Convolutional Filter Approximation Using Fractional Calculus

Julio Zamora Jesus A. Cruz Vargas Anthony Rhodes Lama Nachman
Narayan Sundararajan
Intel Labs
Santa Clara (HQ), CA. United States.
{julio.c.zamora.esquivel,adan.cruz,anthony.rhodes}@intel.com,
{lama.nachman,narayan.sundararajan}@intel.com

Abstract

We introduce a generalized fractional convolutional filter (FF) with the flexibility to behave as any novel, customized, or well-known filter (e.g. Gaussian, Sobel, and Laplacian). Our method can be trained using only five parameters – regardless of the kernel size. Furthermore, these kernels can be used in place of traditional kernels in any CNN topology. We demonstrate a nominal $5X$ *parameter compression per kernel as compared to a traditional* (5×5) *convolutional kernel, and in the generalized case, a compression from* $N \times N$ *to 6 trainable parameters per kernel. We furthermore achieve* $43X$ *compression for 3D convolutional filters compared with conventional* $(7 \times 7 \times 7)$ *3D filters. Using fractional filters, we set a new MNIST record for the fewest number of parameters required to achieve above 99% classification accuracy with only* $3,750$ *trainable parameters. In addition to providing a generalizable method for CNN model compression, FFs present a compelling use case for the compression of CNNs that require large kernel sizes (e.g. medical imaging, semantic segmentation).*

1. Introduction

Computer vision has generated an impressive array of increasingly sophisticated techniques over the last decade, driven chiefly by Convolutional Neural Networks (CNN). Modern deep learning models have achieved state-of-the-art accuracy in a variety of tasks including image classification [8, 32, 53], semantic segmentation [42, 38], object detection [5], pose detection [41], and more. Much of the research in this area is rooted in the development of increasingly deep architectures, comprised of millions (even billions [49]) of trainable parameters to elicit performance increases. Due to the memory and computational constraints of these large-scale models, many of these outstanding results have yet to fully translate over to edge computing devices and other compute constrained environments.

As an alternative to training and deploying unwieldy, overparameterized models, researchers have recently focused on more sustainable network designs [56] in an attempt to generate smaller, more efficient models at the cost of a potentially minor trade-off in accuracy. There have been many approaches proposed in this vein, including ShuffleNet [62], MobileNet [20], HENet [48], and SqueezeNet [22].

Motivated by prior research exploring the addition of adaptive parameters to activation functions [60] and convolutional kernels [61], here we explore the application of fractional calculus [47] to the creation of fractional kernels for CNNs. In this work we apply concepts from fractional calculus to enable a neural network to learn a reduced representation of a convolutional kernel in functional form, i.e., as a *fractional kernel*. In addition, we demonstrate that neural networks utilizing fractional kernels perform comparably to state-of-the-art models on several benchmark data sets (MNIST [35], CIFAR-10 [31], ImageNet [8] and UCF101 [30]), but with a significant reduction in the number of kernel parameters in the compressed layers. We show that this novel convolutional paradigm facilitates the creation of generalized high performance architectures that are more efficient with respect to their memory and compute resource consumption.

In particular, our analysis focuses on larger kernel sizes (e.g., 5×5, 7×7 and above), as our method can leverage the benefits of these larger kernels while yielding appreciable increases in model compression rates. Recent research [46, 37, 58, 52] has demonstrated the superior performance of CNNs using large kernel sizes on high fidelity computer visions tasks including semantic segmentation, super resolution upsampling, and medical imaging; moreover, the positive benefits of using larger kernels to increase the *effective receptive field* of a CNN is well-documented [39, 2]. Despite their proven utility, large kernel size CNNs are nevertheless currently underutilized because of their prohibitive

memory and compute requirements; our research is meant to address this gap.

The main contributions of this work are as follows:

- We describe a methodology to define a convolutional filter of any size with only 6 parameters, a substantial reduction from the $N \times N$ parameters used in traditional convolutional filters.
- We demonstrate a compression rate of ($\frac{N \times N}{6}$) in training parameters required by our method over baseline CNN architectures in the compressed layer for two-dimensional filters.
- We demonstrate a compression rate of ($\frac{N \times N \times N}{8}$) in training parameters required by our method over baseline CNN architectures in the compressed layer for three-dimensional filters.
- We set a new MNIST classification record for the smallest number of parameters (3,750) required to achieve above 99% accuracy.
- A generalized, efficient methodology to convert any conventional CNN to an FF-based CNN.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 presents a detailed explanation of fractional calculus, the mathematics that underpins this work. In the Section 4, we provide a formal description of fractional filters. Section 5 highlight the implementation details of the fractional filters. The results and discussion, and our conclusions are presented in Sections 6 and 7, respectively.

2. Related work

In the main, research in computer vision over the past decade has tended to leverage the benefits of the compositional structure of high-capacity, deep networks [12, 40]. Recent work [18, 10], however, reveals that many deep models suffer from severe inefficiencies due to the presence of gross overparameterization. These discoveries have spurred interest in the development of more efficient CNNs.

SqueezeNet [22] and MobileNets [20], for example, compress the convolutional kernels by using $N \times N \times 1$ kernels instead of $N \times N \times 3$, thus reducing the number of channels processed by the convolutional layer, yielding a reduced model in the number of trainable parameters.

