Condensing CNNs with Partial Differential Equations

Anil Kag, Venkatesh Saligrama
Department of Electrical and Computer Engineering, Boston University
{anilkag, srv}@bu.edu

Abstract

Convolutional neural networks (CNNs) rely on the depth of the architecture to obtain complex features. It results in computationally expensive models for low-resource IoT devices. Convolutional operators are local and restricted in the receptive field, which increases with depth. We explore partial differential equations (PDEs) that offer a global receptive field without the added overhead of maintaining large kernel convolutional filters. We propose a new feature layer, called the Global layer, that enforces PDE constraints on the feature maps, resulting in rich features. These constraints are solved by embedding iterative schemes in the network. The proposed layer can be embedded in any deep CNN to transform it into a shallower network. Thus, resulting in compact and computationally efficient architectures achieving similar performance as the original network. Our experimental evaluation demonstrates that architectures with global layers require $2-5\times$ less computational and storage budget without any significant loss in performance.

1. Introduction

Convolutional neural networks (CNNs) have been the backbone for recent advances in image recognition [19], object detection [33], and other applications [31] interfacing the image modalities. Convolutional filters with limited receptive fields act on localized input regions to generate low-level features. Features used for decision-making are complex functions of these low-level features, achieved through the composition of many such convolutional operators applied in sequence, resulting in deep networks with high inference/train time and large model size.

Recent works [3, 5] have explored neural networks inspired by ordinary differential equations (ODEs), offering richer representation than their discrete counterparts. Resnets [8] can be viewed as a discretized form of ODEs. The final architecture based on these continuous layers leads to higher computational cost in comparison to their discrete counterpart [2], namely due to the costly fixed point solvers. In contrast, we explore novel constraints on the feature maps, based on partial differential equations (PDEs) that offer similar rich representation but with shallower neural networks. In addition, we provide efficient and scalable solvers to provide computational and storage savings.

Proposed Method. We explore a hybrid approach wherein we modify discrete models by embedding a new layer with a global receptive field that operates on the input feature map and computes complex compositions of these low-level features. We call this layer the Global feature layer. It approximately solves a PDE constraint that couples the input and output feature maps. In a typical discrete model, at every input resolution, the same convolutional block is applied repeatedly m times. We modify this structure by keeping only one convolutional block and replacing the $m-1$ blocks with a single global feature layer (see Figure 1). Thus, reducing the deep neural network to a much shallower network without any significant performance loss. It leads to smaller models with low computational and storage costs. In addition, it improves both the train and inference times.

By keeping at least one block from the original architecture, we are incorporating the signature of this architecture. It allows the application of this generic global feature layer to any architecture. Also, since a good start for any iterative solver implies smaller steps to reach the solution, this original block helps to initialize the PDE solution.

Estimated Savings. Suppose a Resnet architecture constructed with three resolutions has m residual blocks, and for the three resolutions, the compute cost is $= \{c_1, c_2, c_3\}$ respectively. The total compute cost for operating this network is $m \times (c_1 + c_2 + c_3)$. Global residual block replaces $m-1$ residual blocks with just one global block and assuming that the cost of this global block is similar, i.e. $= \{c_1, c_2, c_3\}$, then the cost to operate the modified network is $2 \times (c_1 + c_2 + c_3)$. Given that $m > 2$, the modified network can lead to computational savings over Resnet. Similar conclusions can be drawn for storage savings.

Motivational Example. To motivate our approach, we apply the Global feature layer to the Resnet32 [8] architecture, where at each feature map resolution, the same block repeats 5 times. With the experimental setup described in the section 4, we train three models on the CIFAR-10 dataset: (a)

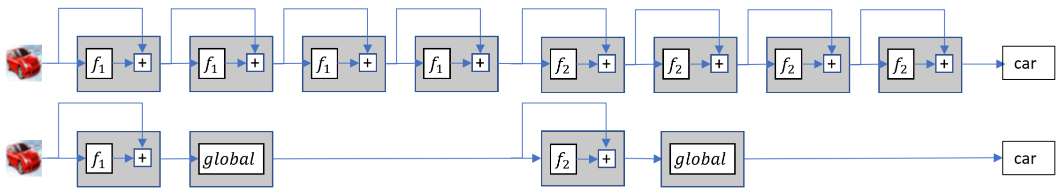


Figure 1. Replacing repeated blocks in a given CNN architecture with the Global layer for compute and model savings.

Resnet32 : same architecture as used in [8], (b) ODE based Resnet32, i.e., MDEQ [2] modified to match feature map configuration as in Resnet32, and (c) ResNet32-Global : replaced repeated blocks with global layers. Table 1 shows that both Resnet32 and MDEQ have similar performance. Note that MDEQ is significantly costly in terms of the floating-point multiply-add operations (MACs). In contrast, the proposed Resnet32-Global results in a much smaller model and a significantly lower computational footprint without any hit in the performance. This experiment clearly shows that the Global layer results in the following benefits:

1. **Shallow network.** Resnet32-Global has $\approx 3\times$ less depth.
2. **Less storage.** Resnet32-Global has $\approx 3\times$ less parameters.
3. **Less compute.** Resnet32-Global uses $\approx 5\times$ less MACs.
4. **Readily embedable in any network.**

Table 1. CIFAR-10 : Comparison between discrete Resnet32, ODE based Resnet32 (MDEQ [2]), and our PDE embedded Resnet32-Global. We compute the depth as the number of blocks in the network. Train and Inference time denote the cost of processing one pass of the train and test dataset on a V100 GPU. Supplementary Table 16 lists results for Resnet ($m = 2$) and CIFAR-100 dataset.

	Accuracy	**#Params**	**#MACs**	**Train Time(s)**	**Inference Time(s)**	**Depth**
Resnet32	92.49%	460K	70M	78	4.45	15
MDEQ	92.28%	1.1M	1.5B	409	23.32	-
Resnet32-Global	91.93%	162K	15M	24	1.91	6

Contributions.

- Proposed a Global feature layer that imposes PDE constraints on the input and output feature maps. Embedding this layer in deep networks results in their shallower variants with a smaller footprint with similar performance.
- Embedded the proposed global layer in many existing CNN architectures and conducted an extensive empirical study on benchmark image recognition datasets to show computational and storage savings.
- Proposed an efficient and approximate PDE solver to embed in the neural network wherein model accuracy can be traded-off for the computational budget.
- We provide pseudo-code for the Global layer that is readily deployable in any popular deep learning library. Our PyTorch implementation is available at https://github.com/anilkagak2/PDE_GlobalLayer.

2. Related Work

There is a vast literature related to models for object recognition, including low complexity models. Here we only include papers closely related to our approach.

Early Skip CNNs. Resnet [8], Highway networks [27], Wide-Resnets [32], Dense-Nets [12], etc. proposed networks with skip/residual connections. These changes helped alleviate the vanishing gradient issues in the deep neural networks. These trained much deeper models and hence achieved significantly better performance than the previous generation models like AlexNet [19], VGG-Nets [26], etc. Note that deeper models implicitly mean larger storage costs and higher compute requirements.

Mobile/IoT ready CNNs. Many initial attempts (SqueezeNet [14], SqueezeNext [6], MobileNets [11]) at designing low complexity models included handcrafted feature blocks (with low rank filters, separable convolutions, etc.) whose composition yielded small models with low floating-point operations. Recently, EfficientNets [29] were proposed to systematically study the effect of width, depth, channels, etc., along with memory and MACs constraints. There have also been efforts [21, 34] to search for neural architectures that outperform hand-crafted architectures. Note that these are complementary to our proposal.

Model Compression/Distillation. An alternate strategy to obtain small models require model compression. Deep-compression [7] is an early work where a pre-trained network is pruned, quantized, and compressed to yield small networks which can be deployed on the edge devices. Other works include distilling [9] knowledge from a larger pre-trained network into a small compact model. We do not pursue these techniques to simplify our exposition.

ODE Inspired CNNs. Related work in this class has the most similarity with our approach. Neural ODEs (NODEs) [3] introduce continuous time layers following an ODE. It uses black-box ODE solvers along with the adjoint method for back-propagation. Augmented Neural ODEs [4] extend NODEs to a richer class of functions while ANODE [5] addresses the gradient computation in the adjoint method to allow for more accurate gradients matching the discretization. Neural ODEs and their variants do not demonstrate their scalability to tasks like Imagenet.

There have been previous works that utilize ODE-inspired models for sequential processing. Some of these models require the architecture to achieve equilibrium [15, 17], [1], where the later has been extended to image modalities by Multi-scale deep equilibrium models (MDEQs) [2]. MDEQs use implicit layers at multiple feature scales to scale to large datasets such as Imagenet. Although they show good performance on large-scale tasks, the model capacity still needs to be nearly the same as the discrete counterparts. Although implicit models offer low memory cost training, they still do not offer much flexibility for inference. In addition, their inference cost is much higher in comparison to discrete variants such as Resnet.

