LOW-COST PARAMETERIZATIONS OF DEEP CONVOLUTIONAL NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Convolutional Neural Networks (CNNs) filter the input data using a series of spatial convolution operators with compactly supported stencils and point-wise nonlinearities. Commonly, the convolution operators couple features from all channels. For wide networks, this leads to immense computational cost in the training of and prediction with CNNs. In this paper, we present novel ways to parameterize the convolution more efficiently, aiming to decrease the number of parameters in CNNs and their computational complexity. We propose new architectures that use a sparser coupling between the channels and thereby reduce both the number of trainable weights and the computational cost of the CNN. Our architectures arise as new types of residual neural network (ResNet) that can be seen as discretizations of a Partial Differential Equations (PDEs) and thus have predictable theoretical properties. Our first architecture involves a convolution operator with a special sparsity structure, and is applicable to a large class of CNNs. Next, we present an architecture that can be seen as a discretization of a diffusion reaction PDE, and use it with three different convolution operators. We outline in our experiments that the proposed architectures, although considerably reducing the number of trainable weights, yield comparable accuracy to existing CNNs that are fully coupled in the channel dimension.

1 INTRODUCTION

Convolutional Neural Networks (CNNs) (LeCun et al., 1990) are among the most effective machine learning approaches for processing structured high-dimensional input data and are indispensable in, e.g., in recognition tasks involving speech (Raina et al., 2009) and image data (Krizhevsky et al., 2012). The essential idea behind CNNs is to replace some or all of the affine linear transformations in a neural network by convolution operators that are typically parameterized using small-dimensional stencils. This has a number of benefits including the increase of computational efficiency of the network due to the sparse connections between features, and a considerable reduction in the number of weights since stencils are shared across the whole feature map (Goodfellow et al., 2016).

In practical applications of CNNs, the features can be grouped into channels whose number is associated with the width of the network. This gives one several opportunities to define interactions between the different channels. Perhaps, the most common approach in CNNs is to fully couple features across channels (Gu et al., 2018; Goodfellow et al., 2016; Krizhevsky et al., 2012). Following this approach, the number of convolution operators at a layer is proportional to the product of the number of input and output channels. Given that performing convolutions is often the computationally most expensive part in training of and prediction with CNNs and the number of channels is large in many applications, this scaling can be problematic for wide architectures or high-dimensional data. Another disadvantage of this type of architecture is the number of weights. Indeed, for deep neural networks, the number of weights that are associated with a wide network can easily reach millions and beyond. This makes the deployment of such networks challenging, especially on devices with limited memory.

In this paper, we propose four novel ways to parameterize CNNs more efficiently, based on ideas from Partial Differential Equations (PDEs). Our goal is to dramatically reduce the number of weights in the networks and the computational costs of training and evaluating the CNNs. One ides, similar to Howard et al. (2017), is to use spatial convolutions for each channel individually and add

Table 1: Cost comparison of different convolution layers for an image with n pixels, stencil of size $m \times m$, and c input and output channels. RD denotes a reaction-diffusion architecture.

	no. of weights	computational costs
fully-coupled	$\mathcal{O}(m^2 \cdot c^2)$	$\mathcal{O}(n \cdot m^2 \cdot c^2)$
RD explicit	$\mathcal{O}(m^2 \cdot c + c^2)$	$\mathcal{O}(n(m^2 \cdot c + c^2))$
RD implicit	$\mathcal{O}(m^2 \cdot c + c^2)$	$\mathcal{O}(n(\log(n) \cdot c + c^2))$
RD circulant	$\mathcal{O}(m^2 \cdot c + c^2)$	$\mathcal{O}((nc)\log(cn))$

1×1 convolutions to impose coupling between them. Our architectures are motivated by the interpretation of residual neural networks (ResNets) (He et al., 2016a;b) as time-dependent nonlinear PDEs (Ruthotto & Haber, 2018). More specifically, we consider a simple Reaction-Diffusion (RD) model, that can model highly nonlinear processes. We derive new architectures by discretizing this continuous model, using 1×1 convolutions as a reaction term, together with cheap forms of a spatial convolution, that are similar to a depth-wise convolution in the number of parameters and cost. This spatial convolution acts similarly to a linear diffusion term that smooths the feature channels. Since the networks we propose originate in continuous models they have distinct theoretical properties that can be predicted using the standard theory of ODEs and PDEs (Ascher & Petzold, 1998).