In [4], the authors demonstrate a dynamic filter framework in which a network generates a single filter used by all nodes in the the first layer of convolutional kernels and is trained with the other network parameters. This network takes an input I_A, with shape $h \times w \times c_A$ where h, w, and c_A are the height, width and number of input channels respectively, and outputs filters F_θ parameterized by $\theta \in R^{s \times s \times c_A \times n \times d}$ where $s \times s$ is the kernel size, n is the number of nodes in the layer, and d is 1 for dynamic convolution or $h \times w$ for dynamic local filtering.

Pruning represents a common technique used to reduce the memory and compute overhead required by overparameterized models [13, 9, 14]. Pruning methods are not in conflict with the present work; both pruning and fractional filters can be applied in tandem to further augment model compression results. Similarly, memory-reducing approaches such as quantization [21] can also be applied in concert with our technique.

The theory of fractional calculus [6] has previously been applied to neural network design. In [60], the authors use fractional calculus to group existing activation functions into families by defining the fractional order of a primitive activation function that is tuned during training. In this way, each neuron in the network learns a bespoke activation function. The authors demonstrate, for instance, that ResNet-18 utilizing this adaptive activation method can outperform a ResNet-100 topology [15].

3. Fractional Calculus

This section describes the motivation behind the use of fractional derivatives and integrals, as a way to define a numerically trainable hyperparameter value to automatically select an optimal convolutional kernel in a neural network.

3.1. Fractional Derivative

In its conventional form, the mathematical derivative is associated with natural values (i.e. the first derivative of a function, second derivative of a function, etc.), and can be defined as:

$$y' = \frac{\mathrm{d}y}{\mathrm{d}x}, y'' = \frac{\mathrm{d}^2 y}{\mathrm{d}x}, y''' = \frac{\mathrm{d}^3 y}{\mathrm{d}x}, \tag{1}$$

where the preceding expressions represent the first, second, and third order derivative of the function y with respect to x. In recent years, fractional calculus has successfully served as a tool for modeling complex dynamics [59], for understanding wave propagation [19], and for working with quantum physics [33], among other applications.

To understand how a fractional derivative works, we begin with a simple illustrative example. Here, we show that the natural n-derivatives of the function $f(x) = x^k$ are defined as:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = kx^{k-1}, \tag{2}$$

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = k(k-1)x^{k-2}, \tag{3}$$

$$\frac{\mathrm{d}^\mathrm{a} f(x)}{\mathrm{d}x^a} = k(k-1)(k-2)\cdots(k-a+1)x^{k-a}. \tag{4}$$

Equation 4 can be rewritten as a product of the factorial operation as:

$$\frac{\mathrm{d}^{\mathrm{a}} f(x)}{\mathrm{d}x^a} = \frac{k!}{(k-a)!} x^{k-a}, \tag{5}$$

For the case above, the factorial operator can only be defined for non-negative integer numbers. In order to generate a fractional derivative, the factorial operator can be replaced by the Gamma function Γ as proposed in [1]:

$$\Gamma(z) = \int_0^\infty t^{(z-1)} e^{-t} dt, \tag{6}$$

For the particular case of $n \in \mathbb{N}$:

$$\Gamma(n) = (n-1)!, \tag{7}$$

a known efficient method to compute Gamma is [7]:

$$\Gamma(z) = \frac{e^{-\gamma z}}{z} \prod_{k=1}^{\infty} \left(\left(1 + \frac{z}{k}\right)^{-1} e^{\frac{z}{k}} \right), \tag{8}$$

where γ is the Euler-Mascheroni constant ($\gamma = 0.577\ldots$) [1]. Thus, replacing the factorial in equation 5 by the Gamma function, the fractional derivative is then given by [17]:

$$D^a f(x) = \frac{\mathrm{d}^{\mathrm{a}} f(x)}{\mathrm{d}x^a} = \frac{\Gamma(k+1)}{\Gamma(k+1-a)} x^{k-a}. \tag{9}$$

The above definition represents the fractional derivative of function $f(x) = x^k$ valid for $k, x \geq 0$. Using analogous definitions, one can similarly construct *fractional integrals*; we omit such a discussion for brevity.

In the following sections, we apply these fractional calculus concepts to define a generalized convolutional filter function that can be used to render a *fractional kernel* for a convolutional neural network.

4. Fractional Filter

The most popular filters used in computer vision include the Gaussian [11], Sobel [26], Laplacian [11] and the so-called Mexican Hat [11] filters. These filters are related through the derivatives of a Gaussian filter. The fractional derivative concept from fractional calculus theory [43] can be used to generate any of these conventional filters, as well as an infinite number of novel filters that represent interpolations between these filters. To this end, we approximate the fractional derivative of a Gaussian function using a truncated Taylor series as follows:

$$D^a G = \frac{1}{uh^a} \sum_{n=0}^{15} \frac{(-1)^n \Gamma(a+1)}{\Gamma(n+1)\Gamma(1-n+a)} \left(e^{-\frac{(x-nh)^2}{u^2}} \right) \tag{10}$$

