Stochasticity in Neural ODEs: An Empirical Study

Viktor Oganesyan[*1] **Alexandra Volokhova**[*12] **Dmitry Vetrov**[13]

ABSTRACT

Stochastic regularization of neural networks (e.g. dropout) is a wide-spread technique in deep learning that allows for better generalization. Despite its success, continuous-time models, such as neural ordinary differential equation (ODE), usually rely on a completely deterministic feed-forward operation. This work provides an empirical study of stochastically regularized neural ODE on several image-classification tasks (CIFAR-10, CIFAR-100, TinyImageNet). Building upon the formalism of stochastic differential equations (SDEs), we demonstrate that neural SDE is able to outperform its deterministic counterpart. Further, we show that data augmentation during the training improves the performance of both deterministic and stochastic versions of the same model. However, the improvements obtained by the data augmentation completely eliminate the empirical gains of the stochastic regularization, making the difference in the performance of neural ODE and neural SDE negligible.

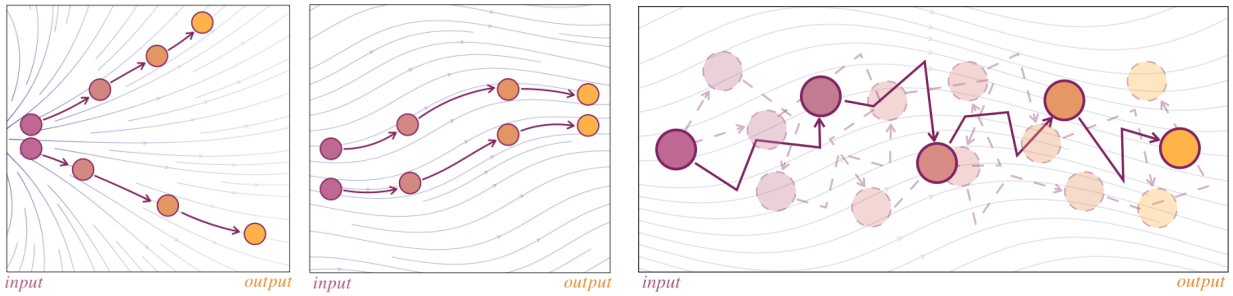
1 Introduction

Deep neural networks describe an expressive parametric family of functions, which can be used for building predictive models in many machine learning tasks. However, the expressive power of neural networks comes at the cost of potential overfitting on the training data. To prevent this undesired behavior many regularization techniques have been developed and successfully applied (l2-regularization, dropout, early stopping, batch normalization) [1, 2, 3].

Recently, continuous-time neural models (neural ODE [4]) have attracted the attention of the community. In contrast to the conventional deep learning models, they operate by parameterizing an ordinary differential equation (ODE) with a neural network. This approach is demonstrated to be promising in the design of normalizing flows [5] and time-series generative models [6, 7, 8, 9] Moreover, continuous-time models directly generalize ResNet architecture [10], which is known to be efficient in practice by preventing gradient saturation and allowing for performance gains with an increase of the depth. For supervised learning (e.g. image classification), this generalization promises such benefits as parameter-efficiency and adaptive computational time [4]. Despite the numerous theoretical benefits, continuous-time models lack empirical analysis and design guidelines, which significantly hinders the development of novel models and their application to practical tasks.

In this paper, we provide an empirical study of the stochastic regularization of neural ODE. An essential way to introduce stochastic into a neural ODE is to extend it to a neural stochastic differential equation (neural SDE) [9, 8, 11, 12, 13]. The intuitive motivation for this extension is the following. As well as in conventional deep learning models, putting noise on the intermediate representations helps us to foster the generalization abilities of the model. An overfitted continuous-time model could be represented as a highly divergent vector field, where small perturbations of initial conditions may result in completely different end-points of the dynamics (see Fig. 1a). In contrast, the continuous-time model that generalizes well should map close initial points to similar outputs guaranteeing the robustness to small perturbations (see Fig. 1b). The input of neural SDE follows one random trajectory from a set of neighboring ones (Fig.1c). Whilst the model learns to predict a correct answer for the input regardless of which particular trajectory it follows. This encourages neural SDE to learn that neighboring trajectories should lead to close outputs, which actually prevents divergence. Thus, introducing the stochasticity at the training stage, we foster the model to learn such parameters that allow for the robust feed-forward procedure.