Our first approach is designed to be employed in any existing CNN layer with equal number of input and output channels. We simply replace the traditional fully coupled convolution with a *linear* sum of depth-wise and 1×1 convolution, like a mask that can be placed on a traditional convolution in any existing CNN. Our second "explicit" RD architecture applies the operators separately with a non-linear activation function operating only following the 1×1 convolution, as the non-linear reaction part of the diffusion reaction equation. The third architecture is more unique. To improve the stability of the forward propagation and increase the spatial coupling of the features, we propose a semi-implicit scheme for the forward propagation through the network. Unlike traditional CNN operators, the semi-implicit scheme applies an inverse of the depth-wise (block diagonal) convolution preceded by a non-linear step involving the 1×1 convolution. This way, the scheme couples *all* the pixels in the image in one layer, even though we are using a depth-wise 3×3 convolution. The inverse of the convolution operator can be efficiently computed using Fast Fourier Transforms (FFT) and over the channels and kernels.

The last idea is to replace the depth-wise convolution structure with a circulant connectivity between the channels. This is motivated by the interpretation of the features as tensors and follows the definition of an efficient tensor product in (Kernfeld et al., 2015) whose associated tensor singular value decomposition has been successfully used for image classification in (Newman et al., 2017). The circulant structure renders the number of trainable convolution stencils proportional to the width of the layer. Using periodic boundary conditions in the other feature dimensions, this convolution can be computed efficiently by extending the FFT-based convolutions in (Mathieu et al., 2013; Vasilache et al., 2014) along the channel dimension, which reduces the cost from $\mathcal{O}(c^2)$ to $\mathcal{O}(c \log c)$ where c is the number of channels.

Table 1 compares the number of weights and the computational complexity associated with the forward propagation through a layer for the standard and reduced architectures. In the table we assume that the explicit RD architecture is directly computed without using FFT, but the FFT-based implementation, which is necessary for the implicit scheme, can also be used for the explicit one.

Our architectures pursue a similar goal than the recently proposed MobileNet architectures that are also based on a mix of 1×1 and "depth-wise" convolutions (Howard et al., 2017; Sandler et al., 2018). The MobileNet architecture involves with significantly less parameters, and in particular avoids the fully coupled convolutions, except for 1×1 convolutions which are cheaper in both computational cost and number of parameters. What sets our work apart from these architectures is that our architectures can be seen as discretization of PDEs, which allows to control their stability and offers new ways for their analysis.

The remainder of the paper is organized as follows. We first describe the mathematical formulation of the supervised classification problem with deep residual neural networks used in this paper. Subsequently, we propose the novel parameterizations of CNNs, describe their efficient implementation, and their computational costs. We perform experiments using the CIFAR10, CIFAR 100, and

STL10 datasets and demonstrate that the performance of the new architectures, despite a considerable reduction in the number of trainable weights, is comparable to residual neural networks using fully-coupled convolutions. Finally, we summarize and conclude the paper.

2 MATHEMATICAL FORMULATION

In this section, we introduce our main notation and briefly describe the mathematical formulation of the supervised classification problem used in this paper, which is based on (Goodfellow et al., 2016). For brevity we restrict the discussions to images although techniques derived here can also be used for other structured data types such as speech or video data.

Given a set of training data consisting of image-label pairs, $\{(\mathbf{y}^{(k)}, \mathbf{c}^{(k)})\}_{k=1}^{s} \subset \mathbb{R}^{n_f} \times \mathbb{R}^{n_c}$ and a residual neural network (ResNets) (He et al., 2016a;b), our goal is to find network parameters $\theta \in \mathbb{R}^n$ and weights of a linear classifier defined by $\mathbf{W}, \mu \in \mathbb{R}^{n_c}$ such that

$$\mathbf{c}^{(k)} \approx S(\mathbf{W}\mathbf{y}^{(k)}(\theta) + \mu), \quad \text{for all} \quad k = 1, 2, \ldots, s,$$

where S is a softmax hypothesis function and $\mathbf{y}^{(k)}(\theta)$ denotes the output features of the neural network applied to the kth image. As common, we model the learning problem as an optimization problem aiming at minimizing a regularized empirical loss function

$$\min_{\theta, \mathbf{W}, \mu} \frac{1}{s} \sum_{k=1}^{s} L(S(\mathbf{W}\mathbf{y}^{(k)}(\theta) + \mu), \mathbf{c}^{(k)}) + R(\theta, \mathbf{W}, \mu),$$

where in this paper L is the cross entropy and R is a smooth and convex regularization function. Solving the optimization problem is not the main focus of the paper, and we employ standard variants of the stochastic gradient descent algorithm (see the original work of Robbins & Monro (1951) and the survey of (Bottou et al., 2016)).

