THE CONTINUOUS CNN: FROM TASK-SPECIFIC TO A UNIFIED CNN ARCHITECTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

Performant Convolutional Neural Network (CNN) architectures must be tailored to specific tasks in order to consider the length, resolution, and dimensionality of the input data. In this work, we tackle the need for problem-specific CNN architectures. We present the *Continuous Convolutional Neural Network* (CCNN): a single CNN able to process data of arbitrary resolution, dimensionality and length without any structural changes. Its key component are its *continuous convolutional kernels* which model long-range dependencies at every layer, and thus remove the need of current CNN architectures for task-dependent downsampling and depths. We showcase the generality of our CCNN by using the *same architecture* for tasks on sequential (1D), visual (2D) and point-cloud (3D) data. Our CCNN matches and often outperforms the current state-of-the-art across all tasks considered.

1 INTRODUCTION

The vast popularity of Convolutional Neural Networks (LeCun et al., 1998) (CNNs) is a result of their high performance and efficiency, which has led them to achieve state-of-the-art in applications across sequential (Abdel-Hamid et al., 2014; Van Den Oord et al., 2016), visual (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014) and high-dimensional data (Schütt et al., 2017; Wu et al., 2019). Nevertheless, a major limitation of CNNs –and other neural networks– is that their architectures must be tailored to particular applications in order to consider the length, resolution and dimensionality of the input data. This has led to a plethora of task-specific architectures (Oord et al., 2016; Bai et al., 2018; Simonyan & Zisserman, 2014; Szegedy et al., 2015; Ronneberger et al., 2015; He et al., 2016; Qi et al., 2017; Wu et al., 2019) which (*i*) hampers the selection of the most appropriate architecture for a particular task, and (*ii*) obscures the transfer and generalization of insights across applications. In this work, we tackle the need for problem-specific CNN architectures and propose a unified CNN architecture that can be used independent of the length, resolution and dimensionality of the data.

CNN architectures are data dependent. Current CNN architectures are task-specific because they are tied to the *length*, *resolution*, and *dimensionality* of the input. The *length* of the data varies from task to task, e.g. audio fragments may span milliseconds to minutes. This requires carefully chosen strides and pooling to capture relevant dependencies across the entire input (Van Den Oord et al., 2016; Lee et al., 2017). In addition, physical signals, e.g., audio, images, are continuous in nature. As such, their semantic meaning is independent of the *resolution* at which they are sampled, e.g.,

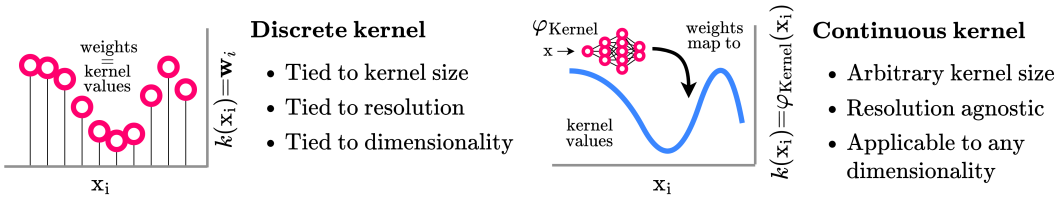
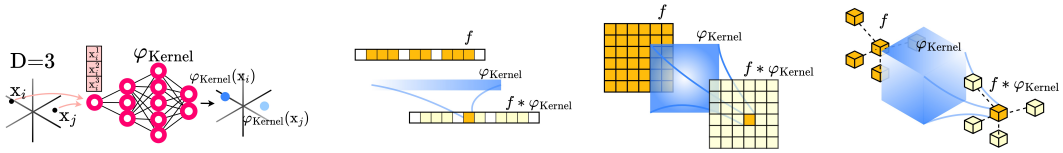


Figure 1: Discrete and continuous convolutional kernels. Discrete convolutional kernels assign a weight $\mathbf{w}_i$ out of a discrete set of weights $\mathbf{W}$ to a relative offset $\mathbf{x} - \tilde{\mathbf{x}}$. This ties the kernel to the length, resolution and dimensionality of the input, limiting the general applicability of the CNN architectures. Instead, our *Continuous Convolutional Neural Network* parameterizes kernel values as a continuous function φ_{Kernel} over the input domain $\mathbb{R}^d$, which decouples it from data characteristics.



(a) Kernel network. (b) D=1, sequences. (c) D=2, images. (d) D=3, volumes.

Figure 2: Continuous convolutional kernels: the key to a unified CNN architecture. The continuous parameterization of convolutional kernels used in this work consists of a small kernel network φ_{kernel} that receives coordinates as input and outputs the value of the convolutional kernel at that position (2a). By changing the dimensionality of the coordinates $\mathbf{x}_i$, the same kernel network can render convolutional kernels for sequential (2b), visual (2c), and higher dimensional data (2d).

the same audio may be expressed at different resolutions. Nevertheless, current CNN architectures are resolution-bounded and thus different resolutions require different CNNs. These limitations aggravate when considering data of different *dimensionalities*. Each input dimension can be defined at different lengths and resolutions, e.g., video, rectangular images and each data modality brings its own conventions for each of these properties, e.g., the resolution of a second of audio (16kHz) (Warden, 2018) strongly contrasts with that of images (32×32) (Krizhevsky et al., 2009).

Towards a unified CNN architecture. As discussed in Sec. 3, the core component that makes CNNs data-dependent are their *discrete convolutional kernels*. Convolutional kernels are implemented via a one-to-one mapping between kernel values and model parameters (Fig. 1 left). This (*i*) binds them to the input resolution and length, and (*ii*) makes them ill suited to model long-range dependencies. The latter results from the large number of parameters needed to construct large convolutional kernels. This is why standard CNNs favour using local kernels in combination with task-dependent depths and pooling layers to model long-range dependencies, at the cost of making them task-dependent.

The need for a continuous parameterization. To overcome task-dependent architectures, it is crucial to define a kernel parameterization that decouples parameter count from kernel size. Following Schütt et al. (2017); Romero et al. (2022b), we use a small neural network to define a continuous mapping from positions to the value of the kernel at those positions. The resulting *Continuous Convolutional Kernels* (Fig. 2), allow for the construction of convolutional kernels of arbitrary size in a parameter efficient manner. Consequently, the same kernel network –and thus the same convolutional layer– can be used regardless of the input length, resolution and dimensionality. We leverage this formulation of Continuous Kernels to introduce the *Continuous Convolutional Neural Network* (CCNN), a single network architecture that can be applied regardless of input characteristics.

Empirical results. To showcase the proposed CCNN, we deploy the same CCNN for several tasks on sequential (1D), visual (2D) and point-cloud (3D) data. Our CCNN matches and often outperforms the current state-of-the-art across all tasks considered. Importantly, the continuous parameterization of our CCNN allows it to handle irregularly sampled data natively. As a result, the CCNN is not restricted to grid data, e.g., 3D voxels, and can be used on point-clouds directly.

