 (b)). In other words, neural ODEs directly learn a governing equation only (i.e., $\frac{\partial \boldsymbol{h}(t)}{\partial t}$), whereas PR-Net learns a governing equation in conjunction with a regression model that conforms with the learned governing equation. The key advantage in our approach is that we can eliminate the necessity of solving integral problems; in neural ODEs, where we learn a governing equation only, solving integral problems is mandatory.

Such forward and inverse problems (i.e., solving PDEs for $h(d,t)$ and identifying governing equations, respectively) arise in many important computational science problems and there have been many efforts applying machine learning/deep learning techniques to those problems (e.g., in earth science (Reichstein et al., 2019; Bergen et al., 2019) and climate science (Rolnick et al., 2019)). Recently, physics-informed or physics-aware approaches (Battaglia et al., 2016; Chang et al., 2017; de Bezenac et al., 2018; Raissi et al., 2019; Sanchez-Gonzalez et al., 2018; Long et al., 2018) have demonstrated that designing neural networks to incorporate prior scientific knowledge (e.g., by enforcing physical laws described in governing equations (Raissi et al., 2019)) greatly helps avoiding over-fitting and improving generalizability of the neural networks. There also exist several approaches to incorporate various ideas of classical mechanics in designing neural-ODE-type networks (Greydanus et al., 2019; Chen et al., 2020; Cranmer et al., 2020; Zhong et al., 2020; Lee & Parish, 2020). However, all these works are concerned with solving either forward or inverse problems whereas we solve the two different problem types at the same time for downstream tasks. The most similar existing work to our work is in (Long et al., 2018). However, this work studied scientific PDEs and did not consider t as a continuous variable but use a set of discretized points of t.

Compared to previous approaches, our proposed method has a distinct feature that forward and inverse problems are solved simultaneously with a continuous variable t. Due to this unique feature, the method can be applied to general machine learning downstream tasks, where we do not have a priori knowledge on governing equations, such as image classification.

We conduct experiments mainly for image classification for its appropriateness (see Appendix A). We introduce our in-depth studies for various aspects of image classification, including adversarial attacks, out-of-distribution classification, feature map analyses, and so forth. Throughout the experiments, our PR-Net shows the best robustness, which is well aligned with our preliminary extrapolation study of a PDE in Appendix C. One can understand the adversarial attack and out-of-distribution classification experiments are challenging extrapolation tasks in image classification. In this regard, learning governing equations shows great robustness for extrapolation in our experiments. Our proposed PR-Net has the following characteristics:

1. PR-Net trains a regression model that outputs a scalar element $h(d,t)$ (without solving any integral problems), and we can consider both d and t as continuous variables. Therefore, it is possible to construct flexible hidden vectors at arbitrary dimensions and layers.
2. PR-Net does not solve integral problems whereas neural ODEs need to solve integral problems.
3. By learning a governing equation, we can regularize the overall behavior of PR-Net. This greatly enhances the robustness of model as shown in our experiments.

2 Partial Differential Equations

The key difference between ODEs and PDEs is that PDEs can have derivatives of multiple variables whereas ODEs should have only one such variable's derivative. Therefore, our PDE-based method

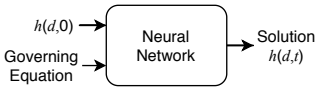


Figure 2: A network predicts solutions at d, t given initial conditions, denoted $h(d, 0)$ for various d, and a governing equation.

Table 1: Two types of PDE problems

Type	Data	What to infer
Forward Problem	- Initial condition - Governing equation	Solution $h(d, t)$
Inverse Problem	- Solution $h(d, t)$ - Initial condition	Governing equation

interprets both the layer of neural network and the dimension of hidden vector as continuous variables, which cannot be done in neural ODEs. In our context, $h(d, t)$ means a hidden scalar element at layer $t \in \mathbb{R}$ and dimension $d \in \mathbb{R}^m$, e.g., $m = 1$ if $\boldsymbol{h}(t)$ is a vector, $m = 3$ if $\boldsymbol{h}(t)$ is a convolutional feature map, and so on.

In this section, we first introduce the forward and inverse problems of PDEs in general contexts (see Table 1). Then, we extend them to design our proposed method in deep-learning contexts.

2.1 Forward Problem of PDEs in General Contexts

The forward PDE problem in general contexts is to find a solution $h(d, t)$, where d is in a spatial domain $\mathbb{R}^m$ and t is in a time domain $[0, T]$, given i) an initial condition $h(d, 0)$, ii) a boundary condition $h(d_{\text{bc}}, t)$, where d_{bc} is a boundary location of the spatial domain $\mathbb{R}^m$, and iii) a governing equation g (Raissi et al., 2019) We note that the boundary condition can be missing in some cases (Kim, 2018). The governing equation is typically in the following form with particular choices of $\alpha_{i,j}$ (Raissi, 2018; Peng et al., 2020):

$$\begin{aligned} g(d, t; h) \stackrel{\text{def}}{=} & h_t - \big(\alpha_{0,0} + \alpha_{1,0} h + \alpha_{2,0} h^2 + \alpha_{3,0} h^3 + \alpha_{0,1} h_d + \alpha_{1,1} h h_d + \alpha_{2,1} h^2 h_d + \alpha_{3,1} h^3 h_d \\ & + \alpha_{0,2} h_{dd} + \alpha_{1,2} h h_{dd} + \alpha_{2,2} h^2 h_{dd} + \alpha_{3,2} h^3 h_{dd} \\ & + \alpha_{0,3} h_{ddd} + \alpha_{1,3} h h_{ddd} + \alpha_{2,3} h^2 h_{ddd} + \alpha_{3,3} h^3 h_{ddd}\big), \end{aligned} \tag{1}$$

where $h_t = \frac{\partial h(d,t)}{\partial t}$, $h_d = \frac{\partial h(d,t)}{\partial d}$, $h_{dd} = \frac{\partial^2 h(d,t)}{\partial d^2}$, and $h_{ddd} = \frac{\partial^3 h(d,t)}{\partial d^3}$. We also note that g is always zero in all PDEs, i.e., $g(d, t; h) = 0$.

