Deep Neural Networks Motivated by Partial Differential Equations

Lars Ruthotto[1,3] and Eldad Haber[2,3]

Journal of Mathematical Imaging and Vision (2020) 62:352-364

Reversible Architectures for Arbitrarily Deep Residual Neural Networks

Bo Chang,[*1,2] **Lili Meng,**[*1,2] **Eldad Haber,**[1,2]
Lars Ruthotto,[2,3] **David Begert,**[2] **Elliot Holtham**[2]

[1]University of British Columbia, Vancouver, Canada. (bchang@stat.ubc.ca, menglili@cs.ubc.ca, haber@math.ubc.ca)
[2]Xtract Technologies Inc., Vancouver, Canada. (david@xtract.ai, elliot@xtract.ai)
[3]Emory University, Atlanta, USA. (lruthotto@emory.edu). This author is supported in part by NSF DMS 1522599.
[*]Authors contributed equally.

AAAI-18

RevNet

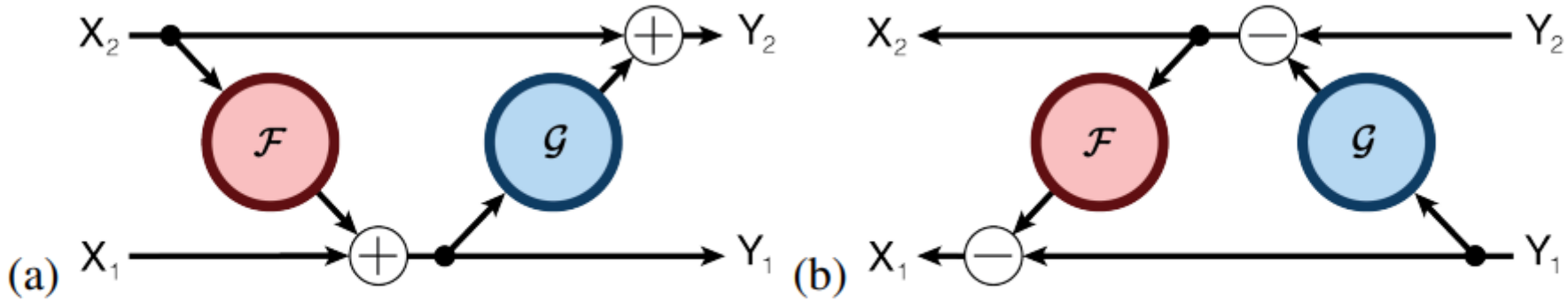


Figure 2: **(a)** the forward, and **(b)** the reverse computations of a residual block, as in Equation 8.

$$y_1 = x_1 + \mathcal{F}(x_2) \qquad x_2 = y_2 - \mathcal{G}(y_1)$$
$$y_2 = x_2 + \mathcal{G}(y_1) \qquad x_1 = y_1 - \mathcal{F}(x_2)$$