2 Related Work

Continuous convolutional kernels. A multitude of previous works investigate a formulation of CNNs based on continuous convolutional kernels pursuing a variety of motivations. A number of these works apply specifically to pointcloud data, interpolating a set of weights to obtain a definition over the continuous input space (Hua et al., 2018; Thomas et al., 2019), expressing the kernels in a polynomial (Xu et al., 2018) or neural network basis (Jia et al., 2016; Schütt et al., 2017; Wu et al., 2019). Other work focuses on exploiting desirable properties of spatially structured kernels on regular domains, for example to increase receptive fields efficiently (Su & Wen, 2021), learn receptive field sizes (Tomen et al., 2021), or implement equivariant convolutions (Sifre & Mallat, 2014; Worrall et al., 2017; Weiler et al., 2018; Weiler & Cesa, 2019; Bekkers, 2019; Finzi et al., 2020). In many works, these continuous-basis kernels are used as drop-in replacement in CNN architectures, hence the use of pooling operations in these architectures means they are still tied to data characteristics.

Modelling kernels through neural networks. Promising results were recently obtained by modelling the convolution kernel a small neural network that maps relative kernel positions to the kernel value

at those positions Jia et al. (2016); Schütt et al. (2017); Wu et al. (2019); Romero et al. (2022b), with their performance attributable to their flexibility in modelling expressive convolutional kernels. Importantly, works in the field of implicit neural representations showed the need for alleviating the spectral bias of ReLU MLPs when using them to model functions implicitly (Tancik et al., 2020; Sitzmann et al., 2020), e.g. through high-dimensional embeddings of input coordinates (Tancik et al., 2020), high-frequency activation functions (Sitzmann et al., 2020) or high-frequency inducing filterings of the input (Fathony et al., 2021). Romero et al. (2022b) transferred these results to the domain of continuous convolution kernels, and showed improvements in modelling such kernels when using high-frequency inducing activation functions (e.g. $\sin$). Romero et al. (2022a) developed Multiplicative Anisotropic Gabor Nets (MAGNets), a parameterization which allows control over the range of frequencies admitted by the parameterization, a desirably property to prevent aliasing in the convolutional kernels.

Romero et al. (2022b;a) show that by decoupling the size of the convolutional kernel from the number of parameters required to construct it, this approach allows for the construction of arbitrarily-sized kernels in a parameter efficient manner. Moreover, this parameterization overcomes the discrete nature of standard kernels and allows for the construction of resolution agnostic convolutional kernels that operate on coordinates of arbitrary resolution. Consequently, the same kernel network –and thus the same convolutional layer– can be used regardless of the input length and resolution. We leverage these observations to introduce an architecture that can be applied successfully over a range of tasks.

3 From Data-Dependent to Data-Independent CNN Architectures

In this section, we study the components of CNN architectures and pinpoint the changes required in order to construct a CNN architecture independent of input lengths, resolutions and dimensionalities.

3.1 Pointwise Operations: Linear Layers, Dropout, Pointwise Nonlinearities and Residual Connections

Pointwise operations are applied independently to each spatial element of the input signal (Fig. 3), e.g., pointwise linear layers, dropout and pointwise nonlinearities. As such pointwise operations do not depend on the specific shape of the input signal and regardless of the input length, resolution and dimensionality. Furthermore, learnable parameters of pointwise operations, e.g., in pointwise convolutional layers and Parametric ReLU (He et al., 2015), can be most generally expressed as a matrix of shape $\mathbf{W} \in \mathbb{R}^{\mathrm{N_{in}} \times \mathrm{N_{out}}}$. As the shape of $\mathbf{W}$ does not depend on the spatial shape of the input, the same number of parameters is required for inputs of any spatial shape.

Conclusion. Based on the previous observations, we conclude that pointwise operations can be used without changes in order to construct a unified CNN architecture.

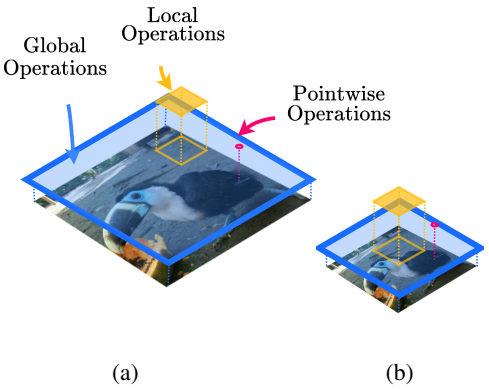


Figure 3: Three operation types that we distinguish between. Local operations are resolution dependent; transferring the same local operation (visualized in yellow) from Fig. 3a to a lower resolution Fig. 3b leads to an increased receptive field.

3.2 Global Operations: Normalization Layers, Dropout and Global Pooling

Global operations aggregate all spatial elements of the input signal for their processing (Fig. 3), e.g., normalization layers, dropout, global pooling. As such, common global operations do not depend on the specific shape of the input signal and their effect is equivalent regardless of the input length, resolution and dimensionality. It is important to note, however, that common global CNN operations only define learnable parameters along the channel axes e.g., the scale and mean parameters of a

normalization layer have shape $\mathbf{w} \in \mathbb{R}^{\mathrm{N_{in}}}$. Nevertheless, if one were to define a global operation for which spatial positions are also assigned weights, the shape of the parameters would be dependent on the input length, resolution and dimensionalty and the previous statement would no longer hold.

Conclusion. The previous observations indicate that global operations that only define channelwise learnable parameters can be used without changes in order to construct a unified CNN architecture. Luckily, this is the case for all common global operations used in CNN architectures.[1]

3.3 Local Operations: Convolutional Layers and Subsampling

Local operations are operations that are applied across the spatial dimensions of the input and that rely on spatial portions of the input (Fig. 3), e.g., convolution, subsampling. In practice, the sizes of the local neighbourhoods are selected based on the size of the input signal, e.g., 3×3 kernels or pooling over 2×2 regions. Unfortunately, if the resolution of the input signal changes then the effect of these operations will differ for each resolution: the portion of the input seen by the convolutional kernel will decrease and the size of the output signal after pooling would increase. Consequently, it is accurate to say that local operations *are dependent on the input length, resolution and dimensionality*.

A possible solution is to adjust the sizes of the operations proportional to the resolution changes. Nevertheless, the learnable parameters in (discrete) convolutional kernels are tied to spatial positions –a convolutional kernel of size 3×3 is implemented with 9 parameters (Sec. **??**)–. Consequently, increasing the size of the convolutional kernel is directly tied to a proportional increase in the number of parameters required to construct it. A better solution results from using a parameterization in which the number of parameters is *independent* from the size of the kernel. By doing so, the same parameterization can be used independently from the input length, resolution and dimensionality.