In many cases, it is hard to solve the forward problem and hence general purpose PDE solvers do not exist. Nevertheless, one can use the following optimization to train a neural network $f(d, t; \boldsymbol{\theta})$ to approximate the solution function $h(d, t)$ as shown in Figure 2 (Raissi et al., 2019):

$$\arg\min_{\boldsymbol{\theta}} \; L_I + L_B + L_G, \tag{2}$$

$$L_I \stackrel{\text{def}}{=} \frac{1}{N_I} \sum_{d} \big(f(d, 0; \boldsymbol{\theta}) - h(d, 0)\big)^2, \tag{3}$$

$$L_B \stackrel{\text{def}}{=} \frac{1}{N_B} \sum_{(d_{\text{bc}}, t)} \big(f(d_{\text{bc}}, t; \boldsymbol{\theta}) - h(d_{\text{bc}}, t)\big)^2, \tag{4}$$

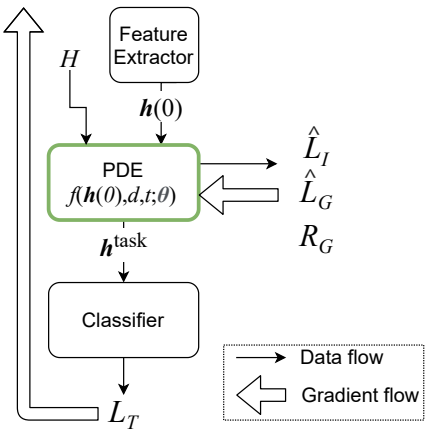
$$L_G \stackrel{\text{def}}{=} \frac{1}{N_G} \sum_{(d,t)} g(d, t; f, \boldsymbol{\theta})^2, \tag{5}$$

where N_I, N_B, N_G are the numbers of training samples, L_I is to train $\boldsymbol{\theta}$ for the initial condition, L_B is for the boundary condition, and L_G is for the governing equation. Because the governing equation is always zero, we simply minimize its squared term. Note that i) $f_t, f_d, f_{dd}, f_{ddd}$ can be easily constructed using the automatic differentiation implemented in TensorFlow or PyTorch, and ii) we only need $h(d, 0), h(d_{\text{bc}}, t)$, which are known a priori, to train $\boldsymbol{\theta}$.

2.2 Inverse Problem of PDEs in General Contexts

The inverse problem is to find a governing equation given i) an initial condition $h(d, 0)$ and ii) a solution function $h(d, t)$ (Raissi, 2018). It learns $\alpha_{i,j}$ in Eq. 1 with the following loss:

$$\arg\min_{\alpha_{i,j}} \frac{1}{N_G} \sum_{(d,t)} g(d, t; h)^2.$$



Algorithm 1 How to train PR-Net

Input: training data X, validating data V, max iteration number max_iter
Output: $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j

Initialize $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j;
$k \leftarrow 0$;
$L_{\text{sum}} \leftarrow \infty$;
while L_{sum} *is not converged and* $k < max\_iter$ **do**
  Train $\boldsymbol{\theta}$, Feature Extractor, and Classifier with L_T ;
  Train $\boldsymbol{\theta}$ with $L_T + \hat{L}_I + \hat{L}_G$;
  Train $(d,t) \in H$ with L_T ;
  Train $\alpha_{i,j}$ for all i,j with $\hat{L}_G + R_G$;
  Validate with V and update the best model;
  $L_{\text{sum}} \leftarrow L_T + \hat{L}_I + \hat{L}_G + R_G$;
  $k \leftarrow k + 1$;
return $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j;

Figure 3: The general architecture and the training algorithm of PR-Net

Given a solution function h and its partial derivative terms, we train $\alpha_{i,j}$ by minimizing the objective loss. Note that we know h in this case. Therefore, the objective loss is defined with h rather than with f, unlike Eq. 5. The optimal solution of $\alpha_{i,j}$ is not unique sometimes. However, we note that no trivial solutions, e.g., $\alpha_{i,j} = 0$ for all i,j, exist for the inverse problem.

3 PDE-Regularized Neural Networks

Before describing our method, we note the importance of learning governing equations illustrated in Appendix C. In all cases, learning governing equations leads to reliable extrapolation results when solving PDEs. Therefore, it is a key part in our work to discover and learn an appropriate governing equation.

Our goal in this work is to replace a system of ODEs (cf. Figure 1 (a)) with a PDE. Assuming that a target task-specific PDE is known *a priori*, given an initial condition $\boldsymbol{h}(0)$ extracted by the feature extractor from a sample x, a forward problem can be solved via the method described in Section 2.1. However, a target task-specific PDE is not known *a priori* in general, and thus, the governing equation should be learned from data via solving the inverse problem. Unfortunately, the solution function $h(d,t)$ is not known *a priori* either in our setting. Therefore, we make an assumption on the governing equation that it consists of the most common partial derivative terms (cf. Eq. 1) and then propose to solve the forward and the inverse problems alternately: to train $\boldsymbol{\theta}$, we fix its governing equation g (more precisely, $\alpha_{i,j}$ for all i,j), and to train $\alpha_{i,j}$ for all i,j, we fix $\boldsymbol{\theta}$.