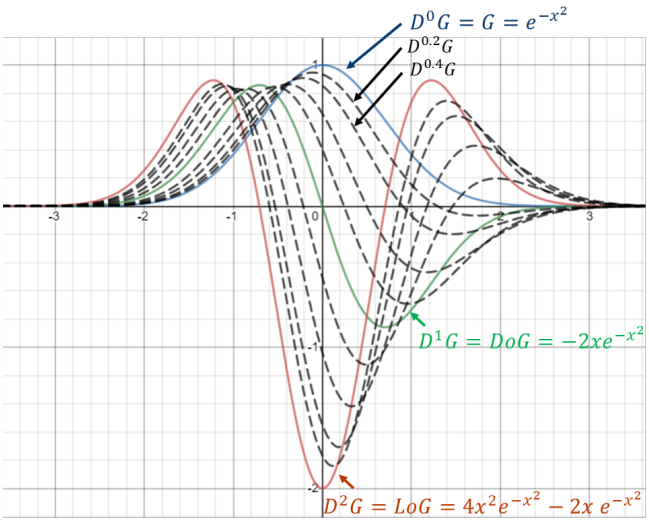

Figure 1. Plots generated by our fractional filter definition in Equation 10. We plot the family of functions generated by taking the fractional derivative of the Gaussian filter (blue), and compare it with its first derivative (Derivative of Gaussian, DoG, in green), and its second derivative (Laplacian of Guassian, LoG, in red). The dashed black functions are the interpolations generated using our fractional filter.

where $G = e^{-x^2}$ is the Gaussian function, a is the fractional derivative order and $\Gamma(a)$ represents the gamma function, defined in 6. Thus, our fractional filter allows for the use of a single general filter that can be tuned for different applications. By changing a single trainable parameter (*viz.*, the order of the fractional derivative) it is possible to generate every fractional instance between the Gaussian filter and the Laplacian filter, as shown in Figure 1.

4.1. Two-Dimensional Fractional Filters

In order to provide a comprehensive description of a generalized two-dimensional filter, we first briefly review three of the most frequently encountered filters in computer vision and their underlying mathematical relationships. When convolved with an image, the Gaussian filter introduces a blur by removing (or filtering out) high frequency image components. The Gaussian filter is defined by the following equation:

$$G(x, y) = e^{-\frac{x^2+y^2}{\sigma^2}} \tag{11}$$

The Sobel filter [27] approximates the first derivative of a Gaussian (DoG) filter. In classical image processing, the Sobel filter is often used to detect sharp borders or edges in an image by reducing the low frequency components and amplifying high frequency features. The mask of this filter is given by the equation:

$$\frac{\partial G(x, y)}{\partial x} = x e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{12}$$

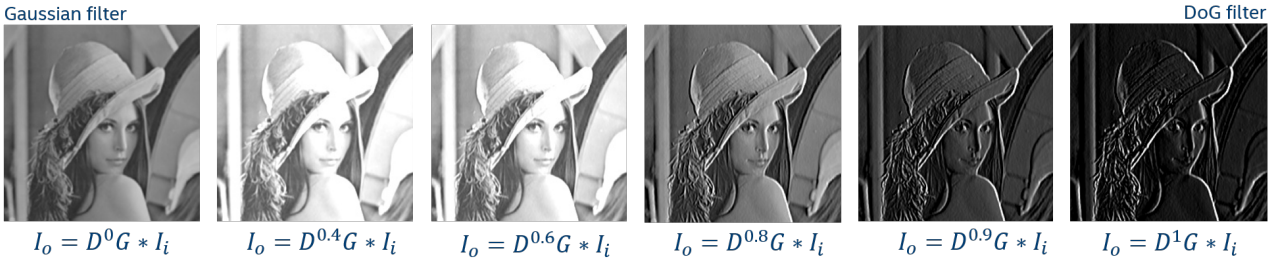

Figure 2. Visualization of repeated application of a fractional filter to the same source image for a range of α parameter values from $[0, 1]$ specifying the fractional derivative of the filter. We illustrate that the fractional filter can interpolate between a Gaussian and a DoG filter by changing a single parameter.

The Laplacian filter is defined by the second derivative of the Gaussian filter. This filter can also be used for edge detection. In application to computer graphics, it is often used to simulate bump mapping. One can generate this filter using the Laplacian operator:

$$\frac{\nabla^2 G(x)}{\partial x} = 4x^2 e^{-x^2} - 2e^{-x^2} \tag{13}$$

The Mexican hat filter can be characterized as the sign-inverted second order derivative of the Gaussian filter [25]. This filter is frequently used as a blob detector. All of the aforementioned classical image filters can be rendered as n-order derivatives of a Gaussian function (see Figure 3).

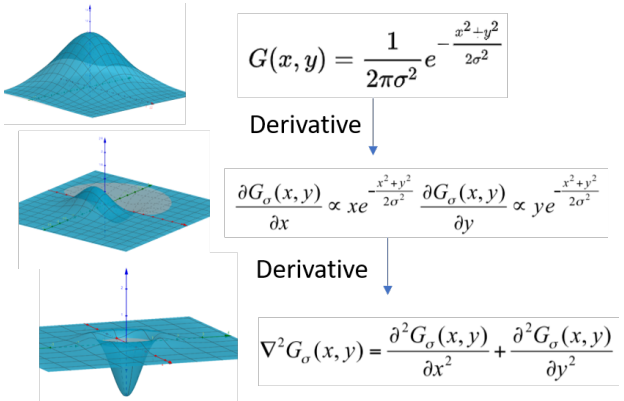

Figure 3. We illustrate the relationship between the Gaussian filter (top), DoG filter (middle) and LoG filter (bottom), which are related by their derivatives.