3.3.1 From Data-Dependent to Data-Independent Convolutional Layers

Conventional CNNs implement a discrete version of the convolution operation defined as:

$$(f * k)^o(\mathbf{x}) = \int_{\mathbb{R}^d} f(\tilde{\mathbf{x}}) k^o(\mathbf{x} - \tilde{\mathbf{x}}) \mathrm{d}\tilde{\mathbf{x}}, \quad o \in [1, ..., \mathrm{N_{out}}), \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}^{\mathrm{N_{in}}}$ is an input signal, and $k : \mathbb{R}^n \to \mathbb{R}^{\mathrm{N_{in}} \times \mathrm{N_{out}}}$ is a convolutional kernel. For the types of data CNNs are commonly used for, e.g. images, the signal f is generally sampled on a discrete grid of equidistant points, and thus it can be described as a function $f : \mathbb{Z}^d \to \mathbb{R}^{\mathrm{N_{in}}}$. Additionally, the signal f is, in practice, non-zero only on a finite subset of the grid $\Omega(f) \subset \mathbb{Z}^d$ with limits given by the range of sampling, e.g., height and width of an image. Accordingly, the convolutional kernel k is defined over the same grid of coordinates, of which generally a subset $\Omega(k) \subset \Omega(f)$ maps to nonzero values. Since both $\Omega(k)$ and $\Omega(f)$ are discrete and finite, the convolution can be implemented as the inner product of function and kernel values at each position:

$$(f * k)^o(\mathbf{x}) = \sum_{\tilde{\mathbf{x}} \in \Omega(k)} f(\tilde{\mathbf{x}}) k^o(\mathbf{x} - \tilde{\mathbf{x}}), \quad \mathbf{x} \in \Omega(f). \tag{2}$$

Note that the same discrete kernel k can be used at every location $\mathbf{x} \in \Omega(f)$ only because the input signal f is defined over an equidistant grid, and hence the discrete kernel k aligns with the features $f(\mathbf{x})$ at every location $\mathbf{x} \in \Omega(f)$.

Discrete convolutional kernels tie convolutional layers to data characteristics. Conventionally, the kernel function k is implemented through a discrete set of randomly initialized weights $\mathbf{W}$, of which each entry $\mathbf{w}_i \in \mathbb{R}^{\mathrm{N_{in}} \times \mathrm{N_{out}}}$ corresponds to a point $\mathbf{x}_i \in \Omega(k)$. This directly ties a model's parameter count to its kernel size. Consequently, an increased kernel size is reflected in a larger set $\Omega(k)$, and thus in a correspondingly larger weight matrix $\mathbf{W}$.

In order to incorporate the long-range spatial dependencies needed to extract high-level features in a parameter-efficient way, we must then resort to pooling operations and the stacking of layers that implicitly increase the receptive field of the kernel. This in turn, makes it so that a CNN can only be applied to data of a certain size: if we apply a network created to model long-range dependencies over images of size 256×256 to images of size 32×32, then intermediary pooling layers would make the spatial extent of the feature maps collapse long before all convolutional layers are applied.

[1] Global discrete convolutional kernels could be interpreted as global operations for which parameters along the spatial dimensions of the input are defined. Consequently, these require special treatment (Sec. **??**).

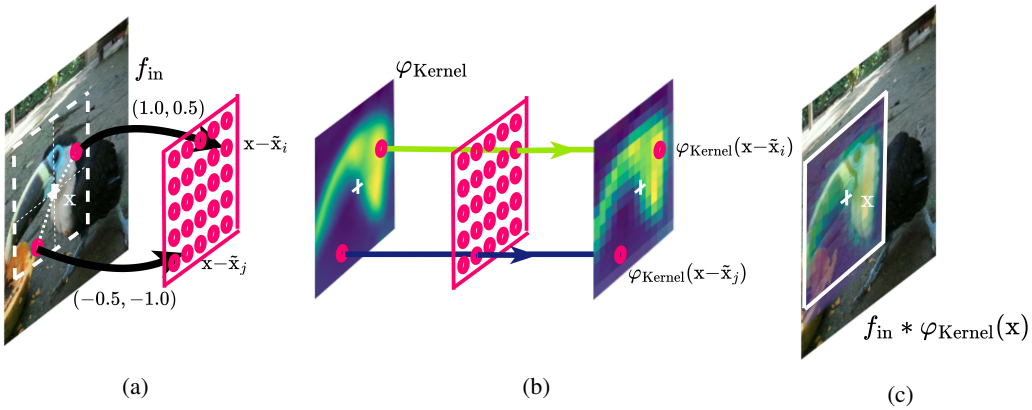


Figure 4: Applying a convolution with a Continuous Kernel to an image. Fig. 4a, given a pixel-position in the input image $\mathbf{x} \in \Omega(f)$, we obtain relative offsets to surrounding pixels $\{\mathbf{x} - \tilde{\mathbf{x}}\}_{\tilde{\mathbf{x}} \in \Omega(f)}$. Fig. 4b we sample the kernel network at each relative position ($\{\varphi_{\text{Kernel}}(\mathbf{x} - \tilde{\mathbf{x}})\}_{\tilde{\mathbf{x}} \in \Omega(f)}$) and Fig. 4c we apply a convolution with the sampled kernel to the input f_{in} at position $\mathbf{x}$.

Moreover, this discrete definition of the kernel k through a discrete set of weights $\mathbf{W}$ ties a model to a given input resolution. However, for many applications, the input signals f are actually a discrete sampling of an underlying continuous function. Therefore, we would like our model to be equivalent independent of the resolution at which f was obtained. With conventional CNNs, there is no trivial way to obtain equivalent kernel values on upsampled data, and hence, no way to apply a trained model directly. By the same reasoning, this method is ill-suited for irregularly sampled data. With weights corresponding to fixed relative positions an infinite number of weights would be needed to cover any continuous domain.

These limitations suggest a better approach; using the model weights to parameterize k as a continuous function over the data domain $\mathbb{R}^d$.

Continuous convolutional kernels: a data independent parameterization. To obtain a formulation for a CNN applicable to arbitrary resolutions and sizes, we require a convolutional layer that is approximatelyinvariant to the set $\Omega(f)$ over which f is sampled. To this end we require a definition of the kernel function k over the continuous domain $\mathbb{R}^d$ instead of $\mathbb{Z}^d$. In order to have this architecture support inputs of arbitrary size, we furthermore require this definition of k to decouple parameter count from the size of the kernel. Since we do not assume our data lies on a fixed equidistant grid, we have $\Delta\tilde{\mathbf{x}} \neq 1$, and obtain:

$$(f * k)^o(\mathbf{x}) = \sum_{\tilde{\mathbf{x}} \in \Omega(f)} f(\tilde{\mathbf{x}}) \varphi^o_{\text{Kernel}}(\mathbf{x} - \tilde{\mathbf{x}}) \Delta\tilde{\mathbf{x}}. \tag{3}$$

The term $\Delta\tilde{\mathbf{x}}$ intuitively corresponds to the inverse density of $\Omega(f)$. For regular grids, this results in a term $\frac{V}{|\Omega(f)|}$, with V the volume integrated over and $|\Omega(f)|$ the number of points (Finzi et al., 2020).