As common in other deep learning approaches, the performance of the image classification hinges upon designing an effective the neural network, i.e., the relation between the input feature $\mathbf{y}^{(k)}$ and its filtered version $\mathbf{y}^{(k)}(\theta)$. The goal in training is to find a θ that transforms the features in a way that simplifies the classification problem. In this paper, we restrict the discussion to convolutional ResNets, which have been very successful for image classification tasks. As pointed out by recent works (Haber & Ruthotto, 2017; Chang et al., 2017; Haber et al., 2017; E, 2017; Chaudhari et al., 2017; Ruthotto & Haber, 2018) there is a connection between ResNets and partial differential equations (PDEs), which allows one to analyze the stability of a convolution ResNet architecture.

We consider a standard ResNet as a baseline architecture. For a generic example $\mathbf{y}_0$, the jth "time step" of the network reads

$$\mathbf{y}_{j+1} = \mathbf{y}_j + \mathbf{F}(\theta_j, \mathbf{y}_j), \quad \text{for all} \quad j = 0, 1, \ldots, N-1, \tag{1}$$

where θ_j are the weights associated with the jth step. This step can be seen as a discretization of the initial value problem

$$\partial_t \mathbf{y}(t) = \mathbf{F}(\theta(t), \mathbf{y}(t)), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad t \in [0, T]. \tag{2}$$

In a simple ResNet, the nonlinear term in equation 1 and equation 2 usually reads

$$\mathbf{F}(\theta, \mathbf{y}) = \mathbf{K}(\theta^{(2)})^\top \sigma(\mathcal{N}(\mathbf{K}(\theta^{(1)})\mathbf{y})), \tag{3}$$

where $\sigma(x) = \max\{x, 0\}$ denotes a element-wise rectified linear unit (ReLU) activation function, the weight vector is partitioned into $\theta^{(1)}$ and $\theta^{(2)}$ that parameterizes the two linear operators $\mathbf{K}(\theta^{(1)})$ and $\mathbf{K}(\theta^{(1)})$, and $\mathcal{N}$ denotes a normalization layer that may have parameters as well (omitted here).

In CNNs the linear operator $\mathbf{K}$ in equation 3 is defined by combining spatial convolution operators, which gives a rich set of modeling options by, e.g., specifying boundary conditions, padding, strides, and dilation (Goodfellow et al., 2016). Our focus in this work is the coupling between different feature channels. Assuming that the input feature $\mathbf{y}$ can be grouped into c channels, the most common choice is to use full coupling in across the channels. As an example, let $\mathbf{y}$ consist of four channels

and the number of output channels be four as well. Then, the operator $\mathbf{K}(\theta)$ is a four by four block matrix consisting of convolution matrices $\mathbf{C}$ parametrized by the different stencils that comprise θ

$$\mathbf{K}(\theta) = \begin{pmatrix} \mathbf{C}\left(\theta^{(1)}\right) & \mathbf{C}\left(\theta^{(2)}\right) & \mathbf{C}\left(\theta^{(3)}\right) & \mathbf{C}\left(\theta^{(4)}\right) \\ \mathbf{C}\left(\theta^{(5)}\right) & \mathbf{C}\left(\theta^{(6)}\right) & \mathbf{C}\left(\theta^{(7)}\right) & \mathbf{C}\left(\theta^{(8)}\right) \\ \mathbf{C}\left(\theta^{(9)}\right) & \mathbf{C}\left(\theta^{(10)}\right) & \mathbf{C}\left(\theta^{(11)}\right) & \mathbf{C}\left(\theta^{(12)}\right) \\ \mathbf{C}\left(\theta^{(13)}\right) & \mathbf{C}\left(\theta^{(14)}\right) & \mathbf{C}\left(\theta^{(15)}\right) & \mathbf{C}\left(\theta^{(16)}\right) \end{pmatrix}. \tag{4}$$

Hence, applying a convolution operator going from c_{in} to c_{out} channels requires $\mathcal{O}(c_{\text{in}} \cdot c_{\text{out}})$ convolutions. Since practical implementations of convolutional ResNets often use hundreds of channels, this coupling pattern leads to large computational costs and to millions of parameters. Hence, we define more efficient coupling strategies in the next section.

3 Low-Cost Parameterizations of Convolution Operators

In this section, we present novel ways to parameterize the convolution operators in CNNs more efficiently. The first convolution operator is a simple sum of a depth-wise and 1×1 convolution, which can be thought of as a masked version of equation 4. The next two networks are discretizations of a new type of ResNet that are inspired by reaction-diffusion PDEs, where depth-wise spatial convolution operator is used as a diffusion process. The last approach imposes a block circulant structure on the diffusion operator in the reaction-diffusion PDE. We also provide detailed instructions on how to implement the proposed architectures efficiently.