PDE Inspired CNNs. [24] proposed new architectures based on parabolic and hyperbolic partial differential equations. These connections with PDEs enable theoretical reasoning, such as the stability of the resulting network. Although the resulting models are small, these models take a significant hit in performance. NeuPDE [28] uses the convolutional filters to approximate the differential operators for generic second order PDE. NeuPDEs downsample the input image to a manageable feature map through many convolutional layers and then finally applies the PDE blocks. This construction helps in reducing the model size but the gains are not sufficient enough.

3. Method

In this section, we will formalize the PDE constraint on the feature representation. We will describe the proposed Global layer, including our PDE choice, and embed an approximate numerical solver in the neural network. Finally, we will provide building blocks and pseudo-code to improve the understanding of our architecture.

Notation. For simplicity, we will assume that the output shape is the same as the input, and we are dealing with only a $2D$ feature map represented by the $X - Y$ plane. Let $I(x, y) \in \mathbb{R}^{h\times w}$ denote the input feature map with $h \times w$ entries. Let $H(x, y) \in \mathbb{R}^{h\times w}$ be the output feature map. We will denote Δ_{xy} as the differential operator (contains partial differential operators for various interactions between the two dimensions in the input).

3.1. PDE Constrained Features.

We enforce the following PDE constraint on the output feature map H

$$\Delta_{xy} H(x, y) = f(I(x, y)) \tag{1}$$

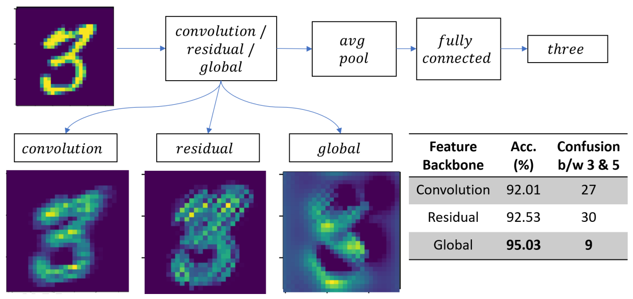
where f is a function applied on the input feature map. The above operator applies globally on the feature map and does not restrict itself to the local receptive field of operators such as one-layer convolutions.

Illustration. Before delving into further details about the global feature layer (namely the exact PDE and the numerical solver), we provide an intuitive example on the MNIST dataset to demonstrate the effectiveness of such a strategy. We construct a network with one feature layer followed by average pooling operation and classifier layer (see Figure 2). Note that the feature map layer constructs only one feature map. It gives rise to three networks by using different feature layers : (a) CNN-Net: convolutional layer as the feature map, (b) Residual-Net: a residual connection between the convolutional layers, and (c) PDE-Net: PDE constrained layer as the feature map. Note that all three networks have 524 parameters to ensure a fair comparison. Thus the only difference between these networks remains in the way features are processed. We train these networks on the MNIST dataset with the same settings (optimizer, learning rate, epochs, no data-augmentations, see Sec. 4) to provide a fair evaluation.

On the held-out test set, CNN-Net achieves 92.01% accuracy, Residual-Net achieves 92.53% accuracy, while PDE-Net achieves 95.03% accuracy. Since the network architecture apart from the feature layer is the same, we can analyze the feature map easily to see the contrast between the two feature representations. Figure 2 shows the intermediate representation from these neural networks. It shows that the feature maps generated by PDE-Net effectively highlight the input object by smoothing the noisy background and increasing brightness around the object edges.

3.2. Global Feature Layer

To embed a PDE in the neural network layer, we need to describe four components of the PDE: (a) its exact form, i.e. Δ_{xy}, (b) a numerical solver, (c) initial guess of the solution, and finally (d) choice of free parameters such as the function



Feature Backbone	Acc. (%)	Confusion b/w 3 & 5
Convolution	92.01	27
Residual	92.53	30
Global	**95.03**	**9**

Figure 2. Toy Example comparing different backbones: Convolutional, Residual, and Global. We show network representation for the input image for the letter three. Intermediate features from Convolutional and Residual backbones do not show bright intensity around the edges and have an uneven background. In contrast, the Global layer smoothens it out and shows bright spots around the digit. Thus, the Global layer provides a better and markedly different representation than the other two backbones. All three networks have 524 parameters. Network with Global layer achieves 95% accuracy while the other two achieves $\approx 92.5\%$ accuracy. It also has a significantly lesser confusion between the letters 3 and 5. See other visualizations in supplementary Sec. A.9.

f. We refer to this new layer as the Global feature layer and describe these components below.

(a) PDE: At the heart of the Global feature layer is the following generic advection-diffusion PDE [1]

$$\frac{\partial}{\partial t}H = \nabla \cdot (D\nabla H) + \nabla \cdot (\mathbf{v}H) + f(I) \tag{2}$$

It lets us treat the input feature map pixels as particles in motion with velocity $\mathbf{v}$ that interact with their neighborhood through diffusion coefficient D. Starting at time $t = 0$ with initial guess of the concentration $H(t = 0)$, the solution of this advection-diffusion equation provides the final particle concentration $H(t = T)$ at time T. It is the output representation of the global feature layer. The motion of the particles affects the concentration and is modelled by the *advection* term $\nabla \cdot (\mathbf{v}H)$. Similarly, the term $\nabla \cdot (D\nabla H)$ describes the *diffusion* phenomenon, where particles shift between low and high concentrations to reach a steady state. Note that both D and $\mathbf{v}$ can be a function of the particle locations. Finally, the term $f(I)$ is the source of the particle concentration.

In our 2D world, the velocity and diffusion coefficients have two components, i.e. $\mathbf{v} = (u, v)$ and $D = (D_x, D_y)$, and the Eq. 2 boils down to the following form [13]

$$\frac{\partial}{\partial t}H(x,y,t)+\frac{\partial}{\partial x}\big(u(x,y,t)H(x,y,t)\big)+\frac{\partial}{\partial y}\big(v(x,y,t)H(x,y,t)\big)$$
$$=\frac{\partial}{\partial x}\Big(D_x\frac{\partial}{\partial x}H(x,y,t)\Big)+\frac{\partial}{\partial y}\Big(D_y\frac{\partial}{\partial y}H(x,y,t)\Big)+f(I(x,y)) \tag{3}$$

(b) Iterative Solver: For an efficient implementation of the global layer, we need a simple and efficient PDE solver that can be embedded in the neural network and can achieve approximate solutions easily. To obtain a finite element scheme, it is standard in the literature to expand the partial differential operators with their finite-difference elements. Assume the discrete steps for x, y and t by δ_x, δ_y and δ_t respectively. Following [13], we discretize the Eq. 3 as (see Supplementary Sec. A.2 for detailed derivation),

$$\begin{aligned}LH^{k+1}_{x,y} &= MH^{k-1}_{x,y} - 2(u_x + v_y)\delta_t H^k_{x,y} + 2\delta_t f(I(x,y))\\ &+ (-A_x + 2B_x)H^k_{x+1,y} + (A_x + 2B_x)H^k_{x-1,y}\\ &+ (-A_y + 2B_y)H^k_{x,y+1} + (A_y + 2B_y)H^k_{x,y-1}\end{aligned} \tag{4}$$

where $L = (1 + 2B_x + 2B_y)$, and $M = (1 - 2B_x - 2B_y)$

$$u_x = \frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x}; v_y = \frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y};$$

$$A_x = \frac{u\delta_t}{\delta_x}; A_y = \frac{v\delta_t}{\delta_y}; B_x = \frac{D_x\delta_t}{\delta_x^2}; B_y = \frac{D_y\delta_t}{\delta_y^2};$$

[1]https://en.wikipedia.org/wiki/Convection-diffusion_equation

Given a suitable initialization of the output feature map H at $t = 0$, Eq. 4 provides an update rule to find the PDE solution at any time $t = T$. We need to initialize the algorithm and we can take K steps of this iteration on the entire 2D map to get the solution at time $T = K\delta_t$.

(c) Initialization: An initial guess of the solution is crucial for the convergence of the previous recursion. A better initial guess leads to faster convergence. Multiple strategies exist to initialize the output feature map, namely (a) input feature map I, (b) fixed function of the input, and (c) a learnable function of the input. We follow the last option. Given an architecture, we use one of its building blocks as the initialization point and learn its parameters during the training stage with back-propagation. Thus, for architecture such as Resnet, at any resolution level, we use the first block as the output of the global feature layer at $t = 0$. We run the PDE for K steps to get the final feature map at time $\frac{K}{\delta_t}$.