We use *Continuous Kernels* (Romero et al., 2022b;a) which provide a continuous parameterization for a convolutional kernel by using a small neural network $\varphi_{\text{Kernel}} : \mathbb{R}^{\text{D}} \to \mathbb{R}^{\text{N}_{\text{out}} \times \text{N}_{\text{in}}}$ as kernel function. This network maps locations in the domain of the kernel $\mathbf{x}_i \in \mathbb{R}^{\text{D}}$ to the values of a set of N_{out} convolutional kernels at that position: $k(\mathbf{x}) \in \mathbb{R}^{\text{N}_{\text{out}} \times \text{N}_{\text{in}}}$, see Fig. 1. We give an example of applying Continuous Kernels to images in Fig. 4

The Continuous Kernel convolutional layer allows for construction of an arbitrarily large kernel without increasing the overall parameter count of the convolutional layer, since the number of model parameters in the kernel function φ_{Kernel} is independent from the number of points $\mathbf{x}_i$ in a neighbourhood that determines the kernel's size. Romero et al. (2022b) shows that such large-scale kernels allow for direct incorporation of long-range spatial dependencies, removing the need for downsampling and stacking of layers to increase receptive fields. This in turn allows us to build an architecture which does not contain resolution-, dimensionality-, and size-dependent layers.

4 THE CONTINUOUS CONVOLUTIONAL NEURAL NETWORK

Previous works (e.g. (Romero et al., 2022b;a)) proposed architectures that fulfil the requirements posited in Sec. 3. However, experiments show that these architectures do not transfer between domains and still require tailoring for specific applications. To obtain an architecture performant on a wide variety of domains, we propose a number of structural improvements to Continuous Convolutional Layers as they are used in Romero et al. (2022a), and address training issues by proposing regularization for continuously defined kernels. Subsequently, we propose a single architecture we apply to a range of tasks.

Separable Continuous Convolutions. Separable convolutions are increasingly used in deep learning architectures to improve parameter- and computational efficiency (Rigamonti et al., 2013; Sifre & Mallat, 2014; Szegedy et al., 2015; Chollet, 2017). Based on observations of the improvements that depthwise separable convolutions yield in CNNs (Chollet, 2017; Tan & Le, 2019; Knigge et al., 2021; Liu et al., 2022), we construct a depth-wise separable version of the Continuous Kernel, in which a spatial convolution is computed with a kernel generated by a kernel network $\varphi_{\text{Kernel}} : \mathbb{R}^{\text{D}} \rightarrow \mathbb{R}^{\text{N}_{\text{in}}}$, followed by a spatially point-wise convolution from N_{in} to N_{out}.

Proper initialization of φ_{Kernel}. We observe that the networks used in previous works as the kernel functions k, are not properly initialized for their purpose of parameterizing convolutional kernels (Schütt et al., 2017; Wu et al., 2019; Romero et al., 2022b;a). To ensure training stability one would like to retain constant variance throughout the activations in the network. To this end, the discrete weights $\mathbf{W}$ of a convolutional kernel k are generally initialized with a variance inversely proportional to the volume (number of pixels $|\Omega(k)|$ and channels N_{in}) integrated over in the convolution operation (He et al., 2015): $\text{Var}(\mathbf{W})$=$g^2/(\text{N}_{\text{in}} \cdot |\Omega(k)|)$, with a g that depends on the nonlinearity used. Nevertheless, in related works the hypernetworks φ_{Kernel} used to parameterize the kernel k are themselves initialized such that a unitary variance throughout is preserved. Consequently, when used as a kernel network, a standard initialization method leads the kernel to have unitary variance, i.e., $\text{Var}(k)$=1. As a result, CNNs using neural networks as kernels experience a layer-wise growth in the variance of the feature maps proportional to $\text{N}_{\text{in}} \cdot |\Omega(\text{k})|$. E.g., we observe that the logits in CKCNNs (Romero et al., 2022b) and FlexNets (Romero et al., 2022a) lie in the order of 1e^{19} upon initialization. This is undesirable; it leads to unstable training and a need for low learning rates.

To address this issue, we require that the variance at the output of φ_{Kernel} equals $\text{g}^2/(\text{N}_{\text{in}} \cdot |\Omega(\text{k})|)$ and not 1. Inspired by Chang et al. (2020), we therefore re-weight the last linear layer of the kernel network φ_{Kernel} by $g^2/\sqrt{\text{N}_{\text{in}} \cdot |\Omega(\text{k})|}$. In our separable convolutions with N_{in} = 1 it reduces to $g^2/\sqrt{|\Omega(k)|}$. The variance of continuous kernel values then follows the conventional initialization of discrete convolutional kernels, and the logits of CCNNs present unitary variance upon initialization.

Regularization. Weight decay as regularization method for CNNs penalizes the L_2 norm of learned kernel values k by adding a loss term minimizing the corresponding weight values $\mathbf{W}$ to prevent overfitting on specific input patterns Krogh & Hertz (1991). With Continuous Kernels, the set of weights $\mathbf{W}$ we are optimizing for do not correspond to these kernel values directly. Since we still want to prevent the network from overfitting on specific input signals, we amend the loss term to reflect this, by penalizing the sampled kernel values k instead of the network weights $\mathbf{W}$, implicitly regularizing the weights of the kernel function φ_{Kernel}. Given $\varphi_{\text{Kernel}}^{:,l}$ the set of kernel values for all channels sampled at a layer l, $\mathcal{L}_{\text{obj}}$ the objective loss function, λ the regularizing parameter:

$$\mathcal{L} = \mathcal{L}_{\text{obj}} + \frac{1}{2}\lambda \sum_{l=0}^{L} ||\varphi_{\text{Kernel}}^{:,l}||^2. \tag{4}$$

Learning of hyperparameters. A promising property of Continuous Kernels is that they enable the learning of parameters that must otherwise be treated as hyperparameters in CNNs with discrete kernels. FlexConvs Romero et al. (2022a), for instance, define their convolutional kernels as the product of a kernel network φ_{kernel} and a trimmed Gaussian mask $\text{Gauss}_{\mu,\sigma}$ with learnable parameters μ, σ, i.e., $\mathbf{K}(\mathbf{c}_i)$=$\varphi_{\text{kernel}}(\mathbf{c}_i) \cdot \text{Gauss}_{\mu,\sigma}(\mathbf{c}_i)$. The Gaussian mask defines the size of the kernel and thus, by learning the mask, one can effectively learn the size of the convolutional kernel during training. We employ the FlexConv approach for learning kernel sizes in our experiments.