Hamiltonian Reversible Block

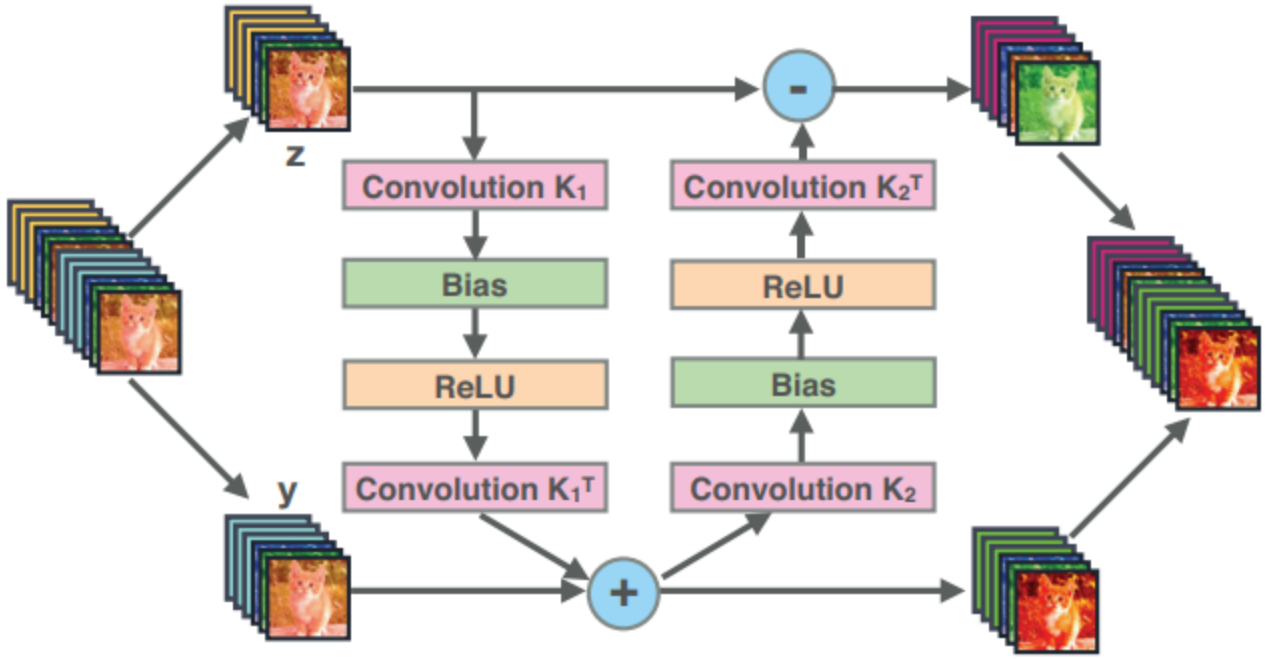


Figure 1: The Hamiltonian Reversible Block. First, the input feature map is equally channel-wise split to $\mathbf{Y}_j$ and $\mathbf{Z}_j$. Then the operations described in Eq. 10 are performed, resulting in $\mathbf{Y}_{j+1}$ and $\mathbf{Z}_{j+1}$. Finally, $\mathbf{Y}_{j+1}$ and $\mathbf{Z}_{j+1}$ are concatenated as the output of the block.

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathbf{K}_{j1}^T \sigma(\mathbf{K}_{j1}\mathbf{Z}_j + \mathbf{b}_{j1}),$$
$$\mathbf{Z}_{j+1} = \mathbf{Z}_j - h\mathbf{K}_{j2}^T \sigma(\mathbf{K}_{j2}\mathbf{Y}_{j+1} + \mathbf{b}_{j2}).$$

Hamiltonian Reversible Neural Network

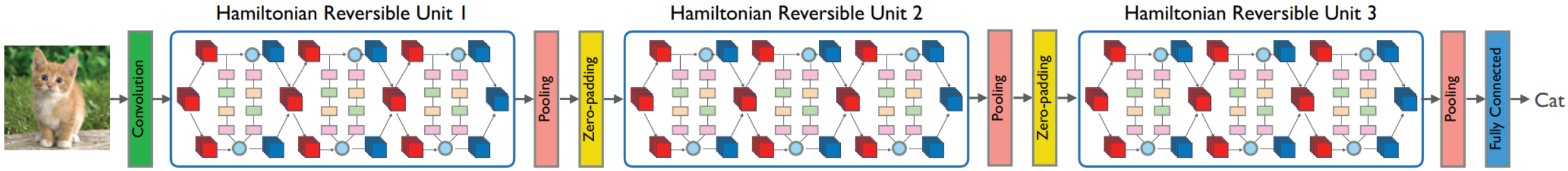


Figure 2: The Hamiltonian Reversible Neural Network. It is the simple stacking of several Hamiltonian reversible blocks as shown in Fig. 1 and pooling layer.