Because the derivative generates the filters and fractional calculus describes how to compute fractional derivatives, it is possible to interpolate between these filters and generate a general filter whose behavior ranges between the previously cited filters simply by changing a single parameter. With this in mind, we define a 2D *Fractional Filter* (FF) as follows:

$$F = AD^a D^b e^{-\frac{(x-x_o)^2+(y-y_o)^2}{\sigma^2}} \tag{14}$$

where a and b represent the fractional derivative order, A, σ, x_o,and y_o represent the parameters used to define the filter. With $A \in (-\infty, \infty)$, $\sigma \in (0, \infty)$, $x_o \in (-\infty, \infty)$, $y_o \in (-\infty, \infty)$ and $a \in (0, 2)$ and $b \in (0, 2)$. Additionally, by changing A, σ, x_o, and y_o, it is possible to generate many of the kernels generated by the convolutional filters in the convolutional layers of a CNN. This approach allows for the reduction of parameters in a convolutional filter from $N \times N$ (where N is the number of pixels in each dimension of the kernel) to only 6. This construct yields a large reduction in the parameters needed to train such a layer, but as a consequence, fractional filters can only approximate the equivalent CNN filter. Thus we expect a slight reduction in model accuracy when the traditional CNN layer is replaced by a layer of fractional filters.

4.2. Three-Dimensional Fractional Filters

We employ a similar approach to generalize fractional filters to higher, n-dimensional filters. In practice, three-dimensional convolutions are common to both video processing [38] and medical imaging [24] applications. Here, for example, a single convolutional kernel of dimension $5 \times 5 \times 5$ requires the specification of 125 independent parameters, while a 3D *fractional filter* can be defined using only 8 parameters. Similarly, in the case of a $7 \times 7 \times 7$ convolutional filter, the number of parameters can be reduced from 343 parameters to 8 generating almost $43X$ compression. We define a 3D FF:

$$F(x, y, z) = AD^a D^b D^c e^{-\frac{(x-x_o)^2+(y-y_o)^2+(z-z_o)^2}{\sigma^2}} \tag{15}$$

where (a, b, c) denotes the fractional derivative per axis,(x_o, y_o, z_o), respectively, and the centroid and scale filter A are all tuneable parameters. In summary, our 3D FF formulation requires the addition of two parameters to scale from 2D; see Figure 5.

5. Implementation

The general 2D fractional filter defined in the previous section comprises 6 independent parameters (Equation 14).

In practice, to efficiently compute this derivative, we take into account only the first 15 terms of the Taylor series:

$$D^a G(x) = \frac{A}{h^a} \sum_{n=0}^{15} \frac{\Gamma(a+1)G(x,y)}{(-1)^n \Gamma(n+1)\Gamma(1-n+a)} \quad (16)$$

where

$$G(x) = e^{-\frac{(x-x_o)^2}{\sigma^2}}, \quad (17)$$

is the Gaussian function, a is the fractional derivative order and $\Gamma(a)$ represents the gamma function, defined in Equation 6. The computation of the fractional derivative b of the filter in terms of y, $D^b G(y)$ is performed analogously. Finally, the fractional derivative of the two-dimensional *fractional kernel* is defined as the realization of $D^a D^b G(x,y) = D^a G(x) \times D^b G(y)$.

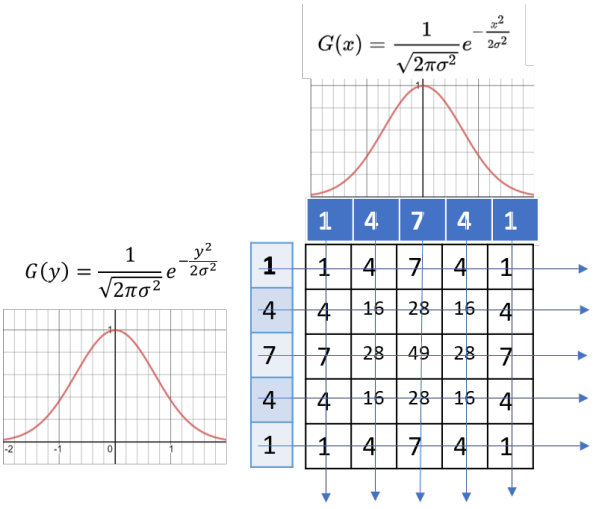

Figure 4. The computation of the N^2 elements of a filter are executed by the exterior product of the N elements on the x-axis and the N elements on the y-axis. The remaining values are calculated by multiplying these x and y vectors. By performing the evaluation of an exponential function $2N$ times instead N^2 times, computational performance is significantly improved. Such performance improvements are particularly beneficial for embedded systems (see Equation 16).

In the prior section we alluded to the computational and memory benefits of using the fractional filter, and the degree to which this savings increases with the respect to the size of the filter. For example, a 3×3 kernel will yield a reduction from 9 to 6 parameters for each traditional kernel replaced by a fractional kernel in a neural network layer. Likewise, the use of a fractional kernel in place of a traditional 5×5 or 7×7 kernel will result in a reduction from 25 and 49 parameters to just 6 parameters respectively.

During training, every round of backpropagation requires recomputing filter parameters, as well as the filter values to generate the mask for the next round. We simplify this computation as follows: the computation of the N^2 elements of a filter are executed by the exterior product of the N elements on the x-axis and the N elements on the y-axis. The filter values are calculated via elementwise multiplication of these x and y vectors (see Figure 4). By performing the evaluation of the exponential functions $2N$ times instead N^2 times, the computational performance of the training algorithm using FFs is significantly improved, as our approach eschews direct evaluation of N^2 Gaussians.

The same method can be used to compute 3D and higher order filters. Concretely, for 3D FFs, we generate three corresponding vectors for x, y and z as in equation 5, computing only $3N$ elements instead of N^3. For a 3D filter of dimension $5 \times 5 \times 5$, our method uses eight parameters instead of 125.