(d) Choice of the free parameters: There are some free parameters in the Eq. 3. To complete the description of the Global feature layer, we list our parameterization for these free parameters, namely (a) function f (b) particle velocity (u, v), and (c) diffusion coefficient (D_x, D_y). For simplicity, we keep f as identity operator on the input and learn other parameters as the depth-wise convolution over the initialization. We point out that, for compute savings, one can further fix these parameters by keeping identity operations or treating these as hyper-parameters. We study the impact of different choices for free parameters in our ablations (see Sec. 4.6 & Supplementary Sec. A.6). We leave the design choice improvements (ex. employing architectural search for better combinations) for future work.

Finally, as a passing remark, we point out that we have not handled the boundary conditions explicitly in our formulation. Ideally, one should carefully design the behavior of the PDE at the boundary. Instead, we roll the image such that the first particle is a neighbor of the last. Since our goal is only to find an approximation, this modification suffices.

Note that our choices for the PDE and the numerical solver are motivated by the ease of implementation and simplicity in exposition. We leave the exploration of various other PDEs (Laplace equations, Heat equations, Navier-Stokes etc.) and better solvers to future work.

Implementation. Algorithm 1 shows the pseudo-code for the Global layer. This feature layer integrates easily in any architecture with appropriate initializations. By default, we take the discrete step sizes to be $\delta_t = 0.2$, and run the recursions till $K = 5$ steps, resulting in the output state at $T = K\delta_t = 1$. We take $\delta_x = \delta_y = 1$ as the pixel values are not available at any finer details. For all our experiments, free parameters in Eq. 3 are depthwise convolutional operators with the same kernel size as the original block. For CIFAR-10 low-budget experiments, we use constant diffusion coefficients and set 1 as their default values.

Algorithm 1 Pseudo Code for the Global Feature Block

```
Input : Input feature map $I \in \mathbb{R}^{h \times w}$
Input : Initial solution guess $F(I)$, Function $f$
Output : Output feature map $O$
Init : $H^{-1} = H^0 = F(I)$
Compute velocity $(u, v)$, diffusion coefficient $(D_x, D_y)$
for $k = 1$ to $K$ do
    Compute $f(H^{k-1}, I)$
    for $x = 1, y = 1$ to $h, w$ do
        Set $H^{k+1}[x, y]$ as per Eq. 4
    end for
    Set output feature map $O = H^K$
end for
```

Differences from existing PDE/ODE CNNs. Existing ODE-based CNNs [3] have focused on showing an equivalence between Resnet like architectures and the continuous-time ODEs. Although such connections provide new insights, few efforts utilize such ideas for compact CNNs due to expensive fixed point solvers and costly residual functions. Thus, these architectures are unable to scale-up to large datasets such as Imagenet (see Supplementary Sec. A.8 for further details).

Existing PDE-based CNNs [24, 28] apply generic stencil operators and do not solve any specific PDE. Furthermore, most of the works use such operators on the heavily downsampled initial feature map. In contrast, we apply the proposed Global layer at every resolution level and remove the dependence on the repetition of the architectural blocks.

We also point out that our update Eq. 4 is not a simple residual connection. It is a discretization corresponding to the PDE Eq. 3, wherein different elements interact in time and spatial dimensions. In contrast, a recursive update of any generic convolution would not necessarily correspond to an iterative scheme and will not converge. In addition, we do not have expensive non-linearities in the update equation that slow down the recursion. Typically in ODE/PDE CNNs, the function f is a residual block with multiple full convolutions and non-linearities like batch-norm and activation functions.

Architectures with Global layer. Fig. 3 shows an schematic of the proposed Global layer. As discussed earlier, the free parameters in this layer are constructed as depthwise convolutions. This layer can be embedded in any existing architectures as seen in Sec. 4.

4. Experiments

In this section, we will apply the proposed Global layer in popular architectures. We will show that the resulting architectures have a much smaller computational and storage footprint than the original models. We will evaluate these models on various benchmark image recognition datasets.

4.1. Datasets

We use popular image classification datasets to show that our architectures provide benefits across many tasks. These datasets are publicly available. Our models are trained from scratch using the available training data. We report evaluation metrics on the publicly available test set.

1. **MNIST-10** [20] : This dataset consists of 10 classes with grayscale images of 28×28 pixels. There are $60,000$ images in the training set and $10,000$ images in the test set. We normalize the data to be mean 0 and variance 1.
2. **CIFAR-10/100** [18] : This dataset consists of RGB images of 32×32 pixels. It contains $50,000$ training and $10,000$ test images. It has two variants: (a) CIFAR-10 images are drawn from 10 classes, and (b) CIFAR-100 images are drawn from 100 classes. Unless explicitly stated, we follow standard data augmentation techniques (mirroring/shifting) used in earlier works [8, 12], followed by normalization to a standard gaussian across channels.
3. **Imagenet-1000** [23] : It is the popular ILSVRC 2012 classification dataset. This 1000 way classification dataset consists of 1.28 million training and $50,000$ validation images. We follow the standard data augmentation (mirroring, resize and crop to shape 224×224) for training and single crop for testing. Similar to previous works, we report results on the validation set.

4.2. Experimental Setup

We implement the Global feature layer in PyTorch using the Algorithm 1. Our experiments include strong baselines such as Resnet [8], Densenet [12], Wide Resnet [32], DARTS [22]. We embed global feature layers in these architectures and remove feature block repetitions resulting in Resnet-Global, Densenet-Global, Wide Resnet-Global, and DARTS-Global. For the Imagenet experiments, we perform similar adjustments to the state-of-the-art architectures such as MobileNetV3 [10] and EfficientNet [29]. We follow guidance from these works for a fair comparison. Unless the original papers recommend extra augmentation or training techniques, we minimize cross-entropy loss with the stochastic gradient descent with momentum optimizer in all our experiments. In addition, we cite known results from the literature for baseline reference.

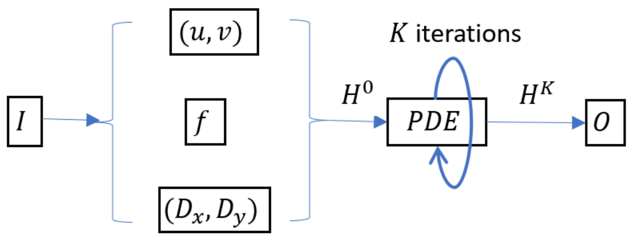


Figure 3. Schematic for the Global layer using the diffusion PDE.

We primarily report accuracy, the number of parameters, and the number of floating-point multiply-add operations[2]. These metrics measure the performance, model size, and computational footprint of a model. Due to lack of space, we tabulate depth, inference, and training times for only a few experiments. We do not pursue compression-related ideas (quantization, deep-compression, distillation, etc.) or hardware optimizations in this work to simplify the crux of our exposition. These can be further incorporated in our scheme to provide similar gains as reported in earlier works.

4.3. Results on MNIST-10

Since the ODE baselines do not scale to large-scale datasets, we compare our architectures with these baselines on MNIST-10 dataset. We use the Resnet architecture in this experiment. Similar to [3, 28], we use one Resnet architecture where we apply one Global or one ODE layer or 6 residual layers after downsampling the input twice. In addition, we use a budget (< 5M MACs) Resnet architecture where we apply one Global layer or residual layers without downsampling (see Supplementary Sec. A.3 for details). We minimize the cross-entropy loss using the SGD optimizer with momentum. We follow a similar experimental setup (epochs, learning rate, scheduler, etc.) as the baselines.

Results. *Resnet-Global at same accuracy has* $3-5\times$ *storage gains (number of parameters) and* $2.5-3\times$ *compute gains.* Table 2 compares the Resnet-Global model with Neural ODEs [3], NeuPDE [28], Resnet [8]. It shows that Resnet-Global achieves similar performance as the baselines while reducing the number of parameters and the compute requirements. In particular, compared to Resnet, our architectures reduce the storage by $3-4\times$ and reduce the compute by $2-3\times$. On the other hand, Neural-ODEs have $3\times$ higher compute footprint when compared to Resnet.

Table 2. Results on MNIST-10. Networks with a Global layer have significantly less storage and compute requirements than ODE, PDE, and discrete CNNs.

Architecture	Accuracy	#Params	#MACs
Neural ODEs [3]	99.49	220K	100M
NeuPDE [28]	99.49	180K	
Resnet [3, 8]	99.59	600K	30M
Resnet-Global (ours)	99.51	136K	14M
Resnet	99.61	33.3K	5.7M
Resnet-Global (ours)	99.43	9.94K	1.7M

4.4. Results on CIFAR-10 & CIFAR-100

Architectures. We evaluated popular residual architectures in this task, namely Resnet, Wide-Resnets, and DenseNets. We ran two variants of Resnet [8]: Resnet32 repeats same residual block $m = 5$ times at every resolution while Resnet56 increases the repetitions to $m = 9$. We strictly adhere to the configuration described in the original paper [8]. We replace the repeated blocks in these architectures with the Global layer, resulting in two Resnet-Global architectures where we keep $m = 1$ and $m = 2$ repetitions.