Finite difference

南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

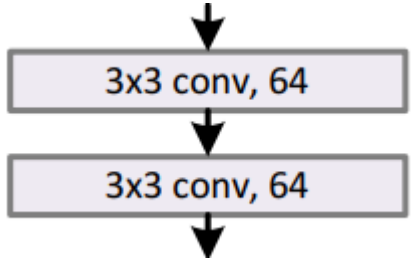
continuous $$u'(x) = \lim_{\Delta x \to 0} \frac{u(x+\Delta x) - u(x)}{\Delta x} \approx \frac{u(x+\Delta x) - u(x)}{\Delta x}$$

discrete $$u'_i = \frac{1}{h}(u_{i+1} - u_i), i = 0, 1, 2, ..., N-1$$

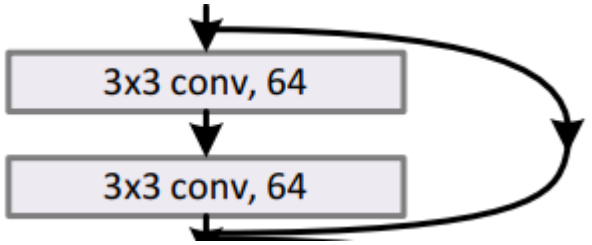
higher derivatives $$u''_i = \frac{1}{h^2}(u_{i+1} + u_{i-1} - 2u_i), i = 0, 1, 2, ..., N-1$$

Residual Networks and Differential Equations

$$\mathbf{F}(\boldsymbol{\theta}, \mathbf{Y}) = \mathbf{K}_2(\boldsymbol{\theta}^{(3)})\sigma\left(\mathcal{N}(\mathbf{K}_1(\boldsymbol{\theta}^{(1)})\mathbf{Y}, \boldsymbol{\theta}^{(2)})\right).$$



$$\begin{cases} \mathbf{Y}_0 = \mathbf{Y} \\ \mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathbf{F}(\boldsymbol{\theta}^{(j)}, \mathbf{Y}_j), \quad \text{for} \quad j = 0, 1, \ldots, N-1 \end{cases}$$



$$\min_{\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}} \frac{1}{2} S(\mathbf{W}\mathbf{Y}_N(\boldsymbol{\theta}) + (\mathbf{B}_W\boldsymbol{\mu})\mathbf{e}_s^\top, \mathbf{C}) + R(\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu})$$

Convolutional ResNets and PDEs

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathbf{F}(\boldsymbol{\theta}^{(j)}, \mathbf{Y}_j)$$

$$u_i' = \frac{1}{h}(u_{i+1} - u_i)$$

$$\partial_t \mathbf{Y}(\boldsymbol{\theta}, t) = \mathbf{F}(\boldsymbol{\theta}(t), \mathbf{Y}(t)), \text{ for } t \in (0, T]$$

$$\mathbf{Y}(\boldsymbol{\theta}, 0) = \mathbf{Y}_0.$$

$$\mathbf{F}(\boldsymbol{\theta}, \mathbf{Y}) = \mathbf{K}_2(\boldsymbol{\theta}^{(3)})\sigma\left(\mathcal{N}(\mathbf{K}_1(\boldsymbol{\theta}^{(1)})\mathbf{Y}, \boldsymbol{\theta}^{(2)})\right)$$

? PDE interpretation

Convolutional ResNets and PDEs

consider a one-dimensional convolution of a feature with one channel

$$\mathbf{y} = [y(x_1), \ldots, y(x_n)]^\top \quad \text{with} \quad x_i = \left(i - \frac{1}{2}\right) h.$$

$$\mathbf{K}_1(\boldsymbol{\theta})\mathbf{y} = [\boldsymbol{\theta}_1\, \boldsymbol{\theta}_2\, \boldsymbol{\theta}_3] * \mathbf{y}$$

$$= \left(\frac{\boldsymbol{\beta}_1}{4}[1\,2\,1] + \frac{\boldsymbol{\beta}_2}{2h}[-1\,0\,1] + \frac{\boldsymbol{\beta}_3}{h^2}[-1\,2\,-1] \right) * \mathbf{y}.$$

$$\begin{pmatrix} \frac{1}{4} & -\frac{1}{2h} & -\frac{1}{h^2} \\ \frac{1}{2} & 0 & \frac{2}{h^2} \\ \frac{1}{4} & \frac{1}{2h} & -\frac{1}{h^2} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \boldsymbol{\beta}_3 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \end{pmatrix}$$