How to Solve Forward Problem. We customize the method presented in Section 2.1 by i) adding a task-specific loss, e.g., cross-entropy loss for image classification, ii) parameterizing the neural network f by the initial condition $\boldsymbol{h}(0)$, and iii) dropping the boundary condition. Let $f(\boldsymbol{h}(0), d, t; \boldsymbol{\theta})$ be our neural network to approximate $h(d,t)$ given the varying initial condition $\boldsymbol{h}(0)$. The definition of the governing equation is also extended to $g(d,t; f, \boldsymbol{h}(0), \boldsymbol{\theta})$[1]. In order to exploit the key benefit of our framework that we can extract any hidden element by querying (d,t) to f, we define that H is a set of (d,t) pairs, where $d \in \mathbb{R}_{\geq 0}, t \in \mathbb{R}_{\geq 0}$, with which we construct the hidden vector $\boldsymbol{h}^{\text{task}}$ that will be

[1]The governing equation g, which consists of partial derivatives of f, is also a neural network. We call the proposed concept as *neural partial differential equations* and the neural network f trained with the concept as *PDE-regularized neural networks*.

used for a downstream task (cf. Figure 3). We use the following loss definition to train $\boldsymbol{\theta}$:

$$\arg\min_{\boldsymbol{\theta}} \quad L_T + \hat{L}_I + \hat{L}_G, \tag{6}$$

$$\hat{L}_I \stackrel{\text{def}}{=} \frac{1}{N_X} \sum_{x \in X} \Big(\frac{1}{\dim(\boldsymbol{h})} \sum_{d} \big(f(\boldsymbol{h}(0), d, 0; \boldsymbol{\theta}) - h(d, 0) \big)^2 \Big), \tag{7}$$

$$\hat{L}_G \stackrel{\text{def}}{=} \frac{1}{N_X} \sum_{x \in X} \Big(\frac{1}{N_H} \sum_{(d,t) \in H} g(d, t; f, \boldsymbol{h}(0), \boldsymbol{\theta})^2 \Big), \tag{8}$$

where L_T is a task-specific loss, X is a training set, N_X is the number of training sample, and N_H is the number of elements in $\boldsymbol{h}^{\text{task}}$, i.e., $\dim(\boldsymbol{h}^{\text{task}})$.

We query $f(\boldsymbol{h}(0), d, t; \boldsymbol{\theta})$ with the (d, t) pairs in H to construct $\boldsymbol{h}^{\text{task}}$. One more important point to note is that in order to better construct $\boldsymbol{h}^{\text{task}}$, we can train even the pairs in H as follows: $\arg\min_{(d,t) \in H} L_T$ (line 7 in Alg. 1). Thus, the elements of $\boldsymbol{h}^{\text{task}}$ can be collected from different dimensions and layers. A similar approach to optimize the end time of integral was attempted for neural ODEs in (Massaroli et al., 2020). To train H, we use only L_T because $\boldsymbol{h}^{last}$ that will be fed into the classifier should work well for a downstream task. The feature extractor and the classifier are also trained only with L_T.

How to Solve Inverse Problem. After fixing $\boldsymbol{\theta}$, we train $\alpha_{i,j}$ for all i, j by using the following L_1 regularized loss with a coefficient w:

$$\arg\min_{\alpha_{i,j}} \quad \hat{L}_G + R_G, \tag{9}$$

where $R_G \stackrel{\text{def}}{=} w \sum_{i,j} |\alpha_{i,j}|$. We minimize the sum of $|\alpha_{i,j}|$ to induce a sparse governing equation according to Occam's razor since in many PDEs, their governing equations are sparse. This optimization allows us to choose the sparsest solution among many possible governing equations. In many cases, therefore, our regularized inverse problem can be uniquely solved.

Training Algorithm. Our overall training algorithm is in Alg. 1. We alternately train $\boldsymbol{\theta}$, $(d, t) \in H$, and $\alpha_{i,j}$ for all i, j. The forward problem to train $\boldsymbol{\theta}$ becomes a well-posed problem (i.e., its solution always exists and is unique) if the neural network f is analytical or equivalently, uniformly Lipschitz continuous (Chen et al., 2018). Many neural network operators are analytical, for example, softplus, fully-connected, exponential, and so on. Under the mild condition of analytical neural networks, therefore, the well-posedness can be fulfilled. The inverse problem can also be uniquely solved in many cases due to the sparseness requirement. As a result, our proposed training algorithm can converge to a cooperative equilibrium. Note that $\boldsymbol{\theta}$, $(d, t) \in H$, and $\alpha_{i,j}$ for all i, j cooperate to minimize $L_T + \hat{L}_I + \hat{L}_G + R_G$. Therefore, the proposed training method can be seen as a cooperative game (Mas-Colell, 1989), as shown in the following theorem. (The proof can be found in Appendix D.)

Theorem 3.1. *Given a learning task, let $\boldsymbol{\theta}^*$ and $\alpha^*_{i,j}$, for all i, j, constitute a cooperative equilibrium solution and governing equation (in terms of $L_T + \hat{L}_I + \hat{L}_G + R_G$) — in other words, we cannot further decrease $L_T + \hat{L}_I + \hat{L}_G + R_G$ 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.*

After finishing the training process, $\alpha_{i,j}$, for all i, j, are not needed any more (because $\boldsymbol{\theta}$ already conforms with the learned governing equation at this point) and can be discarded during testing.

For complicated downstream tasks, training for L_T should be done earlier than others (line 5). Then, the PDE parameters are carefully updated (line 6) and other training procedures follow. The sequence in Alg. 1 produces the best outcomes in our experiments. However, this sequence can be varied for other datasets or downstream tasks.