3.1 The depth-wise and 1×1 convolutions

Most of our new architectures are defined using two types of operators. One is the depth-wise (block diagonal) operator which operates on each channel separately. Omitting the step index j, the operator in matrix form is given by

$$\mathbf{K}_{\text{dw}}(\theta) = \begin{pmatrix} \mathbf{C}\left(\theta^{(1)}\right) & & & \\ & \mathbf{C}\left(\theta^{(2)}\right) & & \\ & & \mathbf{C}\left(\theta^{(3)}\right) & \\ & & & \mathbf{C}\left(\theta^{(4)}\right) \end{pmatrix}. \tag{5}$$

Another building block is the fully-coupled 1×1 convolution operator that couples different channels but introduces no spatial filtering. For $\theta \in \mathbb{R}^{16}$, we denote such an operator as

$$\mathbf{M}(\theta) = \begin{pmatrix} \theta_1 & \theta_5 & \theta_9 & \theta_{13} \\ \theta_2 & \theta_6 & \theta_{10} & \theta_{14} \\ \theta_3 & \theta_7 & \theta_{11} & \theta_{15} \\ \theta_4 & \theta_8 & \theta_{12} & \theta_{16} \end{pmatrix} \otimes \mathbf{I}, \tag{6}$$

where $\otimes$ denotes the Kronecker product, and $\mathbf{I}$ is an identity matrix. Note that $\mathbf{M}(\theta)$ models a reaction between the features in different channels at a given pixel, but introduces no spatial coupling. In MobileNets (Howard et al., 2017), the operators equation 5 and equation 6 are used interchangeably in separate neural network layers, which also include non-linear activation and batch-normalization for each layer separately. We note that the depth-wise convolution is related to the "2D-filter" structurally sparse convolutions in (Wen et al., 2016). There, however, the authors allow the kernels to be all over the matrix and not only on the diagonal, and choose them via group ℓ_1 penalty.

Our first idea to simplify the coupled convolution in equation 4, is to combine the diagonal and off-diagonal weights into one operator that is multiplied as a standard matrix:

$$\mathbf{K}_{\text{lm}}(\theta^{(1)}, \theta^{(2)}) = \mathbf{K}_{\text{dw}}(\theta^{(1)}) + \mathbf{M}(\theta^{(2)}). \tag{7}$$

This is a masked version of the original convolution equation 4, with the mask leaving just the block diagonal and 1×1 convolution terms (note that the diagonal part of $\mathbf{M}(\theta^{(2)})$ can be ignored as the same coefficients appear also in $\mathbf{K}_{\text{dw}}(\theta^{(1)})$). This type of convolution can be used instead of the standard operators in CNNs, as long as the number of input and output channels are equal.

3.2 NETWORKS BASED ON THE REACTION-DIFFUSION EQUATION

As an alternative to the architecture in equations 1-3 we introduce a new class of CNNs based on the reaction-diffusion like equation

$$\partial_t \mathbf{y}(t) = -\mathbf{K}_{\mathrm{dw}}\left(\theta^{(1)}\right)^\top \mathbf{K}_{\mathrm{dw}}\left(\theta^{(1)}\right)\mathbf{y}(t) + \sigma\left(\mathcal{N}\left(\mathbf{M}\left(\theta^{(2)}\right)\mathbf{y}(t)\right)\right), \qquad \mathbf{y}(0) = \mathbf{y}_0. \tag{8}$$

Such equations have been used to model highly nonlinear processes such as pattern formation in complex chemical, biological and social systems. They typically lead to interesting patterns which suggests that they are highly expressive. Therefore, using such models for learning is very intriguing and a natural extension for standard ResNets.

3.2.1 EXPLICIT REACTION DIFFUSION CNN.

In the simplest, "explicit", discretization of the RD equation we use a ResNet structure of

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h\left(-\mathbf{K}_{\mathrm{dw}}\left(\theta_j^{(1)}\right)^\top \mathbf{K}_{\mathrm{dw}}\left(\theta_j^{(1)}\right)\mathbf{y}_j + \sigma\left(\mathcal{N}(\mathbf{M}(\theta_j^{(2)})\mathbf{y}_j)\right)\right), \tag{9}$$

where the time step $h > 0$ is chosen sufficiently small. The first symmetric and positive (semi) definite convolution operates as a diffusion on each channel separately, while the second term—the 1×1 convolution— models a reaction between the features in different channels at a given pixel, without introducing spatial coupling.