We used Wide-Resnet [32] with 40 layers and $4\times$ the width of the residual architecture, commonly referred to as WRN-40-4. It is constructed similar to the above-described Resnet with $m = 6$ repetitions except with $4\times$ the width. We replace these repetitions with a Global layer, resulting in Wide-Resnet-Global with $m = 1$. For DenseNet, we borrowed the cost-efficient variant DenseNet-BC [12] which has a growth rate of 12 and three dense blocks each with 16 dense layers, also known as Densenet-BC (k=12, L=100).

In addition, we apply the Global layer to an efficient architecture found by neural architecture search DARTS [22] which uses a cell found by the search. We modify this network by replacing the cell repetitions with a global layer.

Training Details. All the global architectures and their respective original baselines are trained with SGD+momentum optimizer for 300 epochs. We set the initial learning rate to 0.1 and decay by 10 at epoch 150 and 225. Note that for each architecture we use the recommended hyper-parameter settings (batch size, weight decay, data-augmentation etc.). When hyper-parameter recommendations are missing, we use grid search over weight decay $\{3e-4, 2e-4, 1e-4, 5e-5, 1e-5\}$ and batch size $\{32, 64, 128, 256\}$ on a validation set (see Supplementary Sec. A.4 for final hyper-parameters).

Results. We tabulate primary evaluation metrics in the the Table 3. Additional evaluation metrics such as train/inference times and architecture configurations are tabulated in Table 4. Below we summarize the main findings.

(a) Computational and storage savings. Table 3 shows that Global layer enabled architectures are compact and computationally efficient. In comparison to the baseline architectures, models with Global layer achieve $2.5-4.5\times$ flops reduction and $2.5-4.2\times$ parameters reduction.

(b) Lower training and inference times. It is evident from Table 4 that Global architectures have better training and inference times, with at least $2\times$ reductions. Note that we can discard the training time in many IoT applications as a one-time cost. In contrast, the inference time directly affects the battery drain and the responsiveness of the device.

(c) Shallower neural networks. From Table 4, one can deduce that Global architectures are shallower as they reduce the number of cells in the architecture by nearly $3\times$ in many instances. For example, the popular Resnet56 model has 27 cells while Resnet-Global only has 9 cells.

(d) Integrable in many popular architectures. Table 3 and 4 show that Global layer can be successfully applied across a range of architectures with the aforementioned benefits.

(e) Comparison at a fixed computational budget. It is

[2]Following convention [10, 29] we leverage the benchmarked PyTorch utility https://github.com/Lyken17/pytorch-OpCounter for MACs.

Table 3. Results on CIFAR-10 and CIFAR-100. Architectures with Global layer require $2-5\times$ less computational and storage budget.

Architecture	CIFAR-10			CIFAR-100		
	Accuracy	Params (Savings)	MACs (Savings)	Accuracy	Params (Savings)	MACs (Savings)
DenseNet-BC [12]	95.49	800K	300M	77.73	800K	300M
Resnet56 [8]	93.03	850K	127M	-	-	
NeuPDE [28]	95.39	9M		76.39	9M	
ANODE [5, 28]	94.96	11M		71.28	11M	
Hamiltonian PDE [24]	89.3	262K		64.9	362K	
MDEQ [2]	93.8	10M	8.3B	-	-	
Resnet32 (m=5)	92.49	460K (1.0×)	70M (1.0×)	68.57	473K (1.0×)	70M (1.0×)
Resnet-Global (m=1)	91.93	162K (2.8×)	15M (4.7×)	68.01	168K (2.8×)	15M (4.7×)
Resnet56 (m=9)	93.03	850K (1.0×)	127M (1.0×)	70.48	861K (1.0×)	127M (1.0×)
Resnet-Global (m=2)	93.01	330K (2.6×)	30M (4.2×)	70.06	336K (2.6×)	30M (4.2×)
Densenet	95.32	769K (1.0×)	297M (1.0×)	77.21	800K (1.0×)	297M (1.0×)
Densenet-Global	95.01	465K (1.7×)	136M (2.2×)	75.69	481K (1.7×)	136M (2.2×)
Wide-Resnet	95.91	9.0M (1.0×)	1.30B (1.0×)	79.11	9.0M (1.0×)	1.30B (1.0×)
Wide-Resnet-Global	95.54	2.8M (3.2×)	425M (3.1×)	78.13	2.8M (3.2×)	427M (3.1×)
DARTS	97.11	3.3M (1.0×)	539M (1.0×)	82.51	3.4M (1.0×)	539M (1.0×)
DARTS-Global	96.83	783K (4.2×)	213M (2.5×)	81.89	835K (4.1×)	213M (2.5×)
Resnet	80.76	13K	3.42M	35.21	14K	3.42M
Resnet-Global	82.55	14K	3.6M	43.62	16K	3.6M
Wide-Resnet	83.83	22K	9.8M	39.01	23K	9.8M
Wide-Resnet-Global	85.51	23K	8.7M	50.23	24K	8.7M
DARTS	86.05	39K	7.7M	54.57	43K	7.7M
DARTS-Global	88.44	34K	8.2M	60.68	41K	8.2M

worth noting that for an IoT device, although storage space could be a prohibitive factor for large models, the main issue with such small devices is computational in nature. The number of floating-point operations used for inference directly impacts the battery drain as well as the real-time latency required for any successful ML application. Keeping this in mind, we compare Global architectures with the baselines under a low computational budget, i.e., $<$ 10M MACs. At this regime, Global models achieve much higher accuracy than the baseline architectures. Thus, demonstrating that our models are better suited for IoT applications.

4.5. Results on Imagenet-1000

Experiments on MNIST and CIFAR datasets demonstrate that architectures with a Global layer are compact, shallower, and computationally efficient. In this section, we show that the Global layer improves the state-of-the-art models for the Imagenet dataset. It has been already shown in the literature that MobileNet [10, 25] and EfficientNet [29] models are more cost-efficient than the Resnet/Densenet models.

Architectures. We apply the global layer to MobileNetV2, MobileNetV3, and EfficientNet and replace the repetitions with the Global layer. For MobileNetV2, we use the baseline with a width multiplier of 1. We obtain MobileNetV2-Global by replacing all the invertible residual block repetitions with one Global layer at each feature resolution. We use the large variant of MobileNetV3 with a width multiplier of 1. We create MobileNetV3-Global by replacing the building block (invertible residual + squeeze-and-excite) with a Global layer. Finally, for the EfficientNet family, we only pick the B0 variant due to its low compute requirements. We provide the architecture details along with building blocks in the Supplementary Sec. A.5.

Training Details. Due to computing limitations, we do not re-train the baselines and only report their publicly known performance metrics in Table 5. We train the architectures with the Global layer from scratch using the hyper-parameter recommendations from the baselines. We use the RMSProp optimizer with 0.9 momentum to minimize the cross-entropy loss along with a weight decay term with a value of $1e-5$. The remaining hyper-parameters are available in the Supplementary Sec. A.5.

Table 4. CIFAR-10: Train & Inference times (cost of one pass through train and test dataset on a V100 GPU) along with the number of cells. Total cells are a proxy for depth of the network.

Architecture	Accuracy	Train Time(s)	Inference Time(s)	original cells	global cells	total cells
Resnet56	93.03	119	6.71	27	0	27
Resnet-Global	93.01	56	2.33	6	3	9
Densenet	95.32	138	10.22	48	0	48
Densenet-Global	95.01	24	3.4	24	2	26
Wide-Resnet	95.91	34	4.88	18	0	18
Wide-Resnet-Global	95.54	20	2.71	3	3	6
DARTS	97.11	126	6.86	20	0	20
DARTS-Global	96.83	61	3.43	6	3	9

Results. Table 5 reports the top1 accuracy along with the number of parameters and number of multiply-add operations. Below we summarize the main findings.

(a) Computational and storage savings. Table 5 shows that Global layer enabled architectures are compact and computationally efficient. Our architectures achieve similar performance as the baselines with $1.5\times$ fewer MACs and $2.5\times$ parameters savings. In addition, with a slight reduction in accuracy, we save $2\times$ MACs and $3\times$ parameters.

(b) Lower training and inference times. Global architectures have better training and inference times, with nearly $2\times$ reductions (see Supplementary Sec. A.5).

(c) Integrable in many popular architectures. Table 5 show that the Global feature layer can be successfully applied across a range of architectures.

Table 5. Results for Imagenet.