$$\mathbf{K}_1(\boldsymbol{\theta}) = \boldsymbol{\beta}_1(\boldsymbol{\theta}) + \boldsymbol{\beta}_2(\boldsymbol{\theta})\partial_x + \boldsymbol{\beta}_3(\boldsymbol{\theta})\partial_x^2.$$

$$y_i = \frac{y_i}{1}$$

$$\partial_x y_i = \frac{y_{i+1} - y_i}{h}$$

$$\partial_x^2 y_i = \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2}$$

Convolutional ResNets and PDEs

$$\mathbf{K}_1(\boldsymbol{\theta})\mathbf{y} = [\boldsymbol{\theta}_1\, \boldsymbol{\theta}_2\, \boldsymbol{\theta}_3] * \mathbf{y} \qquad \mathbf{y} = [y(x_1), \ldots, y(x_n)]^\top \quad \text{with} \quad x_i = \left(i - \frac{1}{2}\right) h.$$

$$\begin{pmatrix} \theta_2 & \theta_3 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \theta_1 & \theta_2 & \theta_3 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \theta_1 & \theta_2 & \theta_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & & 0 & 0 & \cdots & \theta_1 & \theta_2 & \theta_3 \\ 0 & & 0 & 0 & \cdots & 0 & \theta_1 & \theta_2 \end{pmatrix} \mathbf{y}$$

$$\theta_1 x(\frac{h}{2}) + \theta_2 x(\frac{3h}{2}) + \theta_3 x(\frac{5h}{2}).$$

$$\theta_1 x(\frac{h}{2}) + \theta_2 x(\frac{3h}{2}) + \theta_3 x(\frac{5h}{2}) = \beta_1 x(\frac{3h}{2}) + \beta_2 \frac{x(\frac{5h}{2}) - x(\frac{h}{2})}{2h} + \beta_3 \frac{x(\frac{5h}{2}) + x(\frac{h}{2}) - 2x(\frac{3h}{2})}{h^2},$$

$$\boldsymbol{\beta}_1(\boldsymbol{\theta})\mathbf{y} + \boldsymbol{\beta}_2(\boldsymbol{\theta})\partial_x\mathbf{y} + \boldsymbol{\beta}_3(\boldsymbol{\theta})\partial_x^2\mathbf{y}$$

Convolutional ResNets and PDEs

$$\begin{aligned}\mathbf{K}_1(\boldsymbol{\theta}) = \boldsymbol{\beta}_1(\boldsymbol{\theta}) + \boldsymbol{\beta}_2(\boldsymbol{\theta})\partial_x + \boldsymbol{\beta}_3(\boldsymbol{\theta})\partial_y \\ + \boldsymbol{\beta}_4(\boldsymbol{\theta})\partial_x^2 + \boldsymbol{\beta}_5(\boldsymbol{\theta})\partial_y^2 + \boldsymbol{\beta}_6(\boldsymbol{\theta})\partial_x\partial_y \\ + \boldsymbol{\beta}_7(\boldsymbol{\theta})\partial_x^2\partial_y + \boldsymbol{\beta}_8(\boldsymbol{\theta})\partial_x\partial_y^2 + \boldsymbol{\beta}_9(\boldsymbol{\theta})\partial_x^2\partial_y^2.\end{aligned}$$

proceed the same way with K2,
when the number of input and output channels is larger than one, K1 andK2 lead to a system of coupled partial differential operators.

$$\mathbf{F}(\boldsymbol{\theta}, \mathbf{Y}) = \mathbf{K}_2(\boldsymbol{\theta}^{(3)})\sigma\left(\mathcal{N}(\mathbf{K}_1(\boldsymbol{\theta}^{(1)})\mathbf{Y}, \boldsymbol{\theta}^{(2)})\right)$$

---

stability $\|\mathbf{Y}(\boldsymbol{\theta}, T) - \tilde{\mathbf{Y}}(\boldsymbol{\theta}, T)\|_F \leq M\|\mathbf{Y}_0 - \tilde{\mathbf{Y}}_0\|_F$