[1]Samsung-HSE Laboratory, National Research University Higher School of Economics
[2]Moscow Institute of Physics and Technology
[3]Samsung AI Center Moscow
[*]Equal contribution. Correspondence to: Alexandra Volokhova avolohova@hse.ru, Viktor Oganesyan voganesyan@hse.ru
Workshop on Integration of Deep Neural Models and Differential Equations, ICLR 2020



(a) (b) (c)

Figure 1: Illustrative trajectories of integration inside the neural ODE (a, b) and the neural SDE (c). Neural ODE performs an integration on a forward pass according to a dynamic function defined by a neural net. Thus, the input of this model follows some trajectory during the integration. Figure (a) shows trajectories into the neural ODE with the dynamic function, which causes a highly divergent vector field. In this case, similar inputs are mapped into significantly different outputs. In contrast to mapping performed by the neural ODE with non-divergent vector field (b), where integration preserves similarity of inputs. Plot (c) illustrates the stochastic nature of trajectories inside the neural SDE. There, the input passes a random trajectory from a set of possible ones, while the model learns to make a correct prediction. We assume this stochasticity encourages the vector field of neural SDE to be less divergent, which leads to better robustness and prevents overfitting.

We study continuous models on three image classification tasks: CIFAR-10, CIFAR-100 [14], TinyImageNet [15]. As a starting point of our study, we compare neural ODE with ResNet. We put both models in equal conditions (in terms of architecture and regularization) and observe that they perform similarly. Further, we introduce stochasticity into the neural ODE using the formalism of stochastic differential equations. We find out that this procedure can regularize neural ODE. However, our experiments show that common data augmentation allows neural ODE to achieve better generalization than the introduction of stochasticity. Therefore, we see that perturbing representations with data augmentation is enough for learning a robust model that generalizes well.

2 Considered models

We conduct experiments with three types of models: residual network, neural ordinary differential equation, neural stochastic differential equation. All these models can be regarded as the integration of a differential equation with some integration scheme. One block of a residual network performs the following mapping:

$$z_{out} = z_{in} + f_\theta(z_{in}), \tag{1}$$

there f_θ is a neural network with parameters θ, z_{in} and z_{out} are an input and an output of a residual block. This mapping corresponds to a one-step Euler method for numerical integration.

Neural ODE extends this idea allowing us to use any numerical integration scheme:

$$dz = f_\theta(z)dt \qquad z_{out} = ODESolver(z_{in}, f_\theta, t_{begin}, t_{end}), \tag{2}$$

there t_{begin} and t_{end} are bounds of integration, $ODESolver$ is some numerical integration method. Further, introduction a stochastic term to the differential equation leads to a neural SDE:

$$dz = f_\theta(z)dt + \sigma dW \qquad z_{out} = SDESolver(z_{in}, f_\theta, \sigma, t_{begin}, t_{end}), \tag{3}$$

where dW is the vector stochastic Wienner process of the same dimensionality as z, σ is a scalar magnitude of stochasticity, $SDESolver$ is a numerical method for SDE integration.

We designed models with a similar architecture to provide an objective and reliable comparison of residual networks, neural ODE, and neural SDE. Our neural networks basically consist of several sequences of blocks, which perform integration according to eq. (1), (2) or (3). These integration blocks are separated by down-sampling blocks. Therefore the main considered models are ResNet (with sequences of residual blocks), ODENet (with sequences of neural ODEs), SDENet (with sequences of neural SDEs). All models have the same architecture except for the type of integration blocks. Moreover, ResNet, ODENet and SDENet have similar functional form of a dynamic function f_θ inside the integration blocks.

3 Experiments

The main purpose of our experiments is to introduce stochasticity into continuous models and to investigate its regularisation properties. Additionally, we compare continuous models with a baseline residual network. As the first step in our study, we compare considered models on a toy task and observe that stochasticity performs as a good regularizer. Inspired by promising results, we continue comparison on CIFAR-10, CIFAR-100, and TinyImageNet.