Architecture	Imagenet		
	Top-1	#Params	#MACs
MobileNet [11]	70.6	4.2M	575M
SqueezeNext [6]	67.44	3.23M	708M
DenseNet-169 [12]	76.2	14M	3.5B
Resnet-152 [8]	77.8	60M	11B
MDEQ-Small [2]	75.5	18M	
MobileNetV2 [25]	72.0	3.4M (1.0×)	300M (1.0×)
MobileNetV2-Global	71.63	1.6M (2.1×)	193M (1.6×)
MobileNetV2-Global-s	69.03	1.2M (2.8×)	150M (2.0×)
MobileNetV3 [10]	75.2	5.4M (1.0×)	219M (1.0×)
MobileNetV3-Global	74.11	3.0M (1.8×)	156M (1.4×)
MobileNetV3-Global-s	71.89	1.8M (3.0×)	110M (2.0×)
EfficientNet-B0 [29]	77.1	5.3M (1.0×)	390M (1.0×)
EfficientNet-B0-Global	76.12	2.4M (2.2×)	244M (1.6×)
EfficientNet-B0-Global-s	74.53	1.8M (2.9×)	201M (1.9×)

4.6. Ablative Experiments

In this section, we probe various aspects of the proposed method. Due to lack of space, we refer the reader to the Supplementary Sec. A.6 for additional ablations.

(A) Impact of K in iterative solver. We study the effect of the K hyper-parameter in the iterative solver. On CIFAR-10 and CIFAR-100 datasets, Table 6 shows the performance of the Resnet-Global model (see Sec. 4.4) as we vary K. Note that the increasing K does not increase the number of parameters in our formulation. In this case, for CIFAR-10, Resnet-Global has 162K parameters, while for CIFAR-100, Resnet-Global has 168K parameters. Our default choice of $K = 5$ is justifiable from the marginal improvement in accuracy with increased computational cost.

(B) Baseline and Global model with same MACs. Our earlier experiments showed that the Global layer improves the computational footprint of any architecture. In this ablation, we compare the Global architectures with the baselines by keeping the same computational budget. Table 7 shows the performance of these models on the CIFAR-100 dataset. Global models improve baselines by up to 4%.

Table 6. Effect of the hyper-parameter K in update Eq. 4.

Architecture	K	CIFAR-10		CIFAR-100	
		Accuracy	#MACs	Accuracy	#MACs
Resnet-Global	1	90.69	14.3M	66.89	14.3M
Resnet-Global	3	91.34	14.7M	67.37	14.7M
Resnet-Global	5	91.93	15M	68.01	15M
Resnet-Global	10	92.01	15.5M	68.23	15.5M
Resnet-Global	20	92.21	17M	68.24	17M

Table 7. Global models with similar budget as original models.

Architecture	CIFAR-100		
	Accuracy	#Params	#MACs
Resnet56	70.48	861K	127M
Resnet-Global	74.33	1.32M	119M
Densenet	77.21	800K	297M
Densenet-Global	78.91	922K	247M
Wide-Resnet	79.11	9M	1.3B
Wide-Resnet-Global	80.53	9M	1.3B
DARTS	82.51	3.4M	539M
DARTS-Global	84.19	2.4M	519M

(C) Non-linear Residual block as f. We compare the cost of using a standard Residual block as the function f in the update Eq. 4 instead of the identity function. Table 8 shows the performance characteristics of Resnet and WideResnet on CIFAR-10 and CIFAR-100. It shows that using Residual block only brings marginal improvements in accuracy ($< 0.5\%$) with a significant increase in compute cost.

Table 8. Ablative experiments to study the effect of the using a Residual block instead of our current choice in update Eq. 4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	f	Acc.	#Params	#MACs	Acc.	#Params	#MACs
Resnet	Identity	91.93	162K	15M	68.01	168K	15M
Resnet	Residual	92.28	175K	27M	68.37	181K	27M
WideResnet	Identity	95.54	2.8M	425M	78.13	2.8M	427M
WideResnet	Residual	95.67	3M	566M	78.34	3.1M	567M

5. Conclusion

We proposed a novel feature layer that couples the input and output feature map with PDE constraints. The proposed Global layer is readily deployable across many existing architectures. We show that the architectures with Global layers are more compact, shallower, and require less compute for inference and training. Empirical evaluations demonstrate that the proposed layer provides $2-5\times$ storage and computational savings. Since our work reduces model footprint, we do not foresee a negative societal impact.

Acknowledgement

This research was supported by Army Research Office Grant W911NF2110246, the National Science Foundation grants CCF-2007350 and CCF-1955981, ARM Research Inc., and the Hariri Data Science Faculty and Student Fellowship Grants.

References

[1] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 3

[2] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. 2020. 1, 2, 3, 7, 8

[3] 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. 1, 2, 5, 6, 12, 13, 15

[4] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 3140–3150. Curran Associates, Inc., 2019. 2

[5] Amir Gholami, Kurt Keutzer, and George Biros. ANODE: unconditionally accurate memory-efficient gradients for neural odes. *CoRR*, abs/1902.10298, 2019. 1, 2, 7

[6] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. *CoRR*, abs/1803.10615, 2018. 2, 8

[7] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. 2

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 2, 5, 6, 7, 8, 13, 16

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[10] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. 5, 6, 7, 8, 14, 15

[11] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 2, 8

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 5, 6, 7, 8, 13

[13] G D Hutomo, J Kusuma, A Ribal, A G Mahie, and N Aris. Numerical solution of 2-d advection-diffusion equation with variable coefficient using du-fort frankel method. 1180:012009, feb 2019. 4, 12

[14] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 2

[15] Anil Kag and Venkatesh Saligrama. Time adaptive recurrent neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15149–15158, June 2021. 3

[16] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5189–5200. PMLR, 18–24 Jul 2021. 17

[17] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*, 2020. 3

[18] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. 1, 2

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 5

[21] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. 2

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 5, 6, 13, 14, 17

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 5

[24] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations, 2018. 3, 5, 7

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 7, 8, 14

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 2

[27] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. 2

[28] Yifan Sun, Linan Zhang, and Hayden Schaeffer. NeuPDE: Neural network based ordinary and partial differential equations for modeling time-dependent data. In Jianfeng Lu and Rachel Ward, editors, *Proceedings of The First Mathematical and Scientific Machine Learning Conference*, volume 107 of

Proceedings of Machine Learning Research, pages 352–372, Princeton University, Princeton, NJ, USA, 20–24 Jul 2020. PMLR. 3, 5, 6, 7, 12, 13

[29] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 2, 5, 6, 7, 8, 15

[30] Trieu Trinh, Andrew Dai, Thang Luong, and Quoc Le. Learning longer-term dependencies in RNNs with auxiliary losses. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4965–4974. PMLR, 10–15 Jul 2018. 17

[31] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning, 2020. 1

[32] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 2, 5, 6, 13

[33] Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review, 2019. 1

[34] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2

A. Appendix

A.1. Experiments with other PDEs.

In this section, we explore few additional PDEs, namely with first order and Laplace differential operators. Our objective is to demonstrate that the Global layer can be naturally extended to any other PDE that is amenable to an iterative solver, thus yielding an efficient update equation.

$$\text{First Order} \quad \frac{\partial}{\partial x}H(x,y) + \frac{\partial}{\partial y}H(x,y) = f(I(x,y)) \tag{5}$$

$$\text{Second Order} \quad \frac{\partial^2}{\partial x^2}H(x,y) + \frac{\partial^2}{\partial y^2}H(x,y) = f(I(x,y)) \tag{6}$$

Similar to advection-diffusion PDE, these PDEs can be discretized by replacing the differential operators with finite differences. Yielding the following update equations. For simplicity, we take $\delta_x = \delta_y = \delta$,

$$\frac{H(x,y) - H(x-\delta,y)}{\delta} + \frac{H(x,y) - H(x,y-\delta)}{\delta} = f(I(x,y))$$
$$\implies H(x,y) = \frac{1}{2}\Big[H(x-\delta,y) + H(x,y-\delta) + \delta * f(I(x,y))\Big] \tag{7}$$

Similarly, the Laplace operator yields the following update

$$\frac{H(x+\delta,y) + H(x-\delta,y) - 2H(x,y)}{\delta^2} + \frac{H(x,y+\delta) + H(x,y-\delta) - 2H(x,y)}{\delta^2} = f(I(x,y))$$
$$\implies H(x,y) = \frac{1}{4}\Big[H(x+\delta,y) + H(x-\delta,y) + H(x,y+\delta) + H(x,y-\delta) - \delta^2 * f(I(x,y))\Big] \tag{8}$$

Table 9 shows the Global layer performance when we use the first order or second order differential operators. It also shows the performance of the update equation when the training and inference use different number of updates (i.e., the hyper-parameter K). For example, while training with large K yields good performance, during inference we can trade-off some accuracy for less number of updates, yielding faster inference.