During inference, the fractional filter is initially calculated (just once, at the time of loading the network) from the stored parameters and then integrated into a CNN workflow just as with traditional kernels. This initial step requires the evaluation of the fractional derivative over the $N \times N$ parameters of the kernel (e.g. 25 elements in the case of a 5×5 filter). By contrast, during training, the the evaluation of the filters is required once per iteration, but we only need to compute $M \times N$ values per filter, as shown below.

With training efficiency in mind, we define the function $f_a(x,y)$:

$$f_a(x,y) = \frac{\Gamma(a+1)G(x,y)}{(-1)^n \Gamma(n+1)\Gamma(1-n+a)} \quad (18)$$

Because it is common to all of the training update steps, Equation 16 can be rewritten more concisely as:

$$D^a G(x) = \frac{A}{h^a} \sum_{n=0}^{15} f_a(x) \quad (19)$$

In the 2D case, we also compute the $D^b G(y)$, so that the (i,j) components of a FF are defined: $K(i,j) = D^a G(i) \cdot D^b G(j)$ for $1 \leq i,j \leq N$ (see Figure 4). Similarly, for the 3D case, we introduce an additional component axis $D^c G(z)$, so that each (i,j,k) element of a FF is defined: $K(i,j,k) = D^a G(i) \cdot D^b G(j) \cdot D^c G(k)$ for $1 \leq i,j,k \leq N$ (see Figure 5). In this manner, for the outer product of the vectors generated by the evaluation of $D^a G$, $D^b G$, and $D^c G$, we can define the kernel for one, two or three dimensions. This approach leads to a reduction in computation from N^M operations per filter to $N \times M$, where N is size of the filter and M denotes the filter dimension. For instance, instead of the explicit evaluation of the 343 components of a $7 \times 7 \times 7$ filter, our method requires only only 21 operations (see Figures 4 and 5 for a complete visualization).

In the case of Python implementations using Automatic Differentiation (AD) [44], training a network with FF is straightforward. However, in the absence of AD frameworks, one can use the following gradient-based training

rules for updating 2D FF parameters.

$$\Delta x_o = \frac{2A}{h^a}\sum_{n=0}^{15}(x - ih - x_o)f_a(x,y)$$

$$\Delta y_o = \frac{2A}{h^a}\sum_{n=0}^{15}(y - y_o)f_a(x,y)$$

$$\Delta A = \frac{1}{h^a}\sum_{n=0}^{15} f_a(x,y)$$

$$\Delta \sigma = \frac{2A}{h^a}\sum_{n=0}^{15}\frac{(x - nh - x_o)^2 + (y - y_o)^2}{\sigma^3} f_a(x,y)$$

$$\Delta a = \frac{A}{h^a}\sum_{n=0}^{15}\left[\Psi(a+1) - \Psi(a-i+1)\right] f_a(x,y)$$

$$\Delta b = \frac{A}{h^b}\sum_{n=0}^{15}\left[\Psi(b+1) - \Psi(b-i+1)\right] f_b(x,y) \quad (20)$$

where Ψ represents the digamma function.

6. Experimental Results

We conducted experiments using four different datasets to test and analyze our fractional filters: MNIST, CIFAR10,ImageNet, and UCF101; we provide details of these experimental results in this section.

6.1. MNIST

For MNIST classification, our baseline topology has six traditional 5×5 convolutional filters in the first layer, 10 traditional 5×5 convolutional filters in the second layer, 10 traditional 5×5 convolutional filters in the third layer and a final fully-connected layer with 160 inputs (10 4×4 activations) and 10 outputs for classification. The number of parameters per layer is: 150, 1500, 2500 and 1600, respectively, and in total we have 5750 parameters (ignoring biases).

It is clear that layer three has the largest number of parameters in the network, and therefore we hypothesize that this layer would benefit the most from the use of fractional kernels in terms of compression. Indeed, we show that we can reduce the number of parameters in this layer by a factor of 4 without sacrificing the accuracy of the network. After training our network using only 500 parameters in the third layer, instead of the original 2500 used in the baseline topology, the accuracy of the fractional filter network dropped from a baseline accuracy of 99.26% to 99.18%, a small reduction (0.08%), given the $5\times$ compression in the number of weights for the tested layer (results are shown in Table 1).

We further examined the accuracy and compression trade-off when using fractional kernels at various layers in the network; we found the greatest benefit in applying fractional filters to the layer(s) with the largest number of filters. In the particular case of MNIST, the baseline CNN has only six filters in the first layer with 150 trainable parameters. Using fractional filters results in a $5X$ reduction in required memory, reducing the number of parameters required to 30 (a reduction of 120 parameters would need to be trained and stored). But the trade-off for using the filter in the first layer is a 0.11% reduction in accuracy. Conversely, when we replace the third layer of the CNN with fractional filters, we reduce the number of trainable parameters in that layer from 2500 to 500 (a reduction of 2000 parameters), the accuracy trade-off is only 0.08%. Naturally, it is also possible to further compress the network by employing the fractional filters in additional layers. For example, when we use fractional filters in both layers 2 and 3, we observe a 0.36% reduction in accuracy, in exchange for eliminating 3500 parameters. This may be worthwhile depending on the application, particularly in an edge compute device. For this experiments we have a $5X$ compression, because we forced parameter $\sigma = 1$, then the filter used only 5 parameters total.