Besides possible regularization properties, neural SDE provides an opportunity to improve quality with averaging predictions. Indeed, one can run trained SDENet n times on test data and average predictions, obtained from different random trajectories. Furthermore, it is possible to train a continuous model in a stochastic mode integrating eq.(3) and switch to a deterministic mode during a test-time evaluation by replacing the integration scheme to eq.(2), which considers only deterministic dynamics f_θ. We explore these opportunities in our experiments.

It should be noted that neural SDE models contain σ as a hyperparameter, which we choose using grid search (details of the search and chosen values can be found in Appendix). Moreover, we use common back-propagation instead of adjoint method [4] to compute gradients during the training of continuous models, because of numerical instability of the adjoint method [16].

We published our code at `https://github.com/AlexandraVolokhova/stochasticity_in_neural_ode`

3.1 Toy dataset

We consider a binary classification task with samples from two 10-dimensional Gaussians. The distance between their centers equals 3.0, each Gaussian has an identity covariance matrix. This toy task has an optimal solution, which achieves 93.3% accuracy.

The results are presented in Table 1. Our experiments show that ResNet and ODENet reach almost the same accuracy. In contrast to them, SDENet is able to achieve much better results reaching an almost optimal solution. It is interesting to note, that in the deterministic test-time mode, SDENet performs as well as averaging along 5-10 trajectories, but requiring less time to compute predictions.

Table 1: Test accuracy of considered models on the toy dataset. We repeat each experiment 5 times with different random seed and report mean $\pm$ standard deviation in percentages. SDENet_0 denotes the deterministic test-time mode, SDENet_n ($n > 0$) denotes averaging predictions along n stochastic trajectories during the test-time evaluation. 'Optimum' means the accuracy of the theoretical optimal solution

ResNet	ODENet	SDENet_0	SDENet_1	SDENet_2	SDENet_5	SDENet_10	SDENet_20	*Optimum*
90.6 ± 0.2	90.8 ± 0.9	92.6 ± 0.3	91.2 ± 0.2	92.0 ± 0.2	92.5 ± 0.3	92.7 ± 0.3	92.8 ± 0.3	93.3

3.2 CIFAR-10, CIFAR-100, and Tiny ImageNet

The considered models are trained on three image classification tasks: CIFAR-10, CIFAR-100, and TinyImageNet. We design ResNet, ODENet and SDENet similarly, except for the type of integration blocks, which are residual blocks, neural ODEs or neural SDEs (see Appendix, Fig.6). Integration is carried out inside ODENet and SDENet by the Runge-Kutta fourth-order method. We conduct our experiments with and without data augmentation in order to compare models in various conditions. The results of our experiments are presented in Table 2 in Appendix and on Figures 2, 3, 4.

ResNet v.s. ODENet According to our experiments, both model performs almost similarly, with a occasional superiority of ResNet (see Fig.2 and Table 2 in Appendix). Since the main difference between ResNet and ODENet is the numerical integration method (Euler v.s. fourth-order Runge-Kutta), we observe that more precise one does not improve the quality of classification. This result is quite reasonable because precise integration in the neural ODE restricts possible mappings to a set of homogeneous ones [17], while rude integration in ResNet allows for breaking homogeneity. Therefore, taking into account [17] and our experiments we conclude that precise integration does not increase the expressiveness of the model.