Both types of linear operators in equation 9 can be implemented efficiently. The fully coupled part in $\mathbf{M}$ is identical to the standard 1×1 convolution, and can be computed by a single call to a matrix-matrix multiplication BLAS routine (`gemm`) without any need to manipulate or copy the data. For 2D convolutions with filter size m the cost of this operation is a factor m^2 cheaper. The depth-wise operator $\mathbf{K}_{\mathrm{dw}}$ can be computed directly by applying standard convolutions on each of the channels. However, it can also be computed using FFT. Similar as above, each operator $\mathbf{C}$ in equation 5 can be evaluated by

$$\tilde{\mathbf{C}}\mathbf{y} = \mathbf{F}_2^{-1}\left((\mathbf{F}_2(\tilde{\mathbf{C}}\mathbf{e}_1)) \odot (\mathbf{F}_2\mathbf{y})\right), \tag{10}$$

where $\mathbf{F}_2$ and $\mathbf{F}_2^{-1}$ are the 2D FFT and inverse FFT, respectively, and $\mathbf{e}_1$ is the first standard basis vector. The cost of this computation scales linearly with the number of channels compared with the number of channels square of the standard fully connected convolution. The convolution is applied using the batched 2D FFT routines in the library `cufft`.

3.2.2 IMPLICIT REACTION DIFFUSION CNN.

Our second type of CNN may be seen as an "implicit" version of the previous convolutional layer, which is known to be a stable way to discretize the forward propagation equation 8. Here, we use a semi-implicit time-stepping

$$\mathbf{y}_{j+1} = \left(\mathbf{I} + h\mathbf{K}_{\mathrm{dw}}(\theta_j^{(1)})^\top \mathbf{K}_{\mathrm{dw}}(\theta_j^{(1)})\right)^{-1}\left(\mathbf{y}_j + h\sigma(\mathcal{N}(\mathbf{M}(\theta_j^{(2)})\mathbf{y}_j))\right). \tag{11}$$

The better stability of the implicit equation stems from the *unconditionally* bounded spectral radius of the first operator, specifically

$$\rho\left(\left(\mathbf{I} + h\mathbf{K}_{\mathrm{dw}}(\theta)^\top \mathbf{K}_{\mathrm{dw}}(\theta)\right)^{-1}\right) < 1, \quad \forall\theta.$$

Since this matrix is part of the Jacobian of the forward step (with respect to the input), the stability properties of the implicit forward propagation (Ruthotto & Haber, 2018) are better than those of its explicit counterpart. This behaviour is well known in the context of time-dependent PDEs. In addition, the implicit step has another advantage. It yields a global coupling between pixels in only one application, which allows features in one side of the image to impact features in other side. The coupling decays away from the center of the convolution and is strongly related to the Green's Function of the associated convolution.

We exploit the special structure of $\mathbf{K}_{\mathrm{dw}}$ to efficiently solve the linear system in equation 11 using $\mathcal{O}(c \cdot n \log(n))$ operations. While, in general, inverting a $K \times K$ matrix inversion requires a $\mathcal{O}(K^3)$

operation, the depth-wise kernel $\mathbf{K}_{\text{dw}}$ is block diagonal and therefore, the inverse is computed by solving each of the c blocks individually and in parallel. Since each diagonal block is a convolution, its inverse can be computed efficiently using a 2D FFT when assuming periodic boundary conditions. To this end we use the formula

$$(h\mathbf{C}^\top\mathbf{C} + \mathbf{I})^{-1}\mathbf{y} = \mathbf{F}_2^{-1}\left((h|\mathbf{F}_2\mathbf{C}\mathbf{e}_1|^2 + 1)^{-1} \odot (\mathbf{F}_2\mathbf{y})\right).$$

The FFT operations are essentially identical to the ones in equation 10, and hence have similar cost.

3.2.3 Block Circulant Convolution as the diffusion.

Another way to increase the computational efficiency of CNNs is based on the interpretation of the image data as a 3D tensor whose third dimension represents the channels. Following the notion of the tensor product in (Kernfeld et al., 2015), we define using the block circulant operator

$$\mathbf{K}_{\text{circ}}(\theta) = \begin{pmatrix} \mathbf{C}\left(\theta^{(1)}\right) & \mathbf{C}\left(\theta^{(2)}\right) & \mathbf{C}\left(\theta^{(3)}\right) & \mathbf{C}\left(\theta^{(4)}\right) \\ \mathbf{C}\left(\theta^{(4)}\right) & \mathbf{C}\left(\theta^{(1)}\right) & \mathbf{C}\left(\theta^{(2)}\right) & \mathbf{C}\left(\theta^{(3)}\right) \\ \mathbf{C}\left(\theta^{(3)}\right) & \mathbf{C}\left(\theta^{(4)}\right) & \mathbf{C}\left(\theta^{(1)}\right) & \mathbf{C}\left(\theta^{(2)}\right) \\ \mathbf{C}\left(\theta^{(2)}\right) & \mathbf{C}\left(\theta^{(3)}\right) & \mathbf{C}\left(\theta^{(4)}\right) & \mathbf{C}\left(\theta^{(1)}\right) \end{pmatrix}. \tag{12}$$