$\partial_t \mathbf{Y}(\boldsymbol{\theta}, t) = \mathbf{F}(\boldsymbol{\theta}(t), \mathbf{Y}(t)),\ \text{for } t \in (0, T]$
$\mathbf{Y}(\boldsymbol{\theta}, 0) = \mathbf{Y}_0.$

$$\mathbf{J}_\mathbf{Y}\mathbf{F} = \mathbf{K}_2(\boldsymbol{\theta})\ \mathrm{diag}(\sigma'(\mathbf{K}_1(\boldsymbol{\theta})\mathbf{Y}))\ \mathbf{K}_1(\boldsymbol{\theta})$$ assume N (Y) = Y

Deep Neural Networks Motivated by PDEs

to obtain a stable network, choosing K2 = −K1T

$$\mathbf{F}_{\text{sym}}(\boldsymbol{\theta}, \mathbf{Y}) = -\mathbf{K}(\boldsymbol{\theta})^{\top}\sigma\left(\mathcal{N}(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y}, \boldsymbol{\theta})\right)$$

1. Parabolic CNN

$$\partial_t \mathbf{Y}(\boldsymbol{\theta}, t) = \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t), \mathbf{Y}(t)), \quad \text{for } t \in (0, T].$$

$$= -\nabla^2 Y$$

if σ (x) = x, N (Y) = Y and K(t) = ∇, (heat equation)

new normalization layer motivated by total variation(TV) denoising

$$\mathcal{N}_{\text{tv}}(\mathbf{y}) = \text{diag}\left(\frac{1}{\mathbf{A}^{\top}\sqrt{(\mathbf{A}\mathbf{y})^2 + \epsilon}}\right)\mathbf{y},$$

operator A ∈ R n/c ×n computes the sum over all c channels for each pixel

Deep Neural Networks Motivated by PDEs

2. Hyperbolic CNNs (Reversibility)

$$u_i'' = \frac{1}{h^2}(u_{i+1} + u_{i-1} - 2u_i), i = 0, 1, 2, ..., N-1$$

2.1 Hamiltonian CNNs

$$\begin{aligned} \partial_t \mathbf{Y}(t) &= \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t), \mathbf{Z}(t)), \qquad \mathbf{Y}(0) = \mathbf{Y}_0 \\ \partial_t \mathbf{Z}(t) &= -\mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t), \mathbf{Y}(t)), \qquad \mathbf{Z}(0) = \mathbf{Z}_0. \end{aligned}$$

$$\begin{aligned} \mathbf{Y}_{j+1} &= \mathbf{Y}_j + \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t_j), \mathbf{Z}_j), \qquad & \mathbf{Z}_j &= \mathbf{Z}_{j+1} + \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t_j), \mathbf{Y}_{j+1}) \\ \mathbf{Z}_{j+1} &= \mathbf{Z}_j - \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t_j), \mathbf{Y}_{j+1}), & \mathbf{Y}_j &= \mathbf{Y}_{j+1} - \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t_j), \mathbf{Z}_j), \end{aligned}$$

2.2 Second-order CNNs

$$\begin{aligned} \partial_t^2 \mathbf{Y}(t) &= \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t), \mathbf{Y}(t)), \\ \mathbf{Y}(0) &= \mathbf{Y}_0, \quad \partial_t \mathbf{Y}(0) = 0. \end{aligned}$$

$$\begin{cases} \mathbf{Y}_{-1} = \mathbf{Y}_0 \\ \mathbf{Y}_{j+1} = 2\mathbf{Y}_j - \mathbf{Y}_{j-1} + \delta_t^2 \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t_j), \mathbf{Y}_j). \end{cases}$$