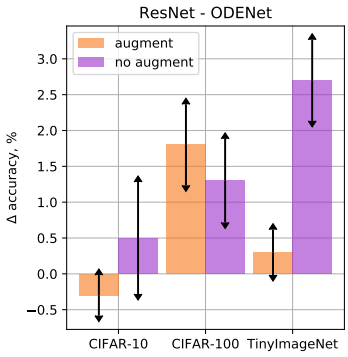
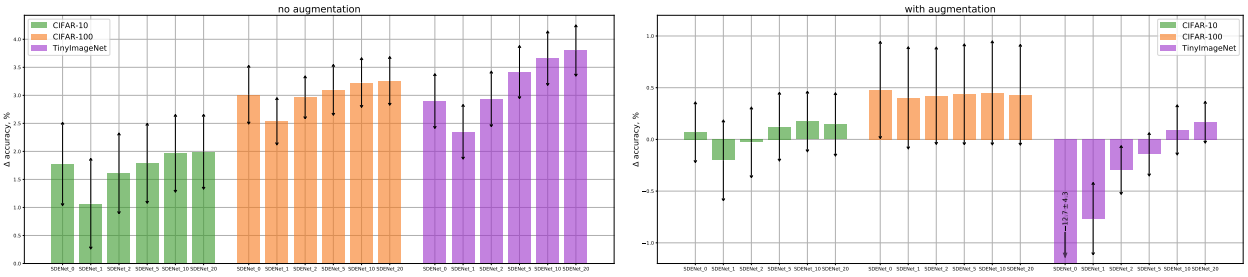


Figure 2: Difference between accuracy of ResNet and ODENet on three classification tasks. Colored bars show mean values of differences averaged by 5 runs, error bars demonstrate the standard deviation.



(a) (b)

Figure 3: Difference between accuracy of SDENet and ODENet on three classification tasks. Figures 3a and 3b present results of experiments in settings without augmentation and with augmentation correspondingly. Colored bars show mean values of differences averaged by 5 runs, error bars demonstrate the standard deviation. SDENet_0 denotes the deterministic test-time mode, SDENet_n ($n > 0$) denotes averaging predictions along n stochastic trajectories during the test-time evaluation. There is a significant negative value for the difference between SDENet_0 and ODENet on TinyImageNet with augmentation. We don't fully depict the corresponding bar for illustrative reasons.

Regularization properties of a neural SDE We observe that the introduction of stochasticity into a neural ODE improves its generalization. SDENet consistently achieves better accuracy than ODENet in our experiments without augmentation (see Fig.3a and Table 2). However, analogous experiments with augmentation show that neural ODE and neural SDE perform similarly (see Fig.3b and Table 2). Additionaly, ODENet with augmentation considerably outperforms SDENet without augmentation (see Table 2 and Fig.5 in Appendix). Therefore, we conclude that stochasticity is actually able to regularize neural ODE, but simple data augmentation does it significantly better. Moreover, an additional regularization effect from stochasticity is immaterial if data augmentation is used in the training procedure.

In addition, we observe that averaging predictions along stochastic trajectories improves performance. As one can see from Fig.3, the quality of predictions continuously rises with an increase in the number of random trajectories in averaging. This result is reasonable because stochasticity is independently introduced to each trajectory, therefore averaging along them is able to reduce variance term of expected generalization error. Moreover, switching to the deterministic mode at test-time performs mostly as averaging along several trajectories (this mode referred to as SDENet_0 at Fig.3). However, SDENet_0 significantly loses quality in experiments on TinyImageNet with augmentation. We assume this effect can be explained as follows. The prediction averaged along the trajectories is an unbiased estimation of the prediction expected over the trajectories. Also, the deterministic trajectory in neural ODE is the expected trajectory in corresponding neural SDE due to the properties of the Wienner process. As far as the mapping from a trajectory to a prediction is a non-linear function, then the prediction based on the deterministic trajectory is a biased estimation to the expected prediction. The value of this bias depends on the variance of trajectories and properties of the non-linear function, that sometimes may lead to drops in the quality at the test time. For this reason, we recommend using the deterministic test-time mode with caution for neural SDE.

Batch normalization inside a neural ODE Batch normalization [3] is a very common technique in modern neural networks. However, there is a significant difference between applying batch normalization in residual networks and neural ODEs. Common implementations of residual networks usually contain batch normalization inside residual blocks. However, if we put batch normalization into the dynamic function f_θ of neural ODE, then the same normalization will be applied to internal representations $z(t)$ at different time points t during numerical integration. In this case, moving averages of batch normalization will be accumulated along all steps of numerical integration. It is not clear how this affects the performance of neural ODEs. Therefore, we design ODENet and SDENet without batch normalization inside their dynamic functions. Additionally, we train ODENet with batch normalization inside its dynamic functions, which we refer to as ODENet+BN. Our experiments show that neural ODE with batch normalization occasionally works significantly worse than without batch normalization, which confirms our apprehension (see Fig.4 and Table 2). However, we suppose this effect should be investigated more thoroughly in the future.

