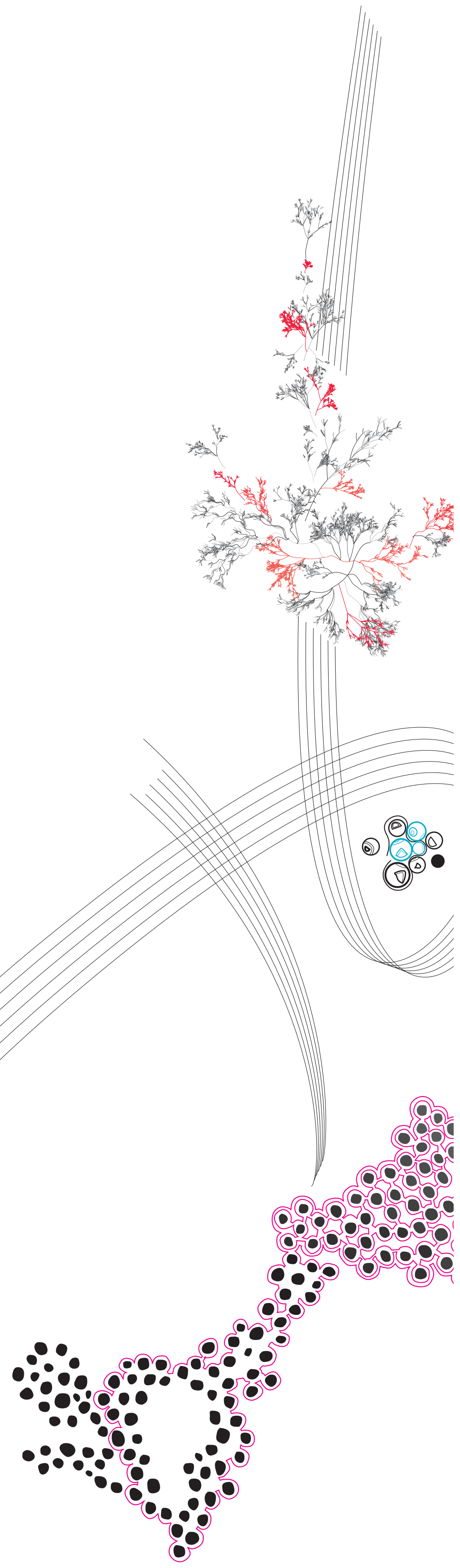
BSc Thesis Applied Mathematics



Continuous representation of kernels used in convolutional neural networks

Scott Jonker

Supervisor: J. Wolterink

January, 2023

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Continuous representation of kernels used in convolutional neural networks

Scott Jonker

January, 2023

Abstract

Within the field of machine learning the research of *Convolutional Neural Networks* (CNN) has been rapidly progressing. A recent development has been representing the kernels used in the convolutional layers continuously through a separate auxiliary neural network. A practical application of a CNN is in medical imaging, where the neural network is trained to detect micro-bubbles in order to map the vascular network. This work investigates the use of continuous kernel representations for the problem of micro-bubble localization. This is done by training the neural network on a simulated ultrasound signal where the result aims to replicate the corresponding ground truth bubble location. This is done with the PyTorch library through the use of Jupyter notebooks. The results show that representing the continuous kernel with a small neural network was successful, but how using large kernels models in turn incorporates long-term dependencies which are not beneficial for the problem context. *Keywords*:

SIREN, convolutional neural network,

1 Introduction

Within the field of machine learning, the study of artificial neural networks (ANNs) is a rapidly progressing topic of research. Convolutional neural networks (CNNs) are a widely used type of ANN that have been successful in applications such as image classification, natural language processing, and medical image analysis. As the name suggests, CNNs contain convolutional layers where a discrete convolution is applied to the input tensor with a kernel that is represented by a sequence of weights. These weights are a portion of the parameters that compose the network which are optimized during the networks training.

Recently, researchers have been enthusiastic regarding CNNs as they have shown to be successful in tasks such as image classification, object detection, and segmentation. Traditionally, CNNs were not suited to tasks such as sequential modeling and natural language processing and were instead dominated by Recurrent Neural Networks (RNNs). Recent research suggests that CNNs are actually able to handle these problems [1], which helps to resolve known issues with RNNs such as vanishing gradients.