Architecture

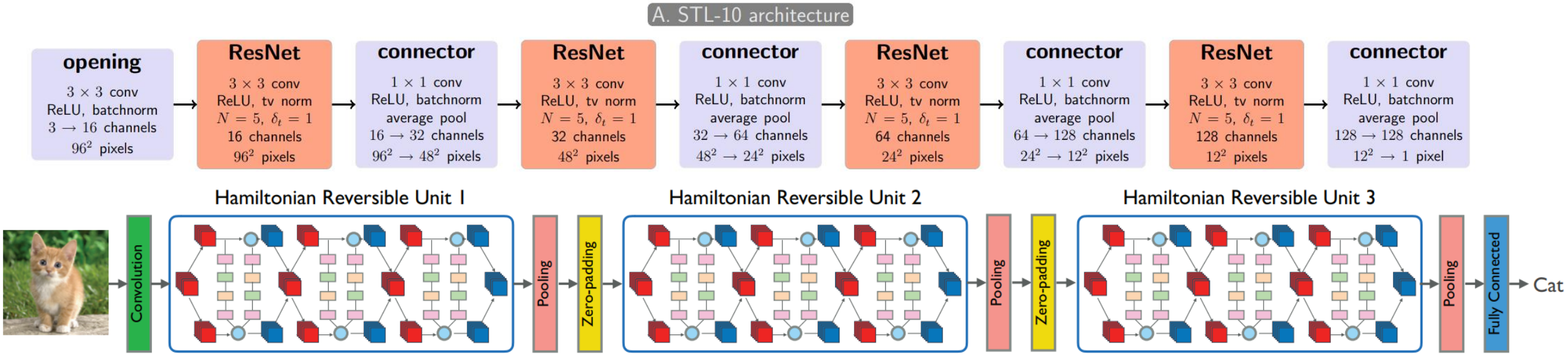


Figure 2: The Hamiltonian Reversible Neural Network. It is the simple stacking of several Hamiltonian reversible blocks as shown in Fig. 1 and pooling layer.

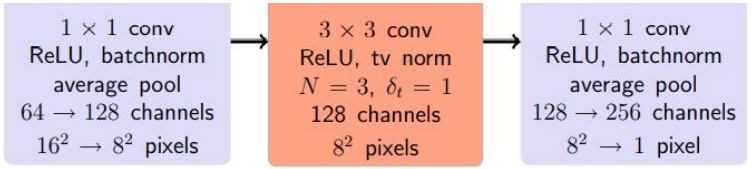


Fig. 2 Overview of network architectures for the STL-10 (top row) and CIFAR-10/100 (bottom row) image classification problems. The architectures consist of ResNet blocks (red) that represent the parabolic, Hamiltonian, and second-order dynamics, respectively. The networks also contain an opening layer and several connector layers (blue) that increase the number of channels and reduce the image resolution (Color figure online)