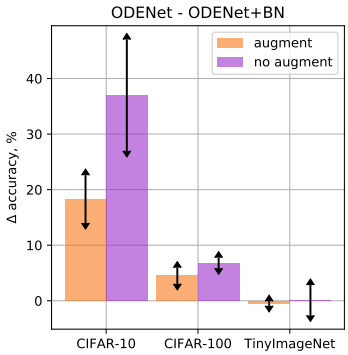


Figure 4: Difference between accuracy of ODENet and ODENet+BN on three classification tasks. Colored bars show mean values of differences averaged by 5 runs, error bars demonstrate the standard deviation.

4 Conclusion

We present an empirical study of neural ODEs and neural SDEs on various classification tasks. The main contribution of this paper is the exploration of regularization properties of neural SDE. We find out that stochastic term in the neural differential equation allows us to increase generalization of the model if we train it without data augmentation. However, when the model learns in a setting with data augmentation, additional stochasticity of the differential equation does not increase the quality. Hence, we conclude that stochasticity is not enough powerful regularizer for neural ODE in case of image classification. Nonetheless, neural SDE may be able to significantly improve quality on tasks, where data augmentation is hard to handle.

In addition, we compare the performance of continuous models and residual networks. We observe that ResNet occasionally works better than neural ODE on image classification tasks. That means more accurate integration, which is performed in the neural ODE, does not increase expressiveness. It is worth to note that more precise integration requires more computations, which leads to longer training and inference procedure. Taking into account our experiments, we would not recommend using continuous models for image classification tasks, as a residual network manages them better and more efficiently. Continuous models seem to be more applicable to time-series generative models, as it is reported in [6, 9].

Acknowledgments

This research is in part based on the work supported by Samsung Research, Samsung Electronics. Results on stochasticity in neural ODE have been supported by the Russian Science Foundation grant no.17-71-20072. This research was supported in part through computational resources of HPC facilities at NRU HSE. We sincerely thank Kirill Neklyudov and Arsenii Ashukha for their sound advice. Special thanks to Kirill for his considerable assistance in writing the text of this paper.

References

[1] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[5] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[6] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

[7] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 13412–13421, 2019.

[8] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9843–9854, 2019.

[9] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Belinda Tzen and Maxim Raginsky. Theoretical guarantees for sampling and inference in generative models with latent diffusions. *arXiv preprint arXiv:1903.01608*, 2019.

[12] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[13] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[15] *https://tiny-imagenet.herokuapp.com*.

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

[17] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Appendix

This section provides all the details of our experiments with CIFAR-10, CIFAR-100, and TinyImageNet, which are necessary for the possible reproduction of our results.

Detailed results of the main experiments

Tabel 2 contains test accuracies of considered models on image classification tasks. Fig.5 illustrates these results for easier comparison.

Table 2: Test accuracy of considered models on three classification tasks in different conditions. Columns 'augment' and 'no augment' contain results of experiments with augmentation and without it respectively. We repeat each experiment 5 times with different random seeds and report the mean $\pm$ standard deviation. SDENet_0 denotes the deterministic test-time mode, SDENet_n ($n > 0$) denotes averaging predictions along n stochastic trajectories during the test-time evaluation.