Using the associated tensor SVD has shown promising results (Newman et al., 2017) on the MNIST data set. We assume periodic boundary conditions (potentially requiring padding). Under this assumption, a matrix-vector product between the block circulant matrix with circulant blocks, $\mathbf{K}_{\text{circ}}$ and a feature vector, can be done using a Fast Fourier Transform (FFT), i.e.,

$$\mathbf{K}_{\text{circ}}\mathbf{y} = \mathbf{F}_3^{-1}((\mathbf{F}_3(\mathbf{K}_{\text{circ}}\mathbf{e}_1)) \odot (\mathbf{F}_3\mathbf{y})).$$

Here, $\mathbf{F}_3$ is a 3D Fast Fourier Transform (FFT), $\mathbf{F}_3^{-1}$ is its inverse (see, e.g., (Hansen et al., 2006) for details). The computational complexity of this product in proportional to $(nc)\log(nc)$ where n is the number of pixels, compared to the order of m^2nc^2 of the fully coupled convolution. It also requires much less parameters. While standard convolution requires $m^2 \cdot c^2$ variables our convolution requires only $m^2 \cdot c$ variables that we save in a 3D array.

4 Experiments

We experimentally compare the architectures proposed in this paper to the ResNet and MobileNet architectures using the CIFAR-10, CIFAR100 (Krizhevsky & Hinton, 2009) and STL-10 (Coates et al., 2011) data sets. Our primary focus is on showing that similar accuracy can be achieved using a considerably smaller number of weights in the networks. All our experiments are performed with the PyTorch software (Paszke et al., 2017).

Base Architecture. We use the same base architecture in all our numerical experiments, which is a slightly modified version of the one described in (Chang et al., 2017). We use three network sizes, but the overall structure is identical between them. Our goal is to use simple, standard, and rather small networks, and show that the new architectures can lead to a performance comparable to a standard ResNet architecture using even less parameters.

Our networks consist of several blocks, that are preceded by an opening layer. This opening layer is a convolutional layer with a 5×5 convolution that increases the number of channels from 3 to 32 or 48, depending on the network. This is followed by a batch normalization, and a ReLu activation. Then, there are several blocks (three or four), each consisting of a ResNet based part with four steps that varies between the different experiments except the ReLu activation and batch normalization. The architectures for the series of steps are:

- ResNet - a step with two fully coupled convolutions as defined in equation 1-equation 3.
- MobileNet - a two layer neural network similar to (Howard et al., 2017)

$$\hat{\mathbf{y}} = \sigma(\mathcal{N}(\mathbf{K}_{\text{bd}}(\theta^{(1)})\mathbf{y}_j)); \quad \mathbf{y}_{j+1} = \sigma(\mathcal{N}(\mathbf{M}(\theta^{(2)})\hat{\mathbf{y}})). \tag{13}$$

- LinearMix - a ResNet step using equation 7 as operators.
- Explicit / implicit RD - the architectures in equation 9, equation 11 respectively.

Table 2: Classification results

	CIFAR10		CIFAR100		STL10	
Architecture	Network	val. acc.	Network	val. acc.	Network	val. acc.
ResNet	A (1.5M)	93.1%	B (3.4M)	71.7%	A(1.5M)	74.9%
ResNet	B (3.5M)	93.0%	C (6.3M)	69.9%	B(3.5M)	75.3%
MobileNet	A (101K)	89.5%	B (251K)	65.6%	A(101K)	74.9%
MobileNet	B (216K)	91.6%	C (423K)	61.9%	B(216K)	77.2%
LinearMix	A (195K)	91.3%	B (456K)	67.9%	A(195K)	75.6%
LinearMix	B (422K)	92.1%	C (789K)	69.2%	B (422K)	75.6%
Exp. RD	A (101K)	88.9%	B (250K)	66.1%	A (101K)	74.6%
Exp. RD	B (216K)	90.6%	C (423K)	65.2%	B (216K)	75.9%
Imp. RD	A (101K)	88.7%	B (250K)	64.6%	A(101K)	73.8%
Imp. RD	B (216K)	90.3%	C (423K)	64.9%	B(216K)	73.4%
Circ. RD	A (101K)	86.0%	B (250K)	60.2%	A(101K)	69.6%
Circ. RD	B (216K)	88.0%	C (423K)	60.0%	B (216K)	70.5%

- Circular RD - the architectures in equation 9 with $\mathbf{K}_{\mathrm{circ}}$ in equation 12 instead of $\mathbf{K}_{\mathrm{dw}}$.