Experiments

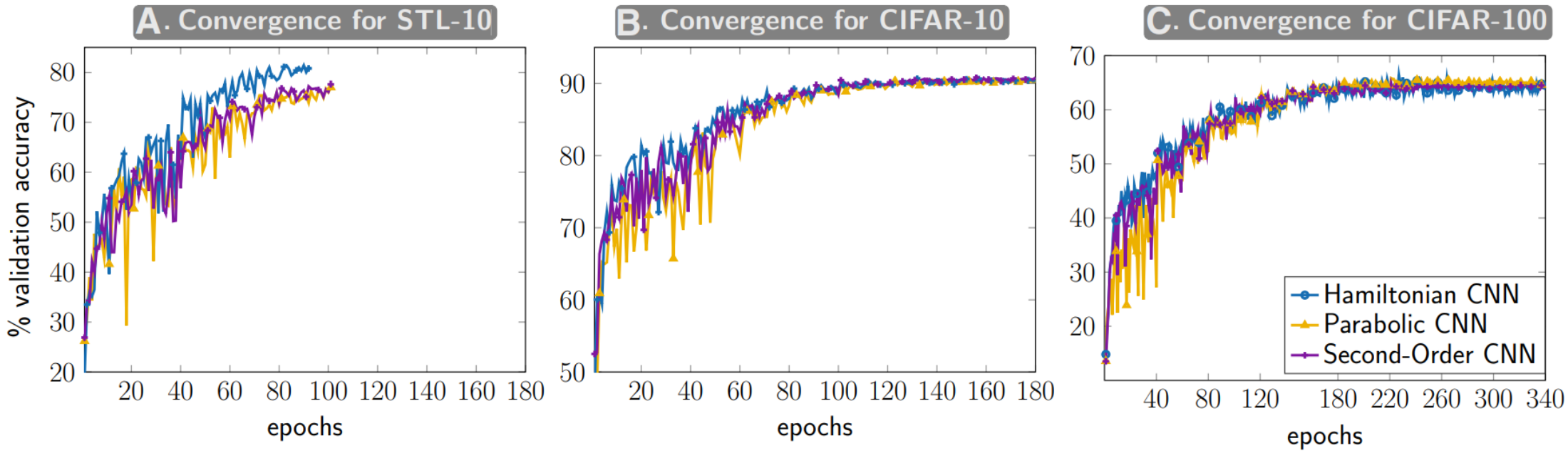


Fig. 3 Performance of the training algorithm for the three proposed architectures applied to the STL-10 (left), CIFAR-10 (middle) and CIFAR-100 (right) datasets. We use randomly choose 80% of the training images to update the weights using SGD. The plots show the validation accuracy computed using the remaining images after every epoch. In these examples, we did not observe considerable overfitting and note that the weights from the final epoch led to adequate validation accuracies (Color figure online)

Experiments

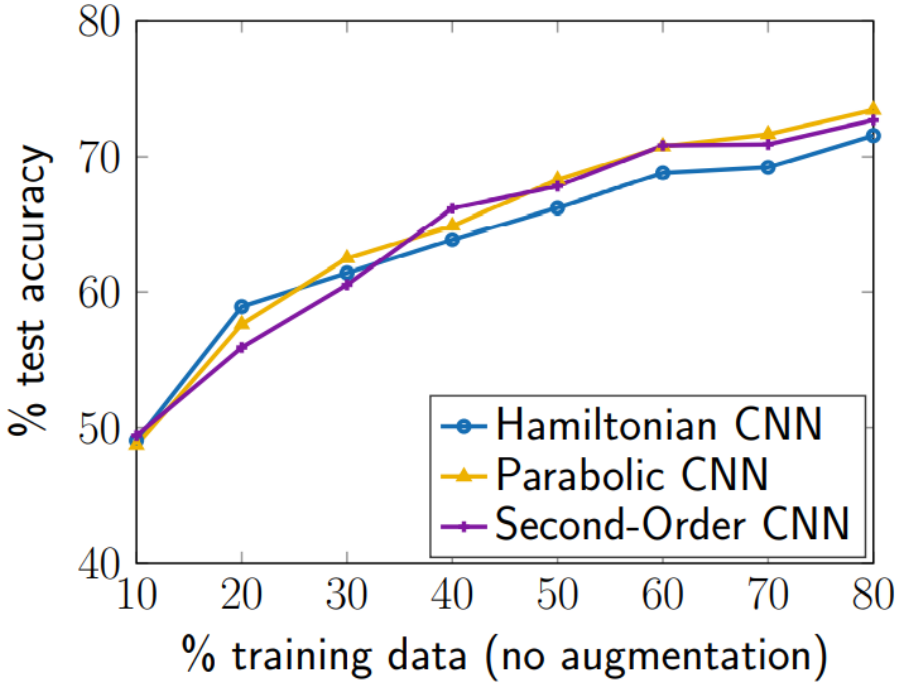


Fig. 4 Improvement of the test accuracy when increasing the number of training images in the STL-10 dataset (from 10% to 80% in increments of 10%). We first train the weights of the three proposed architectures and plot the test accuracy of the final iterate. Here, we do not use any data augmentation. Expectedly, the generalization improves as more images are used for all architectures (Color figure online)