	CIFAR-10		CIFAR-100		TinyImageNet	
	augment	no augment	augment	no augment	augment	no augment
ResNet	90.6 ± 0.2	85.5 ± 0.5	76.5 ± 0.4	63.8 ± 0.5	50.4 ± 0.3	39.5 ± 0.4
ODENet	90.9 ± 0.2	85.0 ± 0.6	74.7 ± 0.4	62.5 ± 0.3	50.1 ± 0.1	36.8 ± 0.4
ODENet+BN	72.6 ± 4.3	48.0 ± 10.0	70.2 ± 1.4	55.7 ± 0.9	50.6 ± 0.4	36.7 ± 2.7
SDENet_0	90.9 ± 0.1	86.7 ± 0.4	75.2 ± 0.1	65.5 ± 0.4	37.4 ± 4.3	39.7 ± 0.2
SDENet_1	90.7 ± 0.3	86.0 ± 0.5	75.1 ± 0.2	65.0 ± 0.3	49.4 ± 0.3	39.1 ± 0.2
SDENet_2	90.8 ± 0.2	86.6 ± 0.4	75.2 ± 0.1	65.4 ± 0.2	49.8 ± 0.2	39.7 ± 0.2
SDENet_5	91.0 ± 0.2	86.7 ± 0.4	75.2 ± 0.2	65.6 ± 0.3	50.0 ± 0.2	40.2 ± 0.2
SDENet_10	91.0 ± 0.1	86.9 ± 0.3	75.2 ± 0.2	65.7 ± 0.3	50.2 ± 0.2	40.4 ± 0.2
SDENet_20	91.0 ± 0.2	87.0 ± 0.3	75.2 ± 0.2	65.7 ± 0.3	50.3 ± 0.2	40.6 ± 0.1

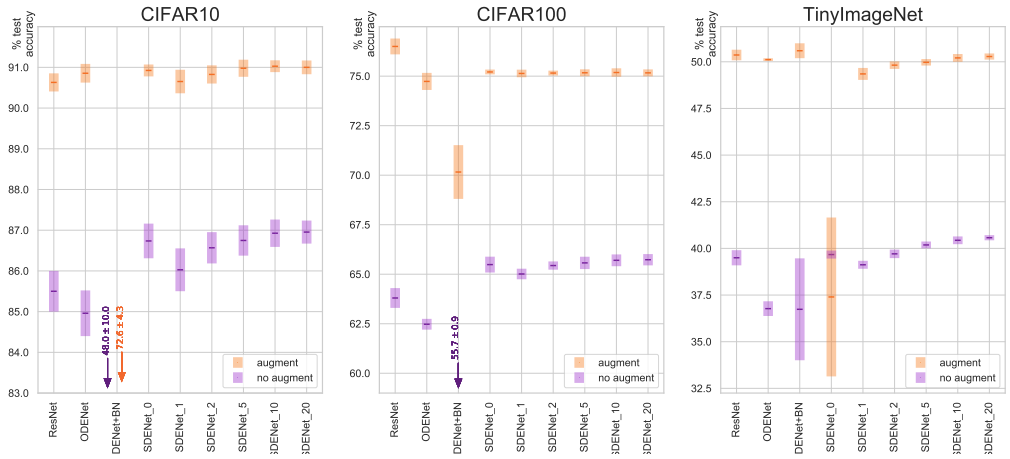


Figure 5: Illustrative representation of the figures from Table 2. Mean values are depicted as bold line segments and standard deviations are depicted as half-transparent rectangles. Outlying values are not shown for visualization reasons.

Hyperparameters and training procedure

We divide the training dataset into train and validation parts for hyperparameter search. After that, we choose hyperparameters, with those the model achieves better accuracy on the validation part. Finally, we join divided parts and train the final model on the full training dataset. The main hyperparameters, which we selected in this way, are σ, learning rate, and batch size.

We conduct our experiments using PyTorch [18] and torchdiffeq library [4]. We use stochastic gradient descent with Nesterov momentum equals 0.9 and learning rate scheduler `ReduceLROnPlateau` with

`threshold_mode='rel'`, `threshold=0.02`, `patience=10`, `factor=0.1`, `min_lr=1e-5`. Also, we use `WarmUpLR` with different `warmup_steps`. Learning rate (lr), warmup_steps (warm) and weight decay (wd) are different for different experiments (see Table 3 and 4). Other heperparameters are shown in the Table 5.