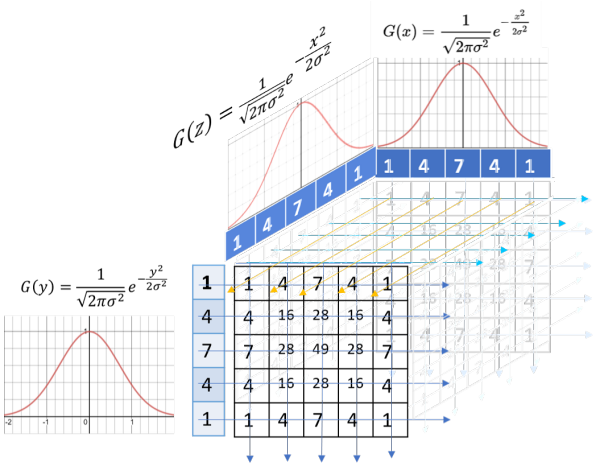

Figure 5. The computation of the N^3 elements of a filter are executed by the exterior product of the N elements on the x-axis and the N elements on the y-axis by N elements on the z-axis. The remaining values are calculated by multiplying these x, and y vectors. By performing the evaluation of required exponential functions $3N$ times instead N^3 times, computational performance is substantially improved.

6.2. Experiment 2: CIFAR-10

For our second set of experiments we used an augmented CIFAR-10 dataset with horizontal flipping, padding, and 32×32 random cropping during training.

One of our goals in the present work is to create a class of highly-efficient state-of-the-art models that can be deployed on embedded devices where memory is highly constrained.

Table 1. MNIST Accuracy vs Memory. Our Baseline CNN employs traditional 5×5 convolutional kernels in the following topology: (layer 1) 6 kernels, (layer 2) 10 kernels, (layer 3) 10 kernels (layer 4) fully connected layer with 10 outputs. The Fractional kernels have the same topology as the Baseline network, but we denote where a traditional kernel layer is replaced with a layer of fractional kernels.

Neural Network	#Parameters	accuracy
LeNet [3]	431K	99.4%
LetNet5 [34]	60K	99.24%
50-50-200-10NN [50]	226K	99.51%
Best Practices [45]	132.5K	99.5%
Baseline CNN	5.75K	99.26%
Fractional Filters (Layer 3)	**3.75K**	99.18%
Fractional Filters (Layer 1)	**5.63K**	99.15%
Fractional Filters (Layer 1&2)	**5.66K**	99.04%
Fractional Filters (Layer 2&3)	**2.25K**	98.9%

Thus we seek to generate the smallest model that can complete a task with the highest possible accuracy given space and resource limitations.

Our baseline topology for this dataset is the Resnet-18 architecture [16]. In order to highlight the impact of our approach, we first explored the effect of converting a single layer of traditional convolutional kernels in the network into a layer of fractional kernels. This baseline architecture is structured as follows: Layer 1 is comprised of 16 3×3 traditional convolutional kernels. Layer 1 feeds into 3 Resnet blocks each comprised of 2 stacks of $16 \times 32 \times 64$ traditional convolutional kernels. The final fully-connected layer outputs 10 nodes which correspond to the CIFAR object classification categories.

Table 2. CIFAR-10 Classification error vs Number of parameters

Neural Network	Depth	#Parameters	Error%
All-CNN [55]	9	1.3M	7.25
MobileNetV1 [20]	28	3.2M	10.76
MobileNetV2 [51]	54	2.24M	7.22
ShuffleNet 8G [62]	10	0.91M	7.71
ShuffleNet 1G [62]	10	0.24M	8.56
HENet [48]	9	0.7M	10.16
ResNet18 [16]	20	0.27M	8.75
Frac-ResNet18	20	**0.18M**	8.71

Table 2 illustrates how our results compare to prior related work, including some recent architectures that have been benchmarked against the CIFAR-10 pattern recognition problem. We limit our analysis to topologies that use less than 2.5M trainable parameters. As we noted in the prior section, we believe that fractional filters can potentially enable efficient implementations of deep learning models for use in embedded systems. Table 2 shows a summary of results from the smallest models recently reported for the CIFAR-10 classification problem.

In our experiments, different layers of the ResNet-18 topology were replaced to study the effect of the fractional filters. The notation in table 2 is as follows: **Frac-ResNet18** indicates that all the ResNet block were entirely comprised of fractional kernels, with the following topology: (16 traditional convolutional kernels (3x3),1 fractional ResNet block, 2 fractional ResNet blocks [16,32,64], and a fully connected layer (10).

Similar to our experimental results for MNIST, the CIFAR-10 results exhibit a degradation in accuracy when replacing all filters in the ResNet blocks of the network, but the accuracy drop is only 0.04%, while the induced parameter reduction 33% in this case. Naturally, if one replaces larger conventional filters with FFs, the compression yield can be improved further.

6.3. Experiment 3: ImageNet

The results shown in Table 3 reflect performance on the ImageNet dataset using a ResNet18 topology with conventional convolutional kernels in the first layer replaced with our proposed fractional kernels (64 fractional conv(7x7), 4 resnet blocks [64,128,256,512][2,2,2,2], and 1000 fully connected). These experiments on ImageNet yield a 0.6% accuracy drop in the top-1 error percentage result, compressing $8\times$ the number of parameters in such layer. because here the filters had 7×7 parameters, these 49 were replaced by only 6 parameters for each filter.

Table 3. ImageNet Accuracy vs Number of parameters

Neural Network	#L1 Parameters	Accuracy
ResNet18 [16]	9,408	69.7%
Frac-ResNet18 L1	**1,152**	69.17%

In this topology the rest of the filters are (3×3) having only 9 parameters each, the effort of reducing to 6 parameters will produce a very light compression on the model, but for instance the last ResNet block of the model has almost one million filters, then amount of removed parameters is about 3,670,016 with 0.5% additional accuracy drop.