Experiments

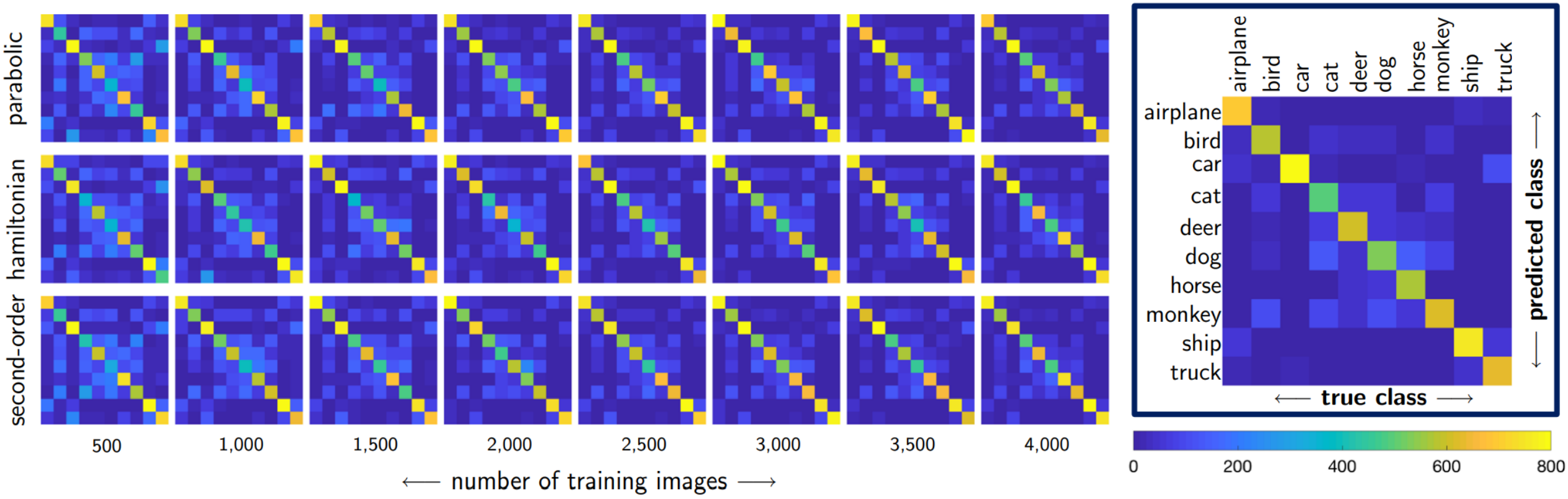


Fig. 5 Confusion matrices for classifiers obtained using the three proposed architectures (row-wise) for an increasing number of training data from the STL-10 dataset (column-wise). The (i, j)th element of the 10×10 confusion matrix counts the number of images of class i for which the predicted class is j. We use the entire test data set, which contains 800 images per class (Color figure online)

Experiments

Table 1 Summary of numerical results for the STL-10, CIFAR-10, and CIFAR-100 datasets

	STL-10		CIFAR-10		CIFAR-100	
	Number of weights (M)	Test data (8000) accuracy %(loss)	Number of weights (M)	Test data (10,000) accuracy %(loss)	Number of weights (M)	Test data (10,000) accuracy %(loss)
Parabolic	1.01	80.9% (0.726)	0.50	90.5% (0.316)	0.65	67.4% (1.185)
Hamiltonian	0.52	80.4% (0.770)	0.26	90.7% (0.334)	0.36	67.1% (1.208)
Second-order	1.01	79.6% (0.770)	0.50	90.6% (0.329)	0.65	66.9% (1.281)
Hamiltonian [9]	1.28	85.5% (n/a)	0.43	92.8% (n/a)	0.44	71.0% (n/a)
Leapfrog [9]	2.44	84.6% (n/a)	0.50	91.9% (n/a)	0.51	69.1% (n/a)
ResNet-110 [25]			1.7	93.4% (n/a)	1.7	74.8% (n/a)
Wide ResNet [51]			0.6	93.2% (n/a)	0.6	69.1% (n/a)

Using the hyperparameters chosen by cross-validation, we train the networks on the entire training data. After training, we compute and report the classification accuracy and the value of cross-entropy loss (in brackets where reported) for the test data. To this end, we use the weights from the final epoch of SGD. We also report the number of trainable weights for each network and for comparison state results from the literature achieved with similarly sized or larger architectures

Thanks