Table 9. **Ablative Experiments on CIFAR-10 : Training with different iterative steps in the solver and inference with varying steps.**

Training			**Accuracy with Inference@K**			
PDE	**Initial Guess**		**K=1**	**K=5**	**K=7**	**K=10**
(First Order)	1-conv	**K=1**	90.71	75.44	61.87	43.02
		K=5	19.2	92.2	90.97	86.75
		K=7	10.01	88.41	92.23	90.19
		K=10	13.52	72.84	89.7	92.47
(First Order)	2-conv	**K=1**	93.17	72.15	62.24	28.56
		K=5	15.26	93.86	93.03	89.4
		K=7	23.21	92.16	93.87	92.79
		K=10	19.87	84.86	92.45	93.89
(Second Order)	1-conv	**K=1**	91.05	72.62	66.57	59.5
		K=5	12.13	91.84	90.65	87.45
		K=7	16.95	89.38	92.34	91.51
		K=10	10.51	65.98	89.13	92.18
(Second Order)	2-conv	**K=1**	93.08	73.34	64.71	55.49
		K=5	21.9	93.55	92.76	90.66
		K=7	17.66	90.14	93.54	93.25
		K=10	18.15	69.72	90.27	94.01

A.2. Discretizing the Diffusion PDE.

In this section, we discretize the following Advection-Diffusion PDE.

$$\begin{aligned}\frac{\partial}{\partial t}H(x,y,t) &+ \frac{\partial}{\partial x}\big(u(x,y,t)H(x,y,t)\big) + \frac{\partial}{\partial y}\big(v(x,y,t)H(x,y,t)\big)\\ &= \frac{\partial}{\partial x}\Big(D_x\frac{\partial}{\partial x}H(x,y,t)\Big) + \frac{\partial}{\partial y}\Big(D_y\frac{\partial}{\partial y}H(x,y,t)\Big) + f(I(x,y)) \end{aligned} \tag{9}$$

Assume the discrete steps for x, y and t by δ_x, δ_y and δ_t respectively. Following [13], we replace the partial differential operators with their finite difference, resulting in

$$\begin{aligned}\frac{H^t_{x,y} - H^{t-1}_{x,y}}{2\delta_t} &+ \Big(\frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x}H^t_{x,y} + u\frac{H^t_{x+1,y} - H^t_{x-1,y}}{2\delta_x}\Big) + \Big(\frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y}H^t_{x,y} + u\frac{H^t_{x,y+1} - H^t_{x,y-1}}{2\delta_y}\Big)\\ &= D_x\frac{H^t_{x+1,y} - H^{t+1}_{x,y} - H^{t-1}_{x,y} + H^t_{x-1,y}}{\delta_x^2} + D_y\frac{H^t_{x,y+1} - H^{t+1}_{x,y} - H^{t-1}_{x,y} + H^t_{x,y-1}}{\delta_y^2} + f(I(x,y))\end{aligned}$$

Thus, re-arranging the above equation, we obtain

$$\begin{aligned} LH^{k+1}_{x,y} &= (1 - 2B_x - 2B_y)H^{k-1}_{x,y} - 2EH^k_{x,y} + 2\delta_t f(I(x,y))\\ &+ (-A_x + 2B_x)H^k_{x+1,y} + (A_x + 2B_x)H^k_{x-1,y}\\ &+ (-A_y + 2B_y)H^k_{x,y+1} + (A_y + 2B_y)H^k_{x,y-1} \end{aligned} \tag{10}$$

where $L = (1 + 2B_x + 2B_y)$, and

$$u_x = \frac{u_{x+1,y} - u_{x-1,y}}{2\delta_x}; v_y = \frac{v_{x,y+1} - v_{x,y-1}}{2\delta_y}; E = (u_x + v_y)\delta_t$$

$$A_x = \frac{u\delta_t}{\delta_x}; A_y = \frac{v\delta_t}{\delta_y}; B_x = \frac{D_x\delta_t}{\delta_x^2}; B_y = \frac{D_y\delta_t}{\delta_y^2};$$

A.3. MNIST Experiments.

Architecture details.

- *Resnet baseline from Neural ODEs [3] and NeuPDE [28].* This network begins with a full convolutional layer with (64 output channels, 3×3 kernel, stride 1). It is followed by the batch norm and ReLU non-linearity. This is followed by another full convolutional layer (64 output channels, 3×3 kernel, stride 2), batch norm and ReLU non-linearity and full convolutional layer (64 output channels, 3×3 kernel, stride 2. Finally, there are 6 basic residual blocks (each consisting of 2 conv layers), followed by average pooling and the classifier layer.

- *Neural ODE architecture.* This network follows the exact same early layers as the above Resnet architecture except the residual layers are now replaced with one ODE layer. Thus, yielding gains in the parameter storage, but the computational footprint goes up, as the number of ODE updates are typically much larger than 6.

- *NeuPDE architecture.* This architecture replaces the residual block with 6 finite-difference layers introduced in [28].

- *Resnet-Global (136K params).* We use the same Resnet architecture described above and replace the 6 residual blocks with one Global layer discussed in Sec. 3.2. We use constant diffusion coefficients ($D_x = D_y = 1$) and use identity as the function f. We set the velocity (u, v) as the depthwise convolutional layers followed by batch norm and ReLU non-linearity.

- *Resnet (33K params).* This network begins with a full convolutional layer with (16 output channels, 3×3 kernel, stride 1). It is followed by the batch norm and ReLU non-linearity. This is followed by 5 basic residual blocks (each consisting of 2 conv layers), followed by average pooling and the classifier layer.

- *Resnet-Global (10K params).* This networks uses the earlier Resnet architecture ($33K$ params) and replaces the residual layers with one Global layer.

Hyper-parameter details. For all the MNIST experiments, we minimize the cross-entropy loss on the training data using the SGD optimizer with learning rate 0.1 and momentum 0.9. We use weight decay of $1e-4$ and batch size of 128. Following [3, 28], we train the model for 160 epochs with learning rate decay by $\frac{1}{10}$ after $60, 100$ and 140 epochs.

A.4. CIFAR Experiments.

Architecture details.

- *Resnet32, Resnet56.* These architectures come from the initial Residual Networks paper [8]. First, it has a full convolution layer (16 output channels, 3×3 kernel, stride 1), followed by basic residual blocks (16 output channels) repeated m times. This is followed by a downsampling convolutional layer (16 output channels, 3×3 kernel, stride 2). This process is repeated again, i.e. m basic residual blocks followed by downsampling layer but with 32 channels, and finally one last time m basic residual blocks with 64 channels. Remaining network consists of average pooling followed by fully connected layer. We obtain Resnet32 with $m = 5$ and Resnet56 with $m = 6$.

- *Resnet-Global ($m = 1, 2$).* We get the global variants of the above described Resnet architectures by replacing the repeated basic residual blocks with one Global layer. We get the Resnet-Global($m = 1$) by keeping only one Residual block as the initialization of the PDE and Resnet-Global ($m = 2$) by keeping two Residual blocks.

- *WideResnet.* This architecture is similar to the Resnet32 and Resnet56, except two changes, i.e. $m = 6$ and the width of the feature maps is multiplied with 4. Thus, feature sizes grow as $(64, 128, 256)$ in the network. This is commonly referred to as the WRN-40-4 [32].

- *WideResnet-Global.* Similar to the Global variants of the Resnet32 and Resnet56, we create the WideResnet-Global by replacing the repitions in the WideResnet with a Global layer.

- *Densenet.* We used the Densenet-BC variant from the Densenet paper [12] for its cost efficiency. We refer the reader to the original paper for detailed architecture setup. Here we provide minimal details to replicate this architecture Densenet-BC ($k = 12, L = 100$). We use the growth-rate 12 and 3 dense blocks with 16 dense layers. This network begins with a full convolution (16 output channels, 3×3 kernel, stride 1). This is followed by a dense block consisting of 16 dense layers, and a transition layer (batch-norm, ReLU, full conv (same output channels, 1×1 kernel, stride 1), then average pooling to down sample the feature map). This process is repeated two more times, yielding three dense blocks. Note that a Densenet concatenates all the feature maps within a dense block. Finally, a batch norm is applied on the resulting before forwarding to a classifier.

- *Densenet-Global.* Note that Densenet is not exactly repeating the same layer in each block (since the feature maps are concatenated). In each dense block, there are 16 dense layers and all the feature maps are forwarded as the output of this block. We cut down the size of this network by using only 8 dense layers in each dense block and apply a Global layer followed by the original transition layer. Thus, yielding Densenet-Global architecture.

- *DARTS.* [22] search for architecture cells using their differential architecture search. They found two optimized cells for the CIFAR dataset, namely, a normal and a reduction cell. A reduction cell transforms the feature map into a downsampled version of the feature map (similar to the operation performed by the transition layer in Densenet or the stride 2 convolution in Resnet). While a normal cell transforms the feature map into another feature map of the same dimension. We refer the user to their implementation (https://github.com/quark0/darts) for the details of these two cells. The final DARTS architecture used in our works is the same in their implementation, that consists of 20 such cells. First is a full convolution with 36 output channels, after which they apply their normal cells and have reduction cells after nearly every 6 normal cells. Followed by the classifier layer.