6.4. Experiment 4: UCF101

UCF101 [54] is an action recognition dataset of realistic action videos, collected from YouTube, with 101 action categories. The action categories can be divided into five types: 1)Human-Object Interaction 2) Body-Motion Only 3) Human-Human Interaction 4) Playing Musical Instruments 5) Sports. With 13,320 videos from 101 action categories, UCF101 gives the largest diversity in terms of actions and with the presence of large variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions, etc.

From the videos we extract sequences of frames with a total of 29,479,886 frames for the training and testing, Our neural network architecture is based on the 3D-ResNet introduced in [29] and [57] which uses 3D CNN instead of 2D CNN, in order to capture spatio-temporal information from the video data. The original architecture is composed by 64 3D filters of ($7 \times 7 \times 7$), a pooling 3D layer of ($3 \times 3 \times 3$), four ResNet layers of 2 block each with 64, 128, 256 and 512 filters respectively each one with 3D filters of ($3 \times 3 \times 3$); every block performs batch normalization, and the network uses average pooling and a final fully- connected layer at output. The proposed fractional filters were used in the first convolutional layer replacing the 64 ($7 \times 7 \times 7$, i.e., 343 parameters per filter) filters with the fractional filters using only 8 parameters per filter. Our results in the in the table 4 shows a drop of 0.2% in accuracy due to this parameter compression. In the table 4 the number of parameters in the first layer is highlighted,in which the fractional filters were used. Making this layer in the ResNet18 the more efficient in terms of number of parameters vs accuracy.

Table 4. UCF101 Accuracy vs Number of parameters, Suffle & learn used cafeNet and OPN uses VGG-M-2048

Neural Network	#L1 Parameters	Accuracy.
Shuffle & Learn [23]	28,224	50.2%
OPN(80x80) [36]	49,392	59.8%
OPN(120x120) [36]	49,392	55.4%
OPN(224x224) [36]	49,392	51.9%
ResNet18 [16]	65,856	60.6%
Frac-ResNet18 L1	**1,536**	**60.4%**

6.5. Converting CNNs into FF-CNNs

Finally, we propose an efficient method to convert any conventional CNN into a FF-based CNN. Using a pre-trained CNN (e.g., ResNet) one can, in parallel, convert each conventional filter into a FF by solving the optimization problem (in the 2D case): $\text{argmin}_{a,b,x_0,y_0,\sigma} \sum_{(i,j)\in C}^{|C|}(F_{ij} - C_{ij})^2$, for each filter C in the CNN; where a, b, x_0, y_0, σ are the 2D FF parameters as referred to in Equation 14; C denotes the discrete conventional filter (e.g., 3×3), C_{ij} connote the corresponding filter values, and F_{ij} represent the filter values for the FF. In general, solving the aforementioned optimization problem is non-trivial. In addition, even individual, slight displacements from an optimal solution for a given filter approximation can accumulate across many filters in a model, leading to severe degradation of the FF-CNN performance.

We experimented with three different methods to solve the above optimization problem as a general procedure for converting conventional CNNs into FF-CNNs. In total, we compared the performance of Conjugate Gradient, the Simplex Method and Particle Swarm Optimization (PSO) [28] for approximating conventional filters with fractional filters. In our experiments, using 10,000 randomly generated filters, PSO significantly outperformed the other standard optimization techniques, yielding close to an order of magnitude smaller normalized error for filter approximation. When using the previously mentioned CNN architectures for the MNIST dataset, PSO converted the conventional CNN pre-trained model with 99.1% performance on MNIST to 98.3% (without any additional refinement training applied).

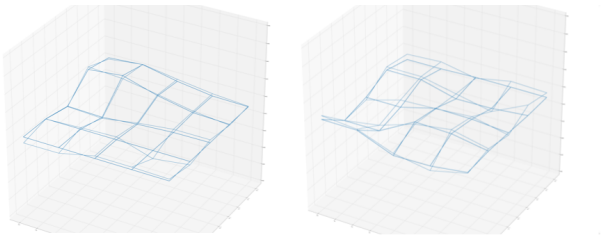

Figure 6. Two randomly selected examples of kernel approximation using the PSO algorithm for 5×5 fractional filters; the approximation is overlaid on top of the original filter in each image.

7. Conclusion

Fractional filters provide a novel solution for kernel compression in deep CNNs through the introduction of a reduced representation of a convolutional kernel in functional form. Importantly, this compression methodology admits of very favorable scaling attributes, as the number of FF parameters required for compression is essentially independent of the original kernel size.

Through experiments, we demonstrate the general effectiveness of FFs for filter compression across the MNIST, CIFAR-10, ImageNet, and UCF101 datasets. Notably, using FFs, we achieve a new record for the smallest model that can achieve greater than 99% accuracy performance on MNIST.

The reduction of CNN parameters using fractional filters can be enhanced considerably in the case of large kernel sizes. Concretely, our method achieves a 4X compression for 5×5 kernels, and 8X compression for 7×7 kernels, etc. This compression rate is larger still when using 3D filters (e.g., 15X for $5 \times 5 \times 5$ kernels and 42X for $7 \times 7 \times 7$ kernels), as shown in our experiments. To this end, fractional filters provide added value as an efficient means to compress CNNs across a variety of challenging, high fidelity use cases that benefit from the use of large kernel sizes, including pixel-level segmentation, medical imaging applications, and super resolution upsampling. Additional to the compression, the fractional filters allows the training of a NN using a pre-selected kernel size like 5×5 to get the parameters $a, b, x_0...$ and reuse them to define a 3×3 or a 7×7 without any retraining.