Complexity Analyses. The adjoint sensitivity method of neural ODEs enables the space complexity of $\mathcal{O}(1)$ while calculating gradients. However, its forward-pass inference time is $\mathcal{O}(\frac{1}{s})$, where s is the (average) step-size of an underlying ODE solver. Because s can sometimes be very small, its inference via forward-pass can take a long time.

Our PR-Net uses the standard backpropagation method to train and its gradient computation complexity is the same as that in conventional neural networks. In addition, the forward-pass inference time is $\mathcal{O}(1)$, given a fixed network f, because we do not solve integral problems.

4 EXPERIMENTS

In this section, we introduce our experimental evaluations with various datasets and tasks. All experiments were conducted in the following software and hardware environments: UBUNTU 18.04 LTS, PYTHON 3.6.6, NUMPY 1.18.5, SCIPY 1.5, MATPLOTLIB 3.3.1, PYTORCH 1.2.0, CUDA 10.0, and NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX TITAN. In Appendix K, we summarize detailed dataset information and provide additional experiments.

PDE-based methods have shown appropriateness for many image processing tasks (as listed in Introduction), and we believe that the main application area of the proposed method in particular is image processing in resource-scarce environments thanks to its efficiency.

4.1 IMAGE CLASSIFICATION WITH MNIST AND SVHN

We reuse the convolutional neural network, called ODE-Net (Chen et al., 2018) to classify MNIST and SVHN and replace its ODE part with our proposed PDE, denoted PR-Net in Table 2. See Appendix E for the architecture and the hyperparameters of the network f in PR-Net for this experiment. We reuse their codes and strictly follow their experimental environments.

Table 2: Image classification in MNIST and SVHN. The inference time is the time in seconds to classify a batch of 1,000 images. In general, PR-Net shows the best efficiency.

Name	# Params	MNIST		SVHN	
		Test Accuracy	Inference Time	Test Accuracy	Inference Time
ResNet	0.60M	0.9966	7.6447	**0.9660**	**8.6721**
RK-Net	0.22M	0.9970	7.4774	0.9652	13.5139
ODE-Net	0.22M	0.9964	24.8355	0.9599	37.6776
PR-Net	0.21M	**0.9972**	**6.5023**	0.9615	9.8263

Its detailed results are summarized in Table 2. We compare with ResNet, RK-Net and ODE-Net. In ResNet, we have a downsampling layer followed by 6 standard residual blocks (He et al., 2016). For RK-Net and ODE-Net, we reuse the codes provided by (Chen et al., 2018). ODE-Net is trained with the adjoint sensitivity method. Our PR-Net, which does not require solving integral problems, shows the best performance in all aspects for MNIST. For SVHN, ResNet shows the best accuracy.

4.2 IMAGE CLASSIFICATION WITH TINY IMAGENET & CIFAR10/100

We use one more convolutional neural network to test with Tiny ImageNet. Tiny ImageNet is the modified subset of ImageNet with downscaled image resolution 64×64. It consists of 200 different classes with 100,000 training images and 10,000 validation images. Our baseline model is Isometric MobileNet V3 (Sandler et al., 2019). We also compare with ODE-Net. Since ODE-Net and PR-Net are both efficient, we suppose that resource-scarce environments, for which MobileNet was designed, are their best application areas. The isometric architecture of Isometric MobileNet V3 maintains constant resolution throughout all layers. Therefore, pooling layers are not needed and computation efficiency is high, according to their experiments. In addition, neural ODEs require an isometric architecture, i.e., the dimensionality of $\boldsymbol{h}(t)$, $t \geq 0$, cannot be varied. In our PR-Net, we do not have such restrictions. For fair comparison, however, we have decided to use Isometric MobileNet V3. We replace some of its MobileNet V3 blocks with ODEs or PDEs, denoted ODE-Net and PR-Net in Table 3, respectively. We train our models from scratch without using any pretrained network, with a synchronous training setup.

Table 3: Image classification in Tiny ImageNet. PR-Net shows better efficiency than ODE-Net.

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
Accuracy (top-1)	0.5809	0.5547	**0.5972**	0.6076	0.5672	**0.6157**
Accuracy (top-5)	0.8049	0.7946	**0.8166**	0.8115	0.7911	**0.8357**
# 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 (top-1 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 (top-5 accuracy)						
Gaussian Noise	0.68	0.6619	**0.7064**	0.7025	0.6757	**0.7205**
Random Crop & Resize	0.7106	0.6935	**0.7357**	0.7215	0.6936	**0.7442**
Random Rotation	0.6372	0.6216	**0.6627**	0.6546	0.6319	**0.6778**
Color Jittering	0.6742	0.6396	**0.6878**	0.6874	0.6506	**0.713**

Table 3 summarizes their results. We report both of the top-1 and the top-5 accuracy, which is a common practice for (Tiny) ImageNet. In general, our PR-Net shows the best accuracy. PR-Net achieves an top-1 accuracy of 0.6157 with 4.56M parameters. The full Isometric MobileNet V3 marks an top-1 accuracy of 0.6578 with 20M parameters and the reduced Isometric MobileNet V3 with 4.30M parameters shows an top-1 accuracy of 0.6076. Considering the large difference on the number of parameters, PR-Net's efficiency is high. In particular, it outperforms others in the top-5 accuracy by non-trivial margins, e.g., 0.7911 of ODE-Net vs. 0.8115 of Isometric MobileNet V3 vs. 0.8357 of PR-Net. In addition, PR-Net shows faster forward-pass inference time in comparison with ODE-Net. The inference time is to classify a batch of 1,000 images.

We also compare our method with MobileNet V3 and ODE-Net for CIFAR10/100. For them, we use the same architecture we used for Tiny ImageNet. The detailed results are summarized in Table 4. Our PR-Net shows the best accuracy in both data sets. ODE-Net shows sub-optimal outcomes.