- *DARTS-Global.* We create the DARTS-Global variant by replacing all but repetitions of the normal cell with two and applying a Global layer at each resolution. We use identity as the function f. We set the velocity (u, v) and diffusion coefficients (D_x, D_y) as the depthwise convolutional layers followed by batch norm and ReLU non-linearity.

- *Budget-Resnet.* We use the same architecture as the Resnet32 but with reduced feature maps, i.e. instead of $(16, 32, 64)$ we have $(6, 8, 8)$ as the feature maps and same repetitions, i.e. $m = 5$. This yields a Resnet model with 13K parameters and nearly 3.4M MAcs.

- *Budget-Resnet-Global.* We use similar architectural setup as the Resnet-Global but we constrain the feature map sizes to be within 3.5M MACs. Instead of using $(16, 32, 64)$, we use $(10, 9, 16)$ as the feature map sizes. Note that this configuration is not a result of any architecture search, but instead a simply random allocation of the feature map to yield a model within similar compute characteristics of the Budget-Resnet model. Since our aim in the budget experiments is to show that with similar compute as the original model, the proposed model achieves much better performance, this configuration suffices. One can resort to architectural search schemes to find the optimized model with best performance, but that is beyond the scope of this work.

- *Budget-WideResnet.* This model is similar as the WideResnet model ($m = 6$ repetitions) except the number of feature maps go from $(64, 128, 256)$ to $(8, 8, 8)$.

- *Budget-WideResnet-Global.* Similar to WideResnet-Global ($m = 1$ repetitions) except the number of channels reduce from $(64, 128, 256)$ to $(12, 16, 12)$.

- *Budget DARTS.* Similar to the DARTS model with number of initial channels reduced from 36 to 3.

- *Budget DARTS-Global.* Similar to the DARTS-Global model with number of initial channels reduced from 36 to 5.

Hyper-parameter details. We follow the recommendations of the related works for various training strategies. We do only use the standard data augmentation discussed in the main text. For all our experiments, we use the SGD optimizer with momentum 0.9. All our models are trained for 300 epochs with pre-defined learning rate decay of $\frac{1}{10}$ at 150 and 225 epochs.

We search of other hyper-parameters (batch size and weight decay) whenever no recommendation is available from the baseline. We use the batch size of 64 for Densenet experiments while the rest of the experiments we use the batch size of 32. We use a weight decay of $1e-4$ for Resnet and Wide-Resnet experiments. While for the Global variants of these architecture we found $5e-5$ to yield best validation performance.

Note that for the DARTS and DARTS-Global experiments, we follow the setup described in [22], it includes cutout data augmentation, auxiliary losses from each reduced cell blocks with weight 0.4, a path dropout of 0.2 and a cosine learning rate scheduler. For computational purposes, we run both the baseline and Global variant only up to 300 epochs as opposed to the recommended 600 epochs. As per their setup, we use a batch size of 96 and weight decay of $3e-4$. For the DARTS-Global experiments, we use a weight decay of $8e-4$.

A.5. Imagenet Experiments.

Architecture details.

- *MobileNetV2.* We used the MobileNetV2 [25] architecture with width multiplier 1.0 (see Table 2 in MobileNetV2 paper). This network has nearly 3.4M parameters and 300M MACs.

- *MobileNetV2-Global.* We create the Global variant of the MobileNetV2 architecture by replacing the repetitions in all resolutions $(112, 56, 28, 14)$ except the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.

- *MobileNetV2-Global-s ($2\times$ less MACs).* We modify the earlier MobileNetV2-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

- *MobileNetV3.* We used the MobileNetV3-Large [10] architecture with width multiplier 1.0 (see Table 1 in MobileNetV3 paper). This network has nearly 5.4M parameters and 219M MACs.

- *MobileNetV3-Global.* We create the Global variant of the MobileNetV3-Large architecture by replacing the repetitions in all resolutions $(56, 28, 14)$ except the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.

- *MobileNetV3-Global-s ($2\times$ less MACs).* We modify the earlier MobileNetV3-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

- *EfficientNet-B0.* We used the EfficientNet-B0 [29] architecture with width multiplier 1.0 (see Table 1 in EfficientNet paper). This network has nearly 5.3M parameters and 390M MACs.
- *EfficientNet-B0-Global.* We create the Global variant of the EfficientNet-B0 architecture by replacing the repetitions in all resolutions $(112, 56, 28, 14)$ except the last repeated block. We learn the velocity and diffusion coefficients with depthwise convolutions.
- *EfficientNet-B0-Global-s ($2\times$ less MACs).* We modify the earlier EfficientNet-B0-Global variant by fixing the diffusion coefficients to be constants and even removing the repetitions in the last resolution.

Hyper-parameter details. We used RMSProp optimizer with momentum 0.9 for all the Imagenet experiments as per the experimental setup recommended by the baselines [10, 29]. We used weight decay $1e-5$. For both MobileNetv3 and EfficientNet-B0, we used dropout 0.2. We also used an exponential moving average (ema) with the decay of 0.9999. As recommended in the EfficientNet-B0, we use AutoAugment policy along with stochastic depth of 0.8. We used a batch size of 512 and learning rate 0.05, with the learning rate decay of about 0.97 every 2.4 epochs. We train these models for 300 epochs.

Inference and Training time comparison. Table 10 compares the inference and training time for various Imagenet models used in the main text. Global architecture shows up to $2\times$ reduction in inference and train time.

Table 10. Imagenet: Train & Inference times (cost of one pass through train and test dataset on a V100 GPU).

Architecture	Accuracy	Train Time(s)	Inference Time(s)
MobileNetV2	72.0	2181s	91s
MobileNetV2-Global	69.03	1414s	58s
MobileNetV3	75.2	1714s	71s
MobileNetV3-Global	71.89	1090s	45s
EfficientNet-B0	77.1	2667s	111s
EfficientNet-B0-Global	74.53	1311s	54s

A.6. Remaining Ablative Experiments.

A.6.1 Neural ODEs without equilibrium.

We compared the proposed layer vs Neural ODE layer with smaller number of iterations. Instead of running the Neural ODEs up to equilibrium, we unrolled the update equation to $K = 5$ steps (similar unrolling as our Global layer). Table 11 shows that even such an optimization does not improve its compute footprint when compared to a standard Resnet architecture. In contrast, Global layer uses at least $2\times$ less MACs than Resnet.

Table 11. Neural ODEs without equilibrium.

Architecture	Accuracy	#Params	#MACs
Neural ODEs [3]	99.49	220K	100M
Neural ODEs (no equilibrium)	99.46	220K	27M
Resnet [3]	99.59	600K	30M
Resnet-Global (ours)	99.51	136K	14M
Resnet	99.61	33.3K	5.7M
Resnet-Global (ours)	99.43	9.94K	1.7M

A.6.2 Impact of increasing K on compute time.

Table 12 provides inference time for the ablative experiment discussed in the main text (see Sec. 4.6(A) and the Table 6). This shows that the inference time increases sub-linearly with K.

A.6.3 Choice of free parameters.

In the main text, we studied some ablations on the choice of one free parameter, i.e., f. Here, we will study the impact of (u, v) as well as (D_x, D_y) on the performance characteristics of the global architectures (see Table 13, 14, & 15). We will use the following options for the free parameter initialization :

Table 12. Adding inference numbers for Resnet-Global architecture in Table 6 (Effect of the hyper-parameter K in update Eq. 4). Inference time is measured as the amount of time taken to pass through the test set on a V100 GPU.

K	CIFAR-10			CIFAR-100		
	Accuracy (%)	#MACs	Inference Time (s)	Accuracy (%)	#MACs	Inference Time (s)
1	90.69	14.3M	1.43	66.89	14.3M	1.57
3	91.34	14.7M	1.74	67.37	14.7M	1.83
5	91.93	15M	1.91	68.01	15M	1.98
10	92.01	15.5M	2.18	68.23	15.5M	2.22
20	92.21	17M	2.48	68.24	17M	2.59

- 'Residual(Basic)' : Used as the building block in many residual networks [8]
- 'Residual(Bottleneck)' : Used as the building block in residual networks with larger depth [8]
- 'FullConv': One layer full 3×3 convolution
- 'PwConv': One layer 1×1 convolution, also referred as the pointwise-convolution
- 'DwConv': One layer 3×3 depthwise convolution (does not involve any channel mixing)
- 'Identity' : No operation (simply pass through the features)

Table 13. (CIFAR-10/100) Ablative experiments to study the effect of the using different free parameter choice (D_x, D_y) than our current choice in update Eq. 4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	(D_x, D_y)	Acc.	#Params	#MACs	Acc.	#Params	#MACs
Resnet	Identity	91.26	159K	14.7M	67.15	165K	14.7M
Resnet	DwConv	91.93	162K	15M	68.01	167K	15.3M
Resnet	PwConv	92.17	170K	16.4M	68.34	176K	16.4M
Resnet	FullConv	92.12	256K	28.9M	68.39	262K	28.9M
Resnet	Residual(Basic)	92.68	353K	43M	68.53	359K	43.3M
Resnet	Residual(Bottleneck)	92.57	279K	32M	68.48	284K	32.4M

Table 14. (CIFAR-10/100) Ablative experiments to study the effect of the using different free parameter choice (u, v) than our current choice in update Eq. 4. Here, all architectures use the Global layer.