References

[1] Stegun I.A. Abramowitz M. Handbook of mathematical functions with formulas, graphs and mathematical tables. 10th ed:258–259, 1972.

[2] Andre Araujo, Wade Davenport Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019.

[3] BAIR/BVLC. Lenet architecture in caffe tutorial. *Github*, 2018.

[4] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016.

[5] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 379–387, Red Hook, NY, USA, 2016. Curran Associates Inc.

[6] Shantanu Das. *Functional Fractional Calculus*. Springer Publishing Company, Incorporated, 2nd edition, 2014.

[7] P. J. Davis. "leonhard euler's integral: A historical profile of the gamma function". *American Mathematical/ doi:10.2307/2309786*, Monthly. 66 (10):849–869, 2016.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks, 2018. cite arxiv:1803.03635.

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv: Learning*, 2019.

[11] R. Gonzalez and R. Woods. *Digital image processing*. Prentice Hall, 2008.

[12] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[13] Song Han. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 2015.

[14] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[15] 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.

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

[17] R. Herrmann. *Fractional Calculus*. World Scientific Publishing, 2011.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[19] Sverre Holm and Sven Peter Näsholm. A causal and fractional all-frequency wave equation for lossy media. *The Journal of the Acoustical Society of America*, 130(4):2195–2202, 2011.

[20] 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.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. In *Journal of Machine Learning Research*, 2018.

[22] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *ICLR2017*, 2017.

[23] Martial Hebert Ishan Misra, C. Lawrence Zitnick. Shuffle and learn: Unsupervised learning using temporal order verification. In *ECCV*, 2016.

[24] Kamal Jnawali, Mohammad R. Arbabshirani, Navalgund Rao, and Alpen A. Patel M.D. Deep 3D convolution neural network for CT brain hemorrhage classification. In Nicholas Petrick and Kensaku Mori, editors, *Medical Imaging 2018: Computer-Aided Diagnosis*, volume 10575, pages 307 – 313. International Society for Optics and Photonics, SPIE, 2018.

[25] Kukjin Kang, Michael Shelley, and Haim Sompolinsky. Mexican hats and pinwheels in visual cortex. *Proceedings of the National Academy of Sciences*, 100(5):2848–2853, 2003.

[26] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[27] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[28] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[29] Yutaka Satoh Kensho Hara, Hirokatsu Kataoka. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet. In *CVPR*, 2018.

[30] Amir Roshan Zamir Khurram Soomro and Mubarak Shah. Ucf101: A dataset of 101 human action classes from videos in the wild. In *CRCV*, 2012.

[31] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[32] 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.

[33] Nick Laskin. Fractional schrödinger equation. *Physical Review E*, 66(5):056108, 2002.

[34] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[35] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[36] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *ICCV*, 2017.

[37] Dongnan Liu, Donghao Zhang, Yang Song, Fan Zhang, Lauren J. O'Donnell, and Weidong Cai. 3d large kernel anisotropic network for brain tumor segmentation. In Long Cheng, Andrew Chi-Sing Leung, and Seiichi Ozawa, editors, *Neural Information Processing - 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part VII*, volume 11307 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2018.

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[39] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks, 2017. cite arxiv:1701.04128.

[40] H. Mhaskar and T. Poggio. Deep vs. shallow networks : An approximation theory perspective. *ArXiv*, abs/1608.03287, 2016.

[41] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[42] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, 2015.

[43] K.B. Oldham and J. Spanier. *The Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary Order*. Dover books on mathematics. Dover Publications, 2006.

[44] Adam Paszke, S. Gross, Soumith Chintala, G. Chanan, E. Yang, Zachary Devito, Zeming Lin, Alban Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[45] John C. Platt Patrice Y. Simard, Dave Steinkraus. Best practices for convolutional neural networks applied to visual document analysis. *ICDAR 2003*, 2003.

[46] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[47] I. Podlubny. *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications*. Mathematics in Science and Engineering. Elsevier Science, 1998.

[48] Ruixin Zhang Qiuyu Zhu. Henet: A highly efficient convolutional neural networks optimized for accuracy, speed and storage. *arXiv:1803.02742*, 2018.

[49] A. Radford. Improving language understanding by generative pre-training. 2018.

[50] Marc Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. *NIPS2006*, 2006.

[51] 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*, pages 4510–4520, 2018.

[52] George Seif and Dimitrios Androutsos. Large receptive field networks for high-scale image super-resolution, 2018.

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

[54] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[55] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[56] Emma Strubell, Ananya Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. *ArXiv*, abs/1906.02243, 2019.

[57] Andrew Zisserman Tengda Han, Weidi Xie. Video representation learning by dense predictive coding. In *ArXiv*, 2019.

[58] Bo Wang, Shuang Qiu, and Huiguang He. Dual encoding u-net for retinal vessel segmentation. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, page 84–92, Berlin, Heidelberg. Springer-Verlag.

[59] Stephen W Wheatcraft and Mark M Meerschaert. Fractional conservation of mass. *Advances in Water Resources*, 31(10):1377–1381, 2008.

[60] Julio Zamora Esquivel, Adan Cruz Vargas, Rodrigo Camacho Perez, Paulo Lopez Meyer, Hector Cordourier, and Omesh Tickoo. Adaptive activation functions using fractional calculus. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[61] Julio Zamora Esquivel, Adan Cruz Vargas, Paulo Lopez Meyer, and Omesh Tickoo. Adaptive convolutional kernels. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[62] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.