Each series of steps is followed by a single "connecting" layer that takes the images and concatenate them the same images multiplied with a depth-wise convolution operator and batch-normalization:

$$\mathbf{x} \leftarrow \mathcal{N}([\mathbf{x}; \mathbf{K}_{\mathrm{dw}}(\theta)\mathbf{x}]).$$

This doubles the number of channels, and following this we have an average pooling layer with down-sample the images by a factor of 2 at each dimension. We have also experimented with other connecting layers, such as a more standard 1×1 convolution either followed by pooling or with strides, leading to similar results. We use three networks that differ in the number of channels:

$$A : 32 - 64 - 128 \quad B : 48 - 96 - 192 \quad C : 32 - 64 - 128 - 256$$

The last block consists of a pooling layer that averages the image intensities of each channel to a single pixel and we use a fully-connected linear classifier with softmax and cross entropy loss.

As the number of parameters is typically small, we do not use regularization for training the networks. For training the networks we use the ADAM optimizer (Kingma & Ba, 2014), with its default parameters. We run 300 epochs and reduce the learning rate by a factor or 0.5 every 60 epochs, starting from 0.01. We used a mini-batch size of 100. We also used standard data augmentation, i.e., random resizing, cropping and horizontal flipping.

Data Sets: The CIFAR-10 and CIFAR100 datasets (Krizhevsky & Hinton, 2009) consists of 60,000 natural images of size 32×32 with labels assigning each image into one of ten categories (for CIFAR10) or 100 categories (for CIFAR100). The data are split into 50,000 training and 10,000 test images. The STL-10 dataset (Coates et al., 2011) contains $13,000$ color-images each of size 96×96 that are divided into $5,000$ training and $8,000$ test images that are split into the ten categories.

Our classification results are given in Table 2. The results show that our different architectures are in par and in some cases better than other networks. The theoretical properties of our architectures can be explained by the standard theory of ODEs and PDEs which makes them more predictable given small perturbations in the network parameters (for example, truncation errors) or noise in the data (Ruthotto & Haber, 2018). The architecture used is computationally efficient and the efficiency increases as the number of channels increases. For many problems where the number of channels is in the thousands, our approach can yield significant benefits compared with other architectures.

4.1 Computational Performance

In this section we compare the runtime of the forward step of our FFT-based convolutions relatively to the runtime of the fully coupled convolution in equation 4, which is computed using the `cudnn` package version 7.1. This package is used in all of the GPU implementations of CNN frameworks known to us. Both the circular and depth-wise convolutions are implemented using `cufft` as noted above. For the direct depth-wise convolution, we use PyTorch's implementation using `groups`.

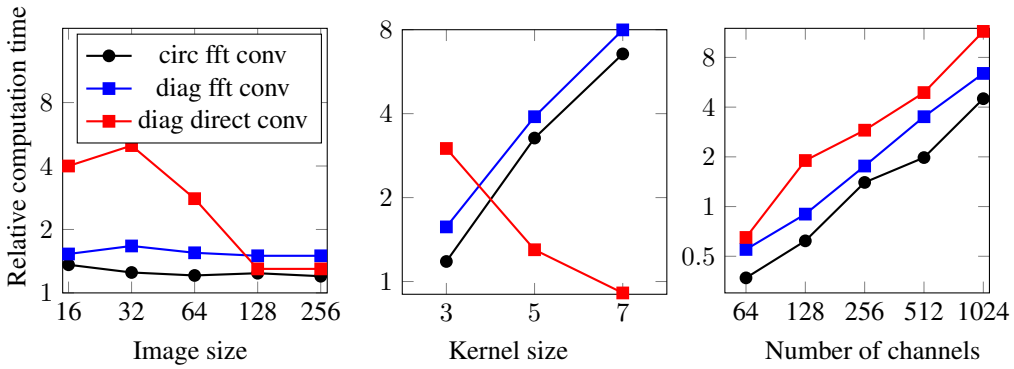

Figure 1: Runtime ratio between `cudnn`'s fully coupled convolution, depth-wise convolution and our implementation of the depth-wise (explicit or implicit) and circular convolutions. Except the measured parameter, the default parameters for all tests are: batch size: 64, image size: 64^2, number of channels: 256, kernel size 3×3.