A refactored residual block. Residual connections (He et al., 2016) are widely used in CNN architectures since they have been shown to provide training stability and improved performance. The operation of a residual block $\mathcal{R}$ is defined as: $\mathcal{R}(f) = a(\psi(f) + f)$, with a any activation function,

Table 1: Experimental results on a range of sequence, image and point-cloud datasets. × - *unable to apply this model, either because of the model's inability to handle the specified data, or because of computational complexity issues. N.A. - not available.*

		Sequence						Image			Point-cloud
	Size	sMNIST	pMNIST	sCIFAR10	Speech-MFCC	Speech-Raw	Speech-0.5x	CIFAR10	CIFAR100	STL10	ModelNet40
Transformer	500K	98.90	97.90	62.20	90.75	×	×	×	×	×	×
LSSL	7.8M	99.53	98.76	84.65	×	×	×	×	×	×	×
S4	7.8M	99.63	98.70	91.13	×	×	×	×	×	×	×
LSSL	300k	N.A.	N.A.	N.A.	93.58	×	×	×	×	×	×
S4	300k	N.A.	N.A.	N.A.	93.96	98.32	96.30	×	×	×	×
CKCNN-Seq	98K	99.31	98.00	62.25	95.30	71.66	65.96	×	×	×	×
CKCNN-Seq-Big	1M	99.32	98.54	63.74	×	×	×	×	×	×	×
FlexTCN-6	375K	99.62	98.63	80.82	97.67	91.73	×	×	×	×	×
ResNet-44	660K	×	×	×	×	×	×	92.90	71.15	NA	×
ResNet-18	11.2M	×	×	×	×	×	×	94.92	**77.50**	81.04	×
Parabolic CNN	502K	×	×	×	×	×	×	88.5	64.8	77.0	×
Hamiltonian CNN	264K	×	×	×	×	×	×	89.3	64.9	78.3	×
CKCNN	630K	×	×	×	×	×	×	86.8	N.A.	N.A.	×
FlexNet-6	670K	×	×	×	×	×	×	92.2	N.A.	N.A.	×
SpiderCNN	N.A.	×	×	×	×	×	×	77.97	N.A.	N.A.	92.4
PointConv	N.A.	×	×	×	×	×	×	89.13	N.A.	N.A.	92.5
DGCNN	1.84M	×	×	×	×	×	×	×	×	×	89.0
PointNet	3.48M	×	×	×	×	×	×	×	×	×	89.2
PointNet++	1.99M	×	×	×	×	×	×	×	×	×	90.0
RepSurf-U	1.48M	×	×	×	×	×	×	×	×	×	**94.7**
$\text{CCNN}_{4,110}$	200K	**99.72**	98.82	90.30	95.01	98.34	96.22	92.78	66.86	81.80	84.44
$\text{CCNN}_{6,380}$	2M	**99.72**	**98.84**	**93.08**	**97.98**	**98.44**	**96.44**	**95.20**	73.16	**83.00**	85.70

and ψ any combination of convolutional layers, activations and normalizations. A simple pointwise addition of f with $\psi(f)$ is possible in case they are defined over the same spatial domain Ω.

For the specific architecture of our residual block $\mathcal{R}$, we can look to recent work showcasing how performance of networks using residual blocks can be strongly improved by changing the nonlinearities used and the position of the normalization layers within the blocks Xiong et al. (2020); Liu et al. (2022). Based on these observations, we modify the FlexNet architecture (Romero et al., 2022a) with a residual network composed of blocks similar to those of S4 networks (Gu et al., 2022). The CCNN architecture is shown in Figure 9.

5 Experiments

Our goal is to define a single architecture specification that can be applied over a range tasks, regardless of the specific data characteristics for that task. To this end, we implement a CCNN model that we validate on a range of datasets. We validate the architecture on several sequential (1D), visual (2D) and pointcloud (3D) benchmark datasets. We show that the *exact same* architecture obtains state-of-the-art results on a number of sequential tasks, competitive performance on image tasks, and is a parameter-efficient point-cloud processing method. Furthermore, we show that the learned kernel functions generalize from regular to irregular data by transferring a model from voxels to pointclouds.

Architecture specifications. To determine the parameter efficiency of our model, we create two CCNNs with differing numbers of hidden layers sizes: $\text{CCNN}_{4,110}$ (4 blocks, 110 channels) and $\text{CCNN}_{6,380}$ (6 blocks, 380 channels). The kernel network φ_{Kernel} is chosen to be a 3-layer MAGNet (Romero et al., 2022a), with 32 hidden units for the $\text{CCNN}_{4,140}$ models, and 64 hidden units for the larger $\text{CCNN}_{6,380}$ models. In each task, the input dimension of the kernel network φ_{Kernel} corresponds to the dimensionality of the data (e.g. 1 for sequence data, 2 for image data, 3 for point-clouds). Since we are using depthwise separable convolutions, the kernel network maps to a set of kernel values $\mathbb{R}^{\text{N}_{\text{in}}}$. Details on hyperparameters and training regime may be found in Appx. B.

CCNN on sequences, images and point-clouds. (Tab. 1) We show that our model performs well over a range of domains, without any architectural changes, by applying to a range of sequence, image and point-cloud tasks. See Appx. A for a detailed description of each dataset.

We consider Sequential and Permuted MNIST (LeCun et al., 1998) and Sequential CIFAR10 (Krizhevsky et al., 2009). In each of these datasets the original 2D structure of images has been discarded by organizing each image into a sequence. Because of the large-scale convolutional kernels which are able to model the long-range dependencies contained in these datasets directly in a parameter-efficient manner, CCNN is able to surpass current SOTA.

Next, CCNN is applied to multiple versions of the Speech Commands dataset (Warden, 2018), a dataset of short high-resolution audio recordings. Speech-Raw contains the unprocessed raw audio at 16kHz, Speech-MFCC a lower resolution mel-frequency cepstrum embedding (MFCC) of the data, and Speech-0.5x a subsampling of the original signal. We note that our model can be applied to

Table 2: Experimental results on tasks in the Long Range Arena dataset. × - *unable to apply this model, either because of the model's inability to handle the specified data.*

	LISTOPS	TEXT	IMAGE	PATHFINDER	AVG.	2DIMAGE	2DPATHFINDER
Transformer	36.37	64.27	42.44	71.40	53.66	×	×
Reformer	37.27	56.10	38.07	68.50	50.56	×	×
BigBird	36.05	64.02	40.83	74.87	57.17	×	×
Linear Trans.	16.13	65.90	42.34	75.30	50.46	×	×
Performer	18.01	65.40	42.77	77.05	51.18	×	×
FNet	35.33	65.11	38.67	77.80	54.52	×	×
Nystromförmer	37.15	65.52	41.58	70.94	57.46	×	×
Luna-256	37.25	64.57	47.38	77.72	59.37	×	×
S4	**58.35**	76.02	87.26	86.05	**80.48**	×	×
$\text{CCNN}_{4,110}$	44.85	83.59	87.62	91.36	76.86	89.48	94.80
$\text{CCNN}_{6,380}$	43.60	**84.08**	**88.90**	**91.51**	77.02	**91.12**	**96.00**

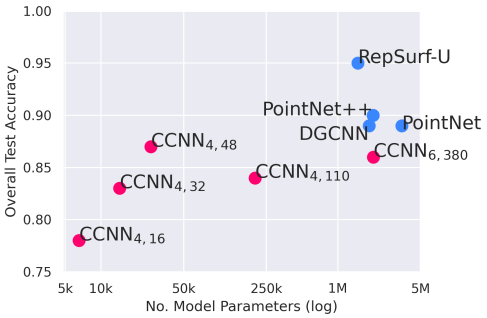


Figure 5: Parameter count versus performance on ModelNet40. Very small CCNN models obtain remarkable results.