Deep neural networks are used in order to incorporate non-linearity and increase receptive field size in CNNs. A method used in order to increase the receptive field size is the dilated convolution, where zeros are embedded in between the kernel weights which increases the receptive field without increasing the overall number of parameters. A reason for these methods is due to the fact that if one was to use large kernels in a shallow network, the total parameters and overall network size will greatly increase and quickly become computationally infeasible.

In order to overcome the restriction on the size of the kernel, a radically different approach has been proposed in the paper *CKConv: Continuous Kernel Convolution For Sequential Data* [4]. As the title suggests, the approach consists of representing the kernel by a continuous function. By doing so, increasing the size of the kernels doesn't impact the number of parameters and is computationally feasible.

Ultrasound localization microscopy is a technique in medical imaging where ultrasound signals are processed by identifying the location of micro-bubbles in order to visualize the underlying vascular structure. This process is difficult due the low concentration of micro-bubbles in the blood vessels. Recently, the Physics and Fluids group of the University of Twente have investigated the use of dilated convolutional neural networks in order to localize micro-bubbles based on raw ultrasound RF data [2]. The aim of my research is to investigate the process and efficacy of using continuous kernels for the same task of micro-bubble localization.

2 Related Work

2.1 Neural representations

Several recent investigations in machine learning have been the application of coordinate-based neural networks, dubbed as **neural fields** [6]. The aim of such networks is to map the coordinates (input) to field that is being represented. One such application is the continuous representations of images.

Following such methods, a small neural network MLP^{Ψ} will be used in order to represent the continuous kernel. The MLP^{Ψ} structure will be based on the paper regarding **SIRENs** [5]. A **SIREN** is a fully connected neural network which utilizes sine as the activation function. The reason for this choice is their reported ability to accurately represent arbitrary functions.

2.2 Continuous kernel convolutions on sequential data

The research conducted in the paper regarding CKConvs [4] is the main influence for this research. The network architecture used demonstrates the success of using continuous kernels representations in convolutional layers.

The experiments conducted in the research consist of classification problems based on sequential data. One task is image classification on the permuted MNIST data set, where the model has to determine the integer a handwritten number represents ranging between zero and nine. Another is classifying spoken words based on audio signals, with a total of ten possible words.

2.3 Micro-bubble localization

The work done by the Physics of Fluids regarding micro-bubble localization [2] is basis for the problem investigated in the report. The research includes data generation as well as micro-bubble localization through a convolutional neural network. The data from the research will be used as well as other key components.

3 Methods

Describe what the problem is. This is deconvolving the raw RF signal into a signal of equal length which should have values between (0,1). Over a certain threshold indicates a bubble is present, while not present when the value is below the same threshold.

3.1 SIREN representation of continuous kernel

To construct a continuous kernel which can be optimized during training a small neural network is used. The network that is used is a multi-layer perceptron utilizing sine activation functions. The model is based on the work done in the paper [5] where it is labeled as a SIREN network. The SIREN MLP^{Ψ} used to represent the continuous kernel has 3-layers, 32 hidden channels, and n out channels with $n = C_{in} * C_{out}$ for the corresponding convolution layer in the main network.

The implementation of the kernel function $\Psi(x)$ is formulated below where ϕ_i is an intermediate layer of the network. The domains of the functions ϕ_1 and ϕ_2 are $\phi_1 : \mathbb{R} \mapsto \mathbb{R}^{32}$ and $\phi_2 : \mathbb{R}^{32} \mapsto \mathbb{R}^{32}$ respectively. The corresponding weight matrices are $\mathbb{R}^{32x1}$ and $\mathbb{R}^{32x32}$ and both bias vectors are of size $\mathbb{R}^{32}$.

The MLP^{Ψ} uses a real valued sequence, defined as $\{\Delta\tau\}$, as the input and the output is the corresponding kernel values. The output consists $C_{in} * C_{out}$ real valued sequences which are used in the convolution.

$$\phi_1(\mathbf{x}) = \sin(\omega_0 \cdot \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \tag{1}$$

$$\phi_2(\mathbf{x}) = \sin(\omega_0 \cdot \mathbf{W}_2\mathbf{x} + \mathbf{b}_2) \tag{2}$$

$$\Psi(\mathbf{x}) = \mathbf{W}_3(\phi_2 \circ \phi_1)(\mathbf{x}) + \mathbf{b}_3 \tag{3}$$