Table 4: Image classification in CIFAR10/100

Data	Name	Acc.	Name	Acc.	Name	Acc.
CIFAR10	M.Net V3	0.9407	ODE-Net	0.9350	PR-Net	**0.9457**
CIFAR100	M.Net V3	0.7603	ODE-Net	0.7243	PR-Net	**0.7683**

4.3 Experiments on Robustness with Tiny ImageNet

To check the efficacy of learning a governing equation, we conduct three additional experiments with Tiny ImageNet: i) out-of-distribution image classification, ii) adversarial attack robustness, and iii) transfer learning to other image datasets. We consider these experiments as extrapolations in image classification, where learning governing equations can be a key. In the first and second experiments, we apply many augmentation/perturbation techniques to generate out-of-distribution/adversarial images and check how each model responds to them. Being inspired by the observations that robust models are better transferred to other datasets (Engstrom et al., 2019a; Allen-Zhu & Li, 2020; Salman et al., 2020), in the third experiment, we check the transfer learning accuracy to other image datasets. We hypothesize that PR-Net shows better robustness than others, since it knows the governing equation for classifying Tiny ImageNet (as in Appendix C for a scientific PDE problem).

Neural networks are typically vulnerable to out-of-distribution and adversarial samples (Shen et al., 2016; Azulay & Weiss, 2019; Engstrom et al., 2019b). More fitted to training data, they typically show less robustness to out-of-distribution and adversarial samples. However, PR-Net follows learned governing equations to process those samples. As such, learning a governing equations can be deemed to implant latent knowledge governing the classification process.

Table 5: Adversarial attacks in Tiny ImageNet

Attack Method	M.Net V3	ODE-Net	PR-Net	M.Net V3	ODE-Net	PR-Net
	Top-1 accuracy			Top-5 accuracy		
FGSM($\epsilon = 0.5/255$)	0.3860	0.3656	**0.4041**	0.6492	0.6398	**0.6911**
FGSM($\epsilon = 1/255$)	0.2304	0.2287	**0.2499**	0.4751	0.4928	**0.5374**
FGSM($\epsilon = 3/255$)	0.0452	**0.0464**	0.0369	0.1232	0.1562	**0.1596**
PGD ($\epsilon = 0.5/255$)	0.3733	0.3525	**0.3910**	0.6508	0.6409	**0.6936**
PGD ($\epsilon = 1/255$)	0.1902	0.1908	**0.2133**	0.4579	0.4810	**0.5281**
PGD ($\epsilon = 3/255$)	0.0218	**0.0235**	0.017	0.0792	0.1093	**0.1144**

Out-of-Distribution Image classification. We use four image augmentation methods: i) adding a Gaussian noise of $\mathcal{N}(0, 0.1)$, ii) cropping a ceter area by size 56×56 and resizing to the original size, iii) rotating into a random direction for 30 degree, and iv) perturbing colors through randomly jittering the brightness, contrast, saturation, and hue with a strength coefficient of 0.2. All these are popular out-of-distribution augmentation methods (Shen et al., 2016; Azulay & Weiss, 2019; Engstrom et al., 2019b).

Our PR-Net shows the best accuracy (i.e., robustness) in the all cases of Table 3. In comparison with ODE-Net, it shows much better robustness, e.g., 0.3812 of ODE-Net vs. 0.4429 of PR-Net for the color jittering augmentation. One interesting point is that all methods are commonly more vulnerable to the random rotation and the color jittering augmentations than the other two augmentations.

Adversarial Attack Robustness. Since the governing equation regularizes PR-Net's behaviors, it can be made robust to unknown adversarial samples. We use FGSM (Goodfellow et al., 2015) and PGD (Madry et al., 2018) to find adversarial samples, to which the robustness of PR-Net is reported in Table 5. With various settings for the key parameter ϵ that controls the degree of adversarial

perturbations, we generate adversarial samples. As the configuration of doubling the number of channels used in each layer, denoted as "Width Multiplier 2," showed better performance in Table 3, we use that configuration for adversarial attack and transfer learning experiments. For all attacks except FGSM($\epsilon = 3/255$) and PGD ($\epsilon = 3/255$), PR-Net shows the best robustness in Table 5. The performance of PR-Net is superior to others in many cases.

Transfer Learning. As reported in (Engstrom et al., 2019a; Allen-Zhu & Li, 2020; Salman et al., 2020), robust models tend to produce feature maps suitable for transfer learning than regular models do. In this regard, we check the transferability of the pre-trained PR-Net for Tiny ImageNet to other datasets: CIFAR100 (Krizhevsky, 2009), CIFAR10 (Krizhevsky, 2009), FGVC Aircraft (Maji et al., 2013), Food-101 (Bossard et al., 2014), DTD (Cimpoi et al., 2014), and Cars (Yang et al., 2015). As shown in Table 6, PR-Net shows the best transfer learning accuracy in all cases except Cars. The improvements over M.Net V3 and ODE-Net are significant for Aircraft and DTD.

Table 6: Transfer learning in Tiny ImageNet

Dataset	M.Net V3	ODE-Net	PR-Net	M.Net V3	ODE-Net	PR-Net
	Top-1 accuracy			Top-5 accuracy		
CIFAR100	0.7676	0.7460	**0.7750**	0.9320	0.9270	**0.9480**
CIFAR10	0.9403	0.9280	**0.9418**	**0.9963**	0.9928	0.9962
Aircraft	0.6233	0.6027	**0.6612**	0.8509	0.8300	**0.8561**
Food-101	0.7317	0.7128	**0.7366**	0.9108	0.9036	**0.9174**
DTD	0.4819	0.4973	**0.5113**	0.7660	0.7465	**0.7957**
Cars	**0.6313**	0.5576	0.6283	**0.8380**	0.7998	0.8319

4.4 Feature Map Analyses

We also analyze the feature maps created by MobileNet V3, ODE-Net, and PR-Net in Figure 5 in Appendix B. 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) was also 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 MobileNet 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)[2]. As shown in Figure 5 in Appendix B, our PR-Net shows the best inversion quality.