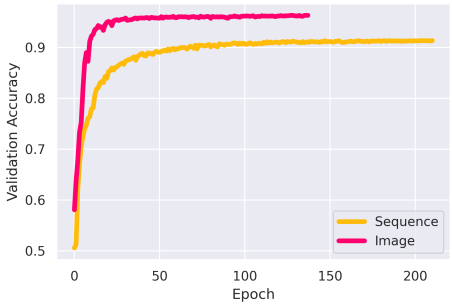


Figure 6: Performance of the $\text{CCNN}_{6,380}$ on the LRA Image task, where samples are either input as sequence or with the original image structure.

each of these data specifications directly, without any modifications, and obtains SOTA in each case, outperforming a range of sequence-specific processing architectures.

On 2D image data, the same CCNN model outperforms other spatially-continuous convolutional vision architectures ((Ruthotto & Haber, 2020),(Tomen et al., 2021), (Romero et al., 2022b), (Romero et al., 2022a)), as well as performing competitively with other large scale models.

To assess performance of the CCNN on irregularly sampled higher dimensional data, we validate on the ModelNet40 dataset (Wu et al., 2015), a dataset consisting of 3D meshes of objects adhering to a range of classes. Following (Wu et al., 2019), we sample a pointcloud from the faces of this mesh, along with the normal vectors of these faces at the respective locations. Due to the large computational complexity of our method on irregular data, we limit the number of sampled points to 512. For more details on applying our model to ModelNet40. On point-cloud data, although performance is not on par with current state of the art (we notice strong overfitting on the training set), CCNN is able to achieve a decent overall classification score with a relatively small number of parameters (200k). Important to note here is that all models compared to are point-cloud specific architectures. Although some of these may be applied to regular data as well, transferring them is non-trivial, and many of these architectures contain techniques and operations specific to point-clouds. We hypothesize that performance on point-clouds may be lacking due to the relative sparsity of this task. Continuous Kernels define a kernel function over the entire continuous domain $\mathbb{R}^3$, even though the number of kernel values sampled over this domain is sparse compared to image data that lies on a dense grid. This leads to sparse supervision for the kernel, making it hard to learn a kernel function that generalizes to unseen data, as this exposes the kernel function to aliasing.

CCNN obtains SOTA on long-range sequence tasks. (Tab. 1, 2) We show that architectures with large-scale continuous convolution kernels can be competitive in tasks with extremely long-range dependencies, by applying to Long-Range Arena (Tay et al., 2021), a collection of datasets with focus on long-range dependencies. We evaluate on a subset of tasks (see Tab. 2), and show remarkable performance, obtaining SOTA on multiple of these tasks. We attribute this to the CCNNs' ability to model long-range dependencies directly and efficiently via the flexible kernel parameterization.

CCNN for parameter efficient pointcloud processing. (Fig. 5) To further investigate the severe overfitting of CCNN on ModelNet40, and to assess overall parameter efficiency of the CCNN

architecture, we train an additional set of models of varying parameter sizes, (see Appx. B.3), and graph model performance versus parameter count for our CCNN on ModelNet40 compared with other point-cloud specific methods. Very small CCNN models achieve remarkable performance in terms of overall accuracy on modelnet40, even outperforming the 200k and 2M models. This shows that the Continuous Kernel paradigm is extremely parameter-efficient at smaller scales. However, this result also clearly highlights a current drawback of our approach: its inefficiency to scale with larger numbers of parameters. This is also seen in previous experiments: larger models do not necessarily improve performance on sequence and image tasks. We treat this result more thoroughly in Sec. 6.

CCNN transfers from regular to irregular data. Our aim was to create a model architecture that is decoupled from data-specific characterstics. Up to this point, our experiments only showed that it is possible to *train* a CCNN architecture for a range of domains and tasks. We are furthermore interested in examining how well our model is able to transfer performance over domains. To investigate, we assess how performance of a CCNN trained on densely sampled regular data transfers to the sparse, continuous domain. We train the smaller variant of our CCNN model on a version of ModelNet10 voxelized to a grid of $40 \times 40 \times 40$, using only the voxel-locations as input to the model. We then transfer this model to a dataset of point-clouds sampled from the continuous meshes that underly the ModelNet10 dataset. After training on voxelized data, CCNN obtains an overal accuracy of 90.99 on the voxelized test set. Applying the trained model to a test set of point-clouds instead, we obtain an overall accuracy of 90.64, showing that performance does not degrade severely when transferring a model from regular data to continuously sampled data. This leads us to conclude that the Continuous Kernel functions learned from regularly sampled data indeed correctly reflect the underlying continuous nature of the input data, as these learned kernel functions successfully generalize to inputs defined over the continuous domain. More details on the setup of this experiment, and visualizations of both datatypes can be found in Appx. A.

The importance of reflecting data characteristics. (Fig. 6) In principle, we could consider all image tasks as a sequential task in which no 2D structure is considered. This is done, for instance with S4 (Gu et al., 2022) due to the complexity of defining high-dimensional state spaces. Nevertheless, this comes at the cost of discarding important structural information. Contrarily, CCNNs can be easily defined on multidimensional spaces simply by changing the dimension of the input coordinates of the kernel networks. Perhaps not surprisingly, we observe that by explicit incorporation of 2D structure of Image and Pathfinder tasks in the LRA benchmark, much better results can be obtained (Tab. 2, right). In PathFinder with 2D images, our largest CCNN obtains an accuracy of 96.00, outperforming the previous state of the art by a margin of almost 10% points and performing remarkably better than the CCNN on flattened images. Additionally, we observe that models trained on the 2D data converge faster than their sequential counterparts (Fig. 6). We note that 2D CNNs with small convolutional kernels, e.g., ResNet-18, were unable to solve Pathfinder due to the lack of fine-grained global context modelling resulting from intermediate pooling layers. This was also seen by Gu et al. (2020).

6 Limitations and Conclusion

Parameter scaling. Although we note in Fig. 5 that the CCNN performs remarkably well with a very small number of parameters, throughout our experimental results we observe that the current CCNN formulation lacks in parameter scaling capabilities; increased parameter counts yield strongly diminishing returns. In a number of settings, we observed strong overfitting to the training set, which may indicate the need for stronger regularization. We hypothesize aliasing effects in the kernel function may furthermore impede performance on irregularly sampled data, which implies a more rigorous approach to ensuring kernel smoothness is needed in these settings.