The experiments are run on a Titan Xp GPU. We report the relative computation time

$$\frac{time(\text{Fully coupled using } \texttt{cudnn})}{time(\text{Our implementation})},$$

so we wish that the ratio is large. We note that the convolutions equation 12 and equation 5 can also be applied using `cudnn` by manually forming the convolution kernel (zero-filled, or circularly replicated), and hence we can use `cudnn` in cases where the time ratio is smaller than 1.

Our tests are reported in Fig. 1. The presented runtime ratio was calculated based on the total time of 100 convolutions. The left most graph presents the execution time ratio with respect to the image size. As expected, the execution time ratio does not depend on the image size in all convolutions, except for the direct method at small scales, which may be faster due to efficient memory access. The middle graph presents the execution time ratio with respect to the stencil size. Here, since both of our implementations apply the FFT on a zero-padded kernel weights, their execution time is independent of the kernel size, and the time ratio compared to the direct `gemm`-based cudnn implementation improves as the kernel grows. The FFT-based implementation is favorable if one wishes to enrich the depth-wise convolution models with wider stencils. The left-most graph presents the execution ratio with respect to the number of channels. Here we can clearly see that the execution ratio is linear in the number of channel, with a ratio of 1 achieved at about 200 channels for the FFT-based implementations. The direct convolution has a better constant, but the overall complexity is similar. Clearly, the considered convolutions are more favorable for wide networks.

5 Discussion and Conclusion

We present four new convolution models with the common goal of reducing the number of parameters and computational costs of CNNs. To this end, we propose alternative ways to the traditional full coupling of channels, and thereby obtain architectures that involve fewer expensive convolutions, avoid redundancies in the network parametrization, and thereby can be deployed more widely. Our work is similar to that of (Howard et al., 2017; Sandler et al., 2018). However, our unique angle is the close relation of our architectures to continuous models given in terms of PDEs that are well understood. This highlights stability of our CNNs and paves the way toward more extensive theory.

Our numerical experiments for image classification show that the new architectures can be almost as effective as more expensive fully coupled CNN architectures. We expect that our architectures will be able to replace the traditional convolutions in classification of audio and video, and also in other tasks that are treated with CNNs. It is important to realize that our new architectures become even more advantageous for 3D or 4D problems, e.g., when analyzing time series of medical or geophysical images. In these cases, the cost of each convolution is much more expensive and the computational complexity makes using 3D CNNs difficult. Here, also the number of weights imposes challenges when using computational hardware with moderate memory.

REFERENCES

Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. abs/1709.03698, 2017. URL `http://arxiv.org/abs/1709.03698`.

P Chaudhari, A Oberman, Stanley Osher, S Soatto, and G Carlier. Deep Relaxation: Partial Differential Equations for Optimizing Deep Neural Networks. pp. 1–22, 2017.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 14th AISTATS*, pp. 215–223, 2011.

Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, May 2018.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, abs/1705.03341:1–21, 2017. URL `http://arxiv.org/abs/1705.03341`.

Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales - A multiscale method for convolution neural networks. abs/1703.02009:1–8, 2017. URL `http://arxiv.org/abs/1703.02009`.

P C Hansen, J G Nagy, and D P O'Leary. *Deblurring Images: Matrices, Spectra and Filtering*. Matrices, Spectra, and Filtering. SIAM, Philadelphia, PA, 2006.

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*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016b.

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. *arXiv preprint arXiv:1704.04861*, 2017.

Eric Kernfeld, Misha Kilmer, and Shuchin Aeron. Tensor–tensor products with invertible linear transforms. *Linear Algebra and its Applications*, 485:545–570, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Adv Neural Inf Process Syst*, 61:10971105, 2012.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Y LeCun, B E Boser, and J S Denker. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast Training of Convolutional Networks through FFTs. December 2013.

Elizabeth Newman, Misha Kilmer, and Lior Horesh. Image classification using local tensor singular value decompositions. *arXiv.org*, June 2017.

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. In *Advances in Neural Information Processing Systems*, 2017.

Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *26th ICML*, pp. 873–880, New York, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553486. URL http://portal.acm.org/citation.cfm?doid=1553374.1553486.

H Robbins and S Monro. A Stochastic Approximation Method. *Ann. Math. Stat.*, 1951. doi: 10.2307/2236626. URL http://www.jstor.org/stable/2236626.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

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*, pp. 4510–4520, 2018.

Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast Convolutional Nets With fbfft: A GPU Performance Evaluation. *arXiv.org*, December 2014.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.