Table 7: The silhouette score of clustering feature maps

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**

Figures 6 and 7 in Appendix B visualize the feature maps of ResNet, ODE-Net, and PR-Net for MNIST and SVHN using t-SNE (Maaten & Hinton, 2008). In terms of human visual perception, they all look similar. Therefore, we further employ the silhouette score to evaluate the quality of clusters on t-SNE embeddings, where the number of clusters denotes the number of classes. PR-Net shows the best clustering outcomes in Table 7, e.g., a silhouette score of 0.4959 for ResNet in MNIST vs. 0.4991 for ODE-Net vs. 0.5053 for RK-Net vs. 0.5079 for PR-Net.

4.5 Training Overhead

Our proposed PR-Net has additional parts to be considered during its training process, e.g., governing equation. As such, our method requires more resources in comparison with other baselines. However, training occurs 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.

Table 8: Training overhead in terms of GPU memory usage (MB) and 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 8 summarizes the training overhead for 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

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

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 problem that calculates 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 uses the standard backpropagation through the RK4 solver, takes much less time than ODE-Net.

The overhead of Tiny ImageNet is summarized in Table 9. As expected, PR-Net requires 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 was also observed for MNIST and SVHN. While the reverse-mode integral of the adjoint sensitivity method has a space complexity of $\mathcal{O}(1)$, in any case it needs to solve an integral problem, which incurs additional time complexity.

Table 9: Training overhead in terms of GPU memory usage (MB) and 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

Figure 4 illustrates the curves of $L_T, \hat{L}_I, \hat{L}_G$ for MNIST. 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. On the contrary, the task loss L_T decreases much faster, which shows the difficulty of learning physical dynamics (i.e., governing equation) governing classification procedures.

5 Discussions & Conclusions

It recently became popular to design neural networks based on differential equations. In most cases, ODEs are used to approximate neural networks. In this work, on the other hand, we presented a PDE-based approach to design neural networks. Our method simultaneously learns a regression model and a governing equation that conform with each other. Therefore, the internal processing mechanism of the learned regression model should follow the learned governing equation. One can consider that this mechanism is a sort of implanting domain knowledge into the regression model. The main challenge in our problem definition is that we need to discover a governing equation from data while training a regression model. Thus, we adopt a joint training method of the regression model and the governing equation.

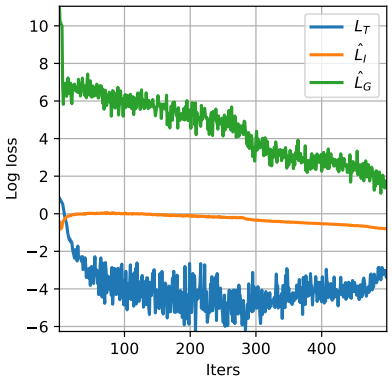


Figure 4: The curves of log loss values decrease as training goes on for MNIST

To show the efficacy, we conducted five experiments: i) MNIST/SVHN classification, ii) Tiny ImageNet and CIFAR10/100 classification, iii) classification with out-of-distribution samples, iv) adversarial attack robustness, and v) transfer learning. Our method shows the best accuracy and robustness (or close to the best) except only SVHN. In particular, the challenging robustness experiments empirically prove why learning an appropriate governing equation is important.

One limitation on this method is that it is sometimes hard to achieve a good trade-off among all different loss and regularization terms. Our method intrinsically involves various terms and we found that it is important to tune hyperparameters (especially for various coefficients and learning rates) in order to achieve reliable performance. In particular, $\alpha_{i,j}$, for all i, j, are important in learning reliable governing equations. Because the trained network f is greatly influenced by the governing equation, hyperparameters should be carefully tuned to learn meaningful governing equations.

6 ETHICS STATEMENT

Image classification is one of the most popular topics in deep learning and has many applications. In general, it has more benefits than risks. As such, we do not see any ethical issues in our research.

7 REPRODUCIBILITY STATEMENT

We include detailed information on reproducibility in Appendix. In particular, we clarify the neural network architectures we used and their best hyperparameter sets for all experiments in our paper. We also submit our source codes and data with associated shell scripts to easily reproduce some selected results, which comply with the 100MB limitation of the supplementary zip file.

REFERENCES

Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning, 2020.

Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *J. Mach. Learn. Res.*, 20:184:1–184:25, 2019.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *NeurIPS*, pp. 4502–4510, 2016.

Karianne J Bergen, Paul A Johnson, V Maarten, and Gregory C Beroza. Machine learning for data-driven discovery in solid earth geoscience. *Science*, 363(6433), 2019.

Lukas Bossard, M. Guillaumin, and L. Gool. Food-101 - mining discriminative components with random forests. In *ECCV*, 2014.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*. 2018.

Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1256–1272, 2016.

Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *CVPR*, 2015.

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *ICLR*, 2020.

Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. In *NeurIPS*, 2018.

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR Deep Differential Equations Workshop*, 2020.

Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolated Adjoint Method for Neural ODEs. *arXiv:2003.05271*, 2020.

Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *ICLR*, 2018.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations, 2019a.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling CNNs with simple transformations. In *ICLR*, 2019b.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, 2020.

Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *NeurIPS*, 2019.

Eldad Haber, Keegan Lensink, Eran Treister, and Lars Ruthotto. Imexnet a forward stable deep neural network. In *ICML*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

Andrew Howard, Mark Sandler, Grace Chu, Lian-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019.