Computational efficiency. A current drawback of the large-scale Continuous Kernels is their large computational cost. This currently prevents the CCNN from being deployed on large-scale data: experiments on ModelNet40, required limiting kernel size to ensure computational feasibility. An important avenue of future research may look into softening this computational requirement, e.g. by means of more informed sampling procedures to estimate the convolution operation.

Conclusion and outlook. To address the need for task-specific architectural configuration, we introduce the *Continuous Convolutional Neural Network* (CCNN): a unified architecture decoupled of resolution, length and dimensionality. We show its validity on a range of domains, obtaining state-of-the-art performance on multiple long-range sequence tasks, performing competitively on image data, and proving a highly parameter-efficient approach to point-cloud processing.

REFERENCES

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Erik J Bekkers. B-spline cnns on lie groups. *arXiv preprint arXiv:1909.12057*, 2019.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Oscar Chang, Lampros Flokas, and Hod Lipson. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1lma24tPB`.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

William Falcon et al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=OmtmcPkkhT`.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, pp. 3165–3176. PMLR, 2020.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33: 1474–1487, 2020.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=uYLFoz1vlAC`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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, 2016.

Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 984–993, 2018.

Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in neural information processing systems*, 29, 2016.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

David M Knigge, David W Romero, and Erik J Bekkers. Exploiting redundancy: Separable group convolutional networks on lie groups. *arXiv preprint arXiv:2110.13059*, 2021.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level cnn architectures. *arXiv preprint arXiv:1712.00866*, 2017.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2754–2761, 2013.

David W. Romero, Robert-Jan Bruintjes, Jakub Mikolaj Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *International Conference on Learning Representations*, 2022a. URL `https://openreview.net/forum?id=3jooF27-0Wy`.

David W. Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. CKConv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2022b. URL `https://openreview.net/forum?id=8FhxBtXSl0`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

Laurent Sifre and Stéphane Mallat. Rigid-motion scattering for texture classification. *arXiv preprint arXiv:1403.1687*, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

Bing Su and Ji-Rong Wen. Log-polar space convolution for convolutional neural networks. *arXiv preprint arXiv:2107.11943*, 2021.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6411–6420, 2019.

Nergis Tomen, Silvia-Laura Pintea, and Jan Van Gemert. Deep continuous networks. In *International Conference on Machine Learning*, pp. 10324–10335. PMLR, 2021.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125:2, 2016.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

Maurice Weiler and Gabriele Cesa. General e (2)-equivariant steerable cnns. *Advances in Neural Information Processing Systems*, 32, 2019.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *Advances in Neural Information Processing Systems*, 31, 2018.

Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5028–5037, 2017.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9621–9630, 2019.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020.

Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 87–102, 2018.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.

APPENDIX
TOWARDS A GENERAL PURPOSE CNN FOR LONG RANGE DEPENDENCIES IN ND

A DATASET DESCRIPTION

Sequential and Permuted MNIST. The MNIST dataset LeCun et al. (1998) consists of 70K gray-scale 28×28 handwritten digits divided into training validation and test sets of 60K and 10K samples, respectively. For validation purposes, the training dataset is further divided into training and validation sets of 55K and 5K samples, respectively.

The sequential MNIST dataset (sMNIST) presents MNIST images as a sequence of 784 pixels for digit classification. Consequently, good predictions require the model to preserve long-term dependencies up to 784 steps in the past. The permuted MNIST dataset (pMNIST) incorporates an additional level of difficulty by permuting the order of all sMNIST sequences with a random permutation. Resultantly, models can no longer rely on local information for the construction of their features and the importance of modelling long-term dependencies becomes more pronounced.

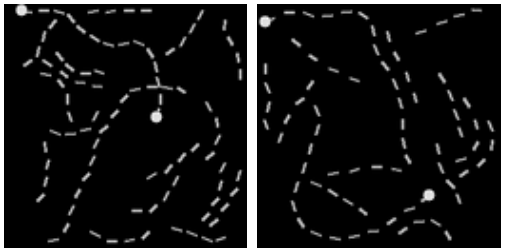
CIFAR10, CIFAR100 and Sequential CIFAR10. The CIFAR10 dataset Krizhevsky et al. (2009) consists of 60K real-world 32×32 RGB images uniformly drawn from 10 classes divided into training and test sets of 50K and 10K samples, respectively. The CIFAR100 dataset Krizhevsky et al. (2009) is similar to the CIFAR10 dataset, with the difference that the images are now uniformly drawn from 100 different classes. For validation purposes, the training dataset of both CIFAR10 and CIFAR100 are further divided into training and validation sets of 45K and 5K samples, respectively.

Analogously to the sMNIST dataset, the sequential CIFAR10 (sCIFAR10) dataset presents CIFAR10 images as a sequence of 1024 pixels for image classification. This dataset is more difficult than sMNIST, as *(i)* larger memory horizons are required to successfully solve the task, and *(ii)* more complex structures and intra-class variations are present in the images.

Speech Commands. The Speech Commands dataset Warden (2018) consists of 105809 one-second audio recordings of 35 spoken words sampled at 16kHz. Following Kidger et al. (2020), we extract 34975 recordings from ten spoken words to construct a balanced classification problem. We refer to this dataset as *Raw Speech Commands*. In addition, we use the preprocessing steps of Kidger et al. (2020) and extract mel-frequency cepstrum coefficients from the raw data. The resulting dataset, referred to as *MFCC Speech Commands*, consists of time series of length 161 and 20 channels.

Long Range Arena. The Long Range Arena benchmark (Tay et al., 2021) consists of 6 tasks with lengths 1K-16K steps encompassing modalities and objectives that require similarity, structural, and visuospatial reasoning. The `Pathfinder`, `Path-X` and `Image` tasks are similar in nature to the sMNIST and sCIFAR10 tasks. These tasks consists of classification tasks performed on images that are treated as sequences.

The `Image` task corresponds to the sequential CIFAR10 dataset with the only difference that the CIFAR10 images are treated as gray-scale images. The `Pathfinder` and `Path-X` tasks are binary tasks in which binary images are provided and the model must predict whether the two points in the images are connected with a line or not –see Fig. 8 for an example–. The difference between both datasets is their resolution. Whereas `Pathfinder` has images of size 32×32, `Path-X` has images of size 128×128. It is important to mention that these tasks are so difficult that even if treated as 2D signals, CNNs without global receptive fields are unable to solve them (Gu et al., 2022).