Table 3: Optimizer parameters no augmentation

	CIFAR-10			CIFAR-100			TinyImageNet		
	lr	warm	wd	lr	warm	wd	lr	warm	wd
ResNet	0.4	0	5e-4	0.4	2	5e-4	0.05	3	1e-4
ODENet	0.1	0	5e-4	0.1	2	5e-4	0.05	0	1e-5
ODENet+BN	0.05	0	5e-4	0.1	2	5e-4	0.1	0	1e-5
SDENet	0.05	0	5e-4	0.1	2	5e-4	0.05	0	1e-5

Table 4: Optimizer parameters with augmentation

	CIFAR-10			CIFAR-100			TinyImageNet		
	lr	warm	wd	lr	warm	wd	lr	warm	wd
ResNet	0.1	0	5e-4	0.2	2	5e-4	0.1	3	1e-4
ODENet	0.05	0	5e-4	0.1	2	5e-4	0.01	3	1e-4
ODENet+BN	0.05	0	5e-4	0.1	2	5e-4	0.05	3	1e-4
SDENet	0.05	0	5e-4	0.1	2	5e-4	0.01	3	1e-4

Table 5: Other hyperparameters. # steps denotes number of steps of numerical integration

	CIFAR-10	CIFAR-100	TinyImageNet
batch size	512	256	256
# steps for ODENet and SDENet	10	3	2
# steps for ODENet+BN	6	3	2
σ (no augment)	0.79	0.25	0.4
σ (with augment)	0.2	0.05	0.3

It is important to note that TinyImageNet from [15] initially consists of three sets: train, validation, and test. Train and validation sets are labeled and the test set does not have labels. Therefore, we use only the train set to train our models and validation set for the final evaluation.

Architectures

Figures 6 and 7 depict architectures of our models for each dataset. We design our models to be strong enough to achieve adequate accuracy on the test set and 100% accuracy on the training set. Hence, our models are able to overfit it is reasonable to study regularizers with them.

We design residual networks and continuous models in a similar way, so ResNet differs from ODENet and SDENet only in the type of integration block see Fig.7). ResBlock is used in ReNet, ODEBlock is used in ODENet and SDENet.

Continuous models have the same architecture, but SDENet differs from ODENet in the stochastic term in eq.3. So, these models differ also in solvers of neural differential equations.

As we mentioned in the section 3.2, we do not put batch normalization into dynamic function f of ODENet and SDENet. However, in our experiment with ODENet+BN, we add batch normalization layers into f in the same way as we do for ResBlock.

Additionally, dynamic function f in ODEBlock depends on time t. We design this dependency adding one extra channel to the internal representation $z(t)$. This extra channel is filled by the value of time t. Therefore, convolution layers have `input_channels = output_channels + 1` in ODEBlock.