Xixi Jia, Sanyang Liu, Xiangchu Feng, and Lei Zhang. Focnet: A fractional optimal control network for image denoising. In *CVPR*, 2019.

Hwajoon Kim. The solution of the heat equation without boundary conditions. *Dynamic Systems and Applications*, 27:653–662, 08 2018.

Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Kookjin Lee and Eric J Parish. Parameterized neural ordinary differential equations: Applications to computational physics problems. *arXiv preprint arXiv:2010.14685*, 2020.

Chunming Li, Chenyang Xu, Changfeng Gui, and Martin D Fox. Level set evolution without re-initialization: a new variational formulation. In *CVPR*, 2005.

Risheng Liu, Zhouchen Lin, Wei Zhang, and Zhixun Su. Learning pdes for image restoration via optimal control. In *ECCV*. Springer, 2010.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-net: Learning PDEs from data. In *ICML*, 2018.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *ICML*, 2018.

Laurens Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft, 2013.

A. Mas-Colell. *Cooperative Equilibrium*, pp. 95–102. Palgrave Macmillan UK, London, 1989.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In *NeurIPS*, 2020.

Stanley Osher and Leonid I Rudin. Feature-oriented image enhancement using shock filters. *SIAM Journal on numerical analysis*, 27(4):919–940, 1990.

Wei Peng, W. Zhou, Jun Zhang, and Wenbing Yao. Accelerating physics-informed neural network training with prior dictionaries. *ArXiv*, abs/2004.08151, 2020.

Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

Hans Pinckaers and Geert Litjens. Neural ordinary differential equations for semantic segmentation of individual colon glands. In *NeurIPS Workshops*, 2019.

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 19(25):1–24, 2018.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2019.

Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? In *NeurIPS*, 2020.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.

Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *ICCV Workshops*, 2019.

Xu Shen, Xinmei Tian, Anfeng He, Shaoyan Sun, and Dacheng Tao. Transform-invariant convolutional neural networks for image classification and search. In *MM*, 2016.

E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Andrew P Witkin. Scale-space filtering. In *Readings in Computer Vision*, pp. 329–332. Elsevier, 1987.

L. Yang, P. Luo, C. C. Loy, and X. Tang. A large-scale car dataset for fine-grained categorization and verification. In *CVPR*, 2015.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *ICLR*, 2020.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *ICML*, 2020.

A PDEs AND COMPUTER VISION

After the introduction of scale space (Koenderink, 1984; Witkin, 1987), PDE-based methods have been actively studied for computer vision and image processing tasks. Starting from anisotropic diffusion for denoising developed by Perona and Malik (Perona & Malik, 1990), PDEs have been utilized in many different tasks including image enhancement via shock filters (Osher & Rudin, 1990), image segmentation (Li et al., 2005), and total-variation-based noise removing (Rudin et al., 1992). Following these approaches, in the data-driven era, instead of hand-crafted PDEs, there have been many efforts to learn PDEs from data and utilize learned PDEs in e.g., image restoration (Liu et al., 2010; Chen & Pock, 2016; Chen et al., 2015). More recently, PDE-inspired deep learning architectures have been developed and applied to image classification (Lu et al., 2018; Ruthotto & Haber, 2019; Haber et al., 2019), and image denoising (Jia et al., 2019).

B FEATURE MAP ANALYSES

We introduce the feature map inversion quality in Figure 5 and the clustering results after projecting each feature map onto a two-dimensional space with t-SNE in Figures 6 and 7.

C IMPORTANCE OF LEARNING GOVERNING EQUATIONS

Here, we demonstrate the importance of learning governing equations 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:

$$g(d,t) = h_t - 0.0001h_{dd} + 5h^3 - 5h = 0, d \in [-1,1],\ t \in [0,1], \tag{10}$$

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:

$$\arg\min_{\boldsymbol{\theta}}\ L_I + L_B + L_R,$$
$$L_R \overset{\text{def}}{=} \frac{1}{N_R}\sum_{(d,t)}\big(f(d,t;\boldsymbol{\theta}) - h(d,t)\big)^2,$$

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 8 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 8 (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 8 (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.

(a) M.Net V3 (b) ODE-Net (c) PR-Net (d) Original

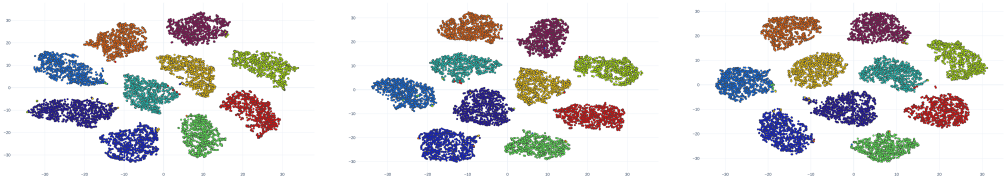
(e) M.Net V3 (f) ODE-Net (g) PR-Net (h) Original

(i) M.Net V3 (j) ODE-Net (k) PR-Net (l) Original

(m) M.Net V3 (n) ODE-Net (o) PR-Net (p) Original

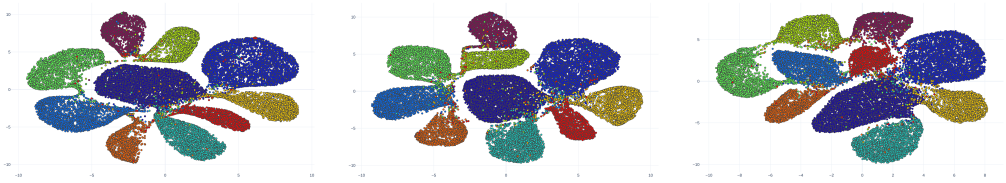
(q) M.Net V3 (r) ODE-Net (s) PR-Net (t) Original