(a) (b)

Figure 7: Positive and negative samples from the `Path-X` dataset

STL-10. The STL-10 dataset (Coates et al., 2011) is a subset of the ImageNet dataset (Krizhevsky et al., 2012) consisting of 13,000 96×96 real-world RGB images uniformly drawn from 10 classes

divided into training and test sets of 5K and 8K images, respectively. For validation purposes, the training dataset is further divided into training and validation sets of 4,500 and 500 samples, respectively.

ModelNet40. The ModelNet40 dataset (Wu et al., 2015) contains 12,311 3D meshes of objects belonging to 40 classes. We use the official split with 9,843 training samples and 2,468 validation samples. In contrast with previous works which sample 1,024 points from these meshes, (e.g. Wu et al. (2019)), we sample 512 points uniformly from the faces of each mesh, along with the normal vectors at each of these positions. These position and normal vectors serve as input to the model.

ModelNet10 Voxelized. ModelNet10 is a subset of ModelNet40 containing orientation-aligned samples from 10 classes of objects. We voxelize ModelNet10 based on a subsampling of 4096 points from the meshes. First, we normalize these points to fall within the interval $[-1, 1]$. Next, we bin these points into a grid of $40 \times 40 \times 40$ voxels. All nonzero voxels get assigned their location as feature value. Afterwards, to align with our point-cloud configuration, we mask out the all but 512 nonzero voxels. Throughout the network, we only apply convolutions at the locations of each nonzero voxel (effectively masking out activations at all other voxel locations). Like with the point-cloud configuration, to limit computational complexity, we limit the convolutions to integrate over the closest 256 points.

(a) (b)

Figure 8: An example of a point-cloud sampled from ModelNet10 (a), and a corresponding voxel representation of the same sample (b).

B Experimental Details

B.1 General Remarks

Code repository and logging. Our code is written in `PyTorch`. We utilize `wandb` (Biewald, 2020) `hydra` (Yadan, 2019) and `pytorch-lightning` (Falcon et al., 2019) for logging and code structuring. Our experiments are performed on NVIDIA TITAN RTX, A6000 and A100 GPUs, depending on the size of the datasets and inputs considered, and our code is publicly available at ... .

Residual Block Architecture. The architecture we use for our residual block is visualized in figure 9.

Normalized relative positions. The kernel network φ_{Kernel} can, in principle, receive arbitrary coordinates as input. However, considering unitary step-wise relative positions, i.e., 0, 1, 2, ... , N, can be problematic from a numerical stability perspective as N may grow very large, e.g., N=16000 for the Speech Commands dataset. Consequently, based on insights from the Implicit Neural Representations, e.g., Sitzmann et al. (2020); Fathony et al. (2021), we normalize the coordinates such that they lie in the space $[-1, 1]^{\text{D}}$ for D-dimensional kernels. To this end, we map largest unitary positions seen during training $[0, N]$ to a uniform linear space in $[-1, 1]$.

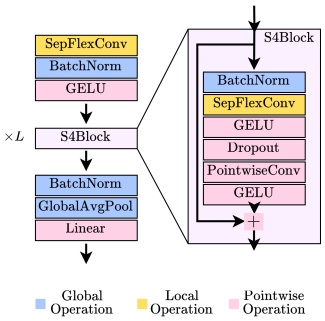


Figure 9: The CCNN architecture.

Table 3: Best hyperparameters found for $\text{CCNN}_{4,140}$ on all tasks considered.

	w_0	Dropout	Learning Rate	Weight Decay	Batch Size	Epochs
sMNIST	2976.49	0.1	0.01	1e-6	100	210
pMNIST	2985.63	0.2	0.02	0	100	210
sCIFAR10	2386.49	0.0	0.02	0	50	210
Speech Commands (raw)	1295.61	0.2	0.02	1e-6	20	160
Speech Commands (mfcc)	750.18	0.2	0.02	1e-6	100	110
ListOps	784.66	0.1	0.001	1e-6	50	60
Text	2966.60	0.2	0.001	1e-5	50	60
Image	4005.15	0.2	0.01	0	50	210
Pathfinder	2272.56	0.2	0.01	0	100	210
CIFAR10	1435.77	0.1	0.02	0.0001	50	210
CIFAR100	3521.55	0.1	0.02	0.0001	50	210
STL10	954.28	0.1	0.02	0	64	210
2DImage	2085.43	0.2	0.02	1e-6	50	210
2DPathfinder	1239.14	0.1	0.01	0	100	210
ModelNet40	50	0.0	0.02	1e-8	16	200

Table 4: Best hyperparameters found for $\text{CCNN}_{6,380}$ on all tasks considered.

	w_0	Dropout	Learning Rate	Weight Decay	Batch Size	Epochs
sMNIST	2976.49	0.1	0.01	0	100	210
pMNIST	2985.63	0.2	0.02	0	100	210
sCIFAR10	4005.15	0.25	0.01	0	50	210
Speech Commands (raw)	1295.61	0.2	0.02	1e-6	20	160
Speech Commands (mfcc)	750.18	0.2	0.02	1e-6	100	110
ListOps	784.66	0.25	0.001	0	50	60
Text	2966.60	0.3	0.02	0	50	60
Image	4005.15	0.1	0.01	0	50	210
Pathfinder	2272.56	0.1	0.01	1e-6	100	210
CIFAR10	1435.77	0.15	0.02	0	50	210
CIFAR100	679.14	0.2	0.02	0	50	210
STL10	954.28	0.1	0.01	1e-6	64	210
2DImage	2306.08	0.2	0.02	0	50	210
2DPathfinder	3908.32	0.2	0.01	0	100	210
ModelNet40	50	0.0	0.02	1e-7	32	200

B.2 Hyperparameters and training details

Optimizer and learning rate scheduler. All our models are optimized with AdamW (Loshchilov & Hutter, 2017) in combination with a cosine annealing learning rate scheduler Loshchilov & Hutter (2016) and a linear learning rate warm-up stage of 10 epochs.

Best hyperparameters found. We perform hyperparameter search on the learning rate, dropout rate, weight decay, and ω_0 of our CCNNs for each task considered.[2] The best hyperparameters found are reported in Tables 3 and 4.

B.3 Additional experiments

Parameter efficiency on ModelNet40. To further assess parameter efficiency of the CCNN on ModelNet40, we run a number of experiments with smaller model sizes. Results for the following models are shown in Fig. 5: $\text{CCNN}_{4,16}$ with 4 residual blocks, a channel size of 16 and a kernel network hidden size of 8, with 6,545 total parameters. $\text{CCNN}_{4,32}$ with 4 residual blocks, a channel size of 32 and a kernel network hidden size of 8, with 14,401 total parameters. $\text{CCNN}_{4,48}$ with 4 residual blocks, a channel size of 48 and a kernel network hidden size of 8, with 26,353 total parameters. We use a learning rate of $2e^{-2}$, no weight decay, and an ω_0 of 50.

[2] ω_0 serves as a prior on the variance of the data that is fitted with several types of implicit neural representations, e.g., SIRENs Sitzmann et al. (2020), MFNs Fathony et al. (2021), etc.