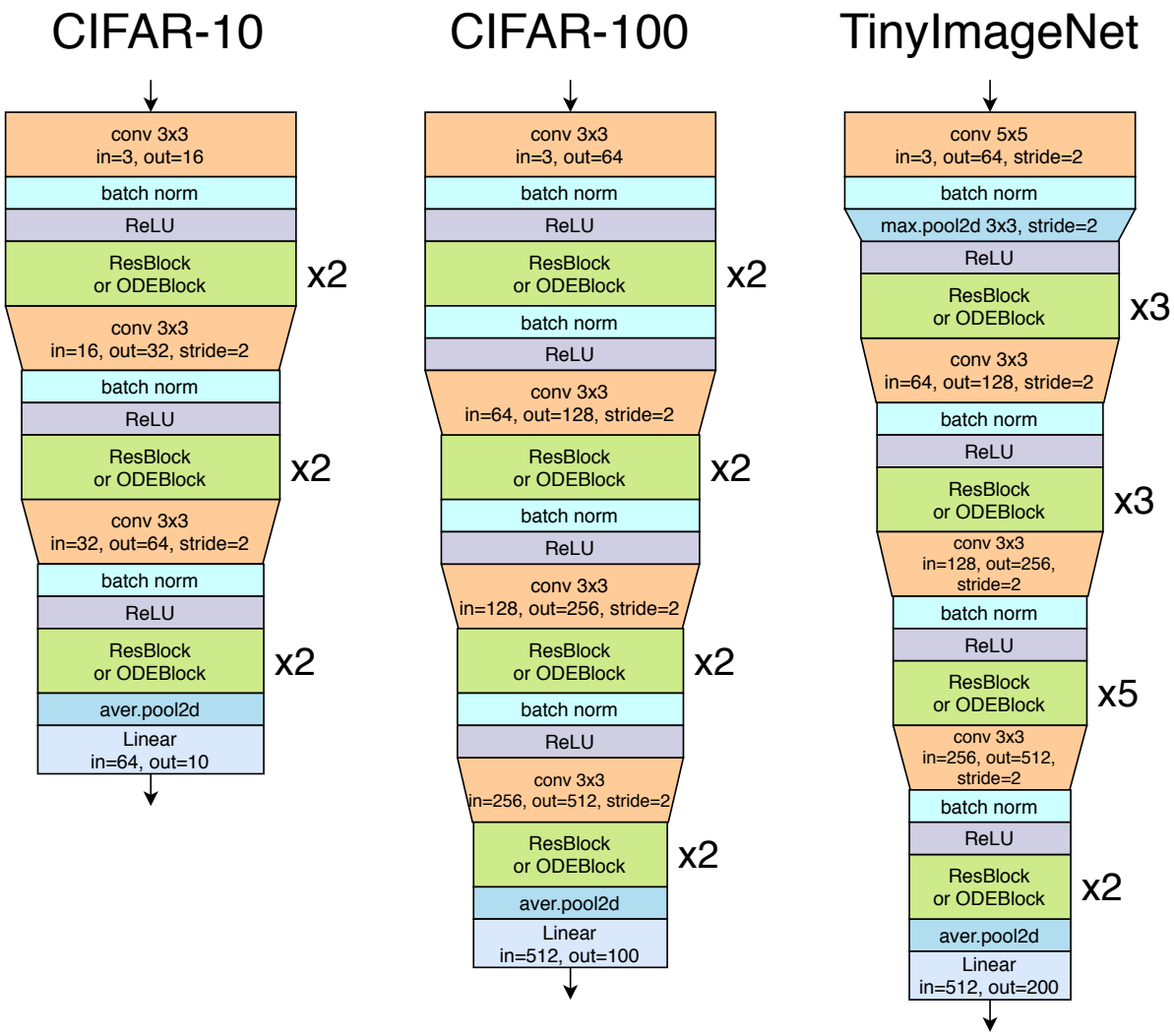


Figure 6: Architectures of models. $\times\ n$ near a block means that it is repeated n times

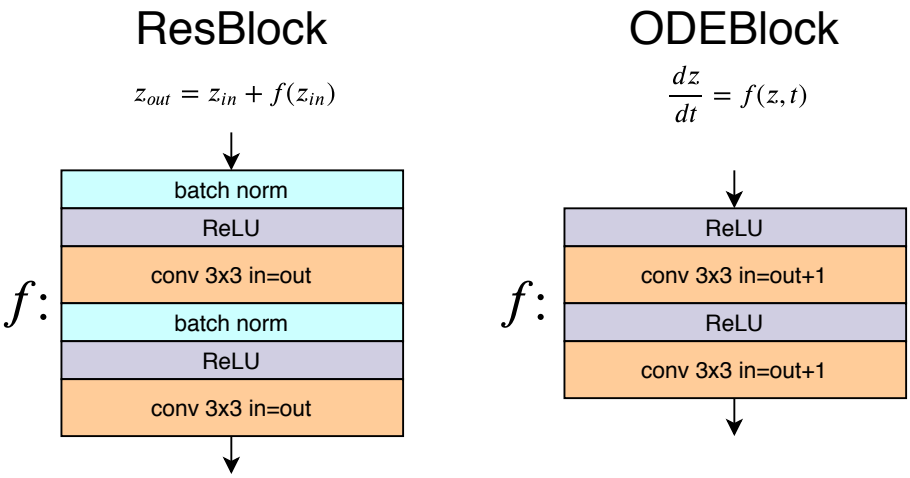


Figure 7: Integration blocks