Global	Function	CIFAR-10			CIFAR-100		
Architecture	(u, v)	Acc.	#Params	#MACs	Acc.	#Params	#MACs
Resnet	Identity	91.36	159K	14.7M	67.07	165K	14.7M
Resnet	DwConv	91.93	162K	15M	68.01	167K	15.3M
Resnet	PwConv	92.11	170K	16.4M	68.23	176K	16.4M
Resnet	FullConv	91.98	256K	28.9M	68.28	262K	28.9M
Resnet	Residual(Basic)	92.47	353K	43M	68.45	359K	43.3M
Resnet	Residual(Bottleneck)	92.43	279K	32M	68.41	284K	32.4M

Table 15. (MNIST-10) Ablative experiments to study the effect of the using different free parameter choice (u, v, D_x, D_y) than our current choice in update Eq. 4. Here, all architectures use the Global layer.

Global	Function	(D_x, D_y)			(u, v)		
Architecture		Acc.	#Params	#MACs	Acc.	#Params	#MACs
Resnet	Identity	99.36	134K	13.9M	99.38	134K	13.9M
Resnet	DwConv	99.49	136K	14M	99.46	136K	14M
Resnet	PwConv	99.56	142K	14.3M	99.52	142K	14.3M
Resnet	FullConv	99.43	208K	16.6M	99.47	208K	16.6M
Resnet	Residual(Basic)	99.62	282K	19.3M	99.53	282K	19.3M
Resnet	Residual(Bottleneck)	99.59	225K	17.2M	99.54	225K	17.2M

A.6.4 Training & Inference Time Comparison.

We further extend the motivational example to CIFAR-100 dataset as well as another Resnet32 model with $m = 2$ repetitions. We show the results in Table 16. This shows that even when Resnet is equipped with same depth as the Resnet-Global model, it still has much worse compute and storage footprint. In addition, by reducing the number of repetitions, Resnet suffers a significant degradation in performance.

Table 16. CIFAR-10 & CIFAR-100 : Comparing discrete Resnet32 (m=5), ODE based Resnet32 (MDEQ[10]), our PDE embedded Resnet32-Global and Resnet32 (m=2) replacing Global layer with a Residual block in Resnet32-Global. We compute the depth as the number of blocks in the network. Inference time denote the cost of processing one pass of the test dataset on a V100 GPU.

CIFAR	Method	Acc.	#Params	#MACs	Train Time(s)	Inf. Time(s)
10	Resnet32 (m=5)	92.49%	460K	70M	78	4.45
	MDEQ	92.28%	1.1M	1.5B	409	23.32
	Resnet32-Global	91.93%	162K	15M	24	1.91
	Resnet32 (m=2)	89.42%	175K	27M	33	2.32
100	Resnet32 (m=5)	68.57%	473K	70M	82	4.57
	MDEQ	67.89%	1.1M	1.5B	413	25.66
	Resnet32-Global	68.01%	168K	15M	29	2.03
	Resnet32 (m=2)	63.35%	182K	27M	41	2.65

A.7. Discussion.

Comments about MDEQ Empirical Evaluations. MDEQ did an unfair comparison with Resnet. They added auxiliary losses at various input resolutions for the deep equilibrium features but fail to add such auxiliary losses to the Resnet training routine. It has been shown in the literature that adding auxiliar losses improves performance in neural networks [16, 22, 30] . They should have enabled such auxiliary losses in the Resnet experiments as well for fair comparison.

A.8. Advantages of the Global layer as compared to Neural ODEs and NeuPDE.

- As authors of NeuPDE point out (see Line 2-3 on pg 367 http://proceedings.mlr.press/v107/sun20a/sun20a.pdf), they are unable to scale-up to large-scale datasets such as ImageNet. That Neural ODE is not scalable to ImageNet datasets is well-known. For instance, see MDEQ https://arxiv.org/pdf/2006.08656.pdf Sec. 2, para 3, where this point is discussed. For reference, we repeat their point; even to handle MNIST, with input resolution $28 \times 28 \times 1$, Neural ODE must downsample to 7×7, and that its lack of scalability to realistic vision tasks arises from numerical instability. In fact, MDEQ while reporting ImageNet results, leaves out Neural ODEs. We also point out scalability in Section 2.

- MDEQ, which we compare extensively, can be thought of as the "scalable" version of Neural ODEs. Our reported results are significantly better than MDEQ (see Table 3 & 5).

- We emphasize that NeuPDE suffers the same scalability since they are really Neural ODEs but with the input replaced by a larger feature input such as monomials of the input, or a collection of partial derivatives of the input (see Eq. 1 or Eq. 9 in http://proceedings.mlr.press/v107/sun20a/sun20a.pdf). This is like substituting linear features with polynomial features in SVMs. Our approach has no connection to NeuPDEs. We learn a PDE and solve it to equilibrium (more in the spirit of DEQ/MDEQ).

A.9. Illustrative Example Visualizations.

Similar to the illustration in the method section, we add another representation for the letter 3 for a different input in the Figure 4. Here also, it is evident that Global layer representation highlights the corners very brightly in contrast to the representation from other feature backbones.

We show the velocity vectors associated with this new example in the Figure 5. One can clearly notice that there is some non-zero activity along the corners of the digit 3.

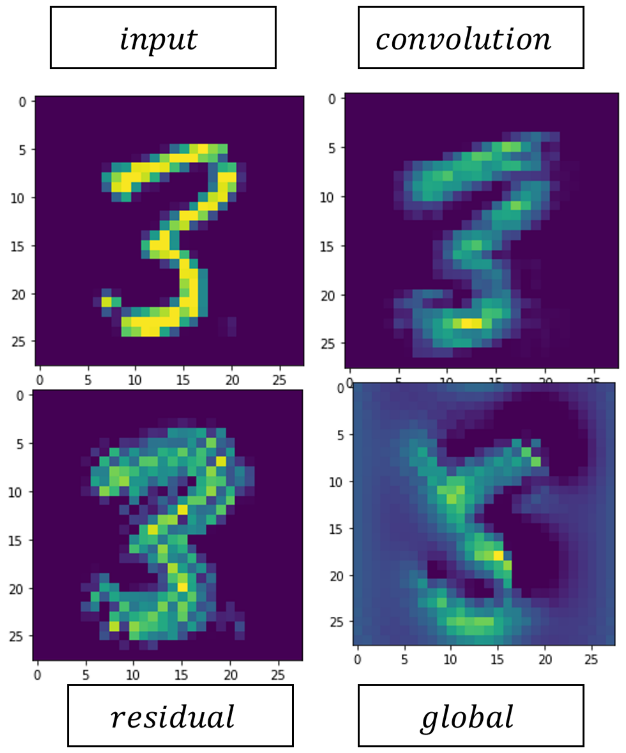


Figure 4. Another example to demonstrate the visual differences between different representation for the MNIST input letter 3

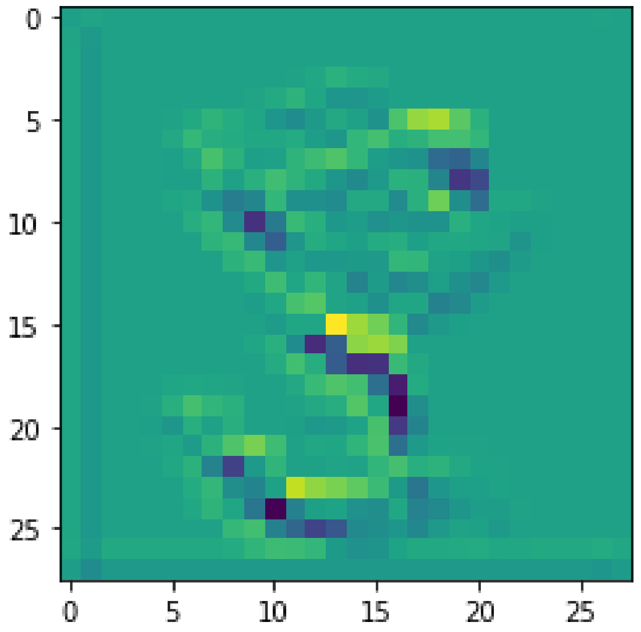


Figure 5. This represents the velocity vectors associated with the example in Figure 4. Note that there is some non-zero activity along the corners (represented by the very bright or very dark spots on the edges of the letter 3).