Figure 5: Representation inversion with Tiny ImageNet



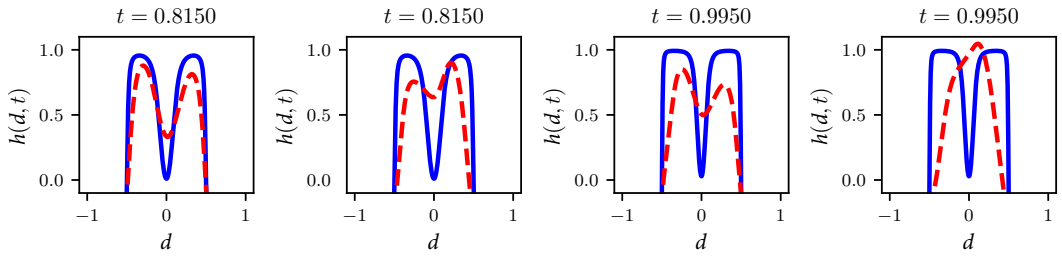
(a) ResNet (b) ODE-Net (c) PR-Net

Figure 6: The visualization of feature maps for MNIST. We use t-SNE to project the feature maps onto a two-dimensional space.



(a) ResNet (b) ODE-Net (c) PR-Net

Figure 7: The visualization of feature maps for SVHN. We use t-SNE to project the feature maps onto a two-dimensional space.



(a) G.E. Aware (b) G.E. Ignorant (c) G.E. Aware (d) G.E. Ignorant

Figure 8: 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.

Table 10: 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		

D PROOF

Theorem D.1. *Given a learning task, let $\boldsymbol{\theta}^*$ and $\alpha^*_{i,j}$, for all i, j, constitute a cooperative equilibrium solution and governing equation (in terms of $L_T + \hat{L}_I + \hat{L}_G + R_G$) — in other words, we cannot further decrease $L_T + \hat{L}_I + \hat{L}_G + R_G$ 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. 11 — note that the following terms appear in the right-hand side of Eq. 1:

$$\begin{aligned}&\alpha_{0,0} + \alpha_{1,0} f + \alpha_{2,0} f^2 + \alpha_{3,0} f^3 + \alpha_{0,1} f_d + \alpha_{1,1} f f_d \\ &+ \alpha_{2,1} f^2 f_d + \alpha_{3,1} f^3 f_d + \alpha_{0,2} f_{dd} + \alpha_{1,2} f f_{dd} + \alpha_{2,2} f^2 f_{dd} \\ &+ \alpha_{3,2} f^3 f_{dd} + \alpha_{0,3} f_{ddd} + \alpha_{1,3} f f_{ddd} + \alpha_{2,3} f^2 f_{ddd} + \alpha_{3,3} f^3 f_{ddd}\end{aligned} \tag{11}$$

For Eq. 11 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 in many cases, e.g., using softplus instead of ReLU. 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 sparsest governing equation that minimizes the loss. Therefore, its solution can be also uniquely defined as our training pursues it.

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

E IMAGE CLASSIFICATION WITH MNIST AND SVHN

We describe detailed experimental environments. Table 10 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 10. We note that PR-Net has the same output size as that of ODE-Net. Refer to Section J about how we construct $\boldsymbol{h}^{\text{task}}$ with the set H of (d, t) pairs.

Table 11: 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.

F IMAGE CLASSIFICATION WITH TINY IMAGE NET & CIFAR10/100

We describe detailed experimental environments. Table 11 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 11 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 11. We note that PR-Net has the same output size as that of ODE-Net. Refer to Section J 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.

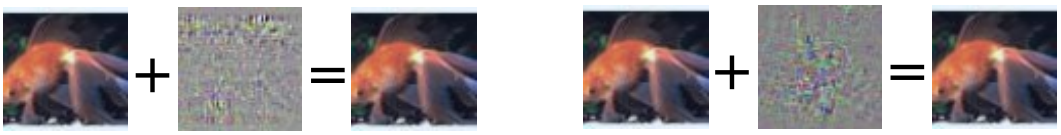
G ADVERSARIAL ATTACK WITH TINY IMAGENET

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 9: Out-of-distribution examples



(a) FGSM (b) PGD

Figure 10: 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).

H 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.

I Out-of-distribution and Adversarial Image Samples

We introduce a selected set of images that we produced for our robustness experiments. Figure 9 shows a set of image samples for the out-of-distribution image classification and Figure 10 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 torch with a confidence of 0.3546 and the perturbed image by PGD is predicted as candle with a confidence of 0.4810.

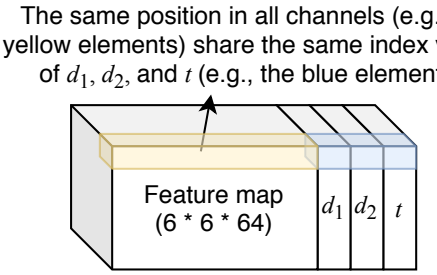


Figure 11: An illustration for MNIST/SVHN on how to increase the processing efficiency by discretizing the last dimension and share (d, t) pairs.

Table 12: 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

J Discretizing Feature Map Dimensions for Efficient Processing

One more advantage of using PDEs is that we can discretize some dimensions[3]. 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 11 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.

K Additional Experimental Results – Per-class accuracy

All datasets we used are summarized in Table 12. 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 13, 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

[3] 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.

Table 13: 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	0.4466	**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 14: 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**

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 14. 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 6. We summarize their means and standard deviations of per-class accuracy in Table 15. 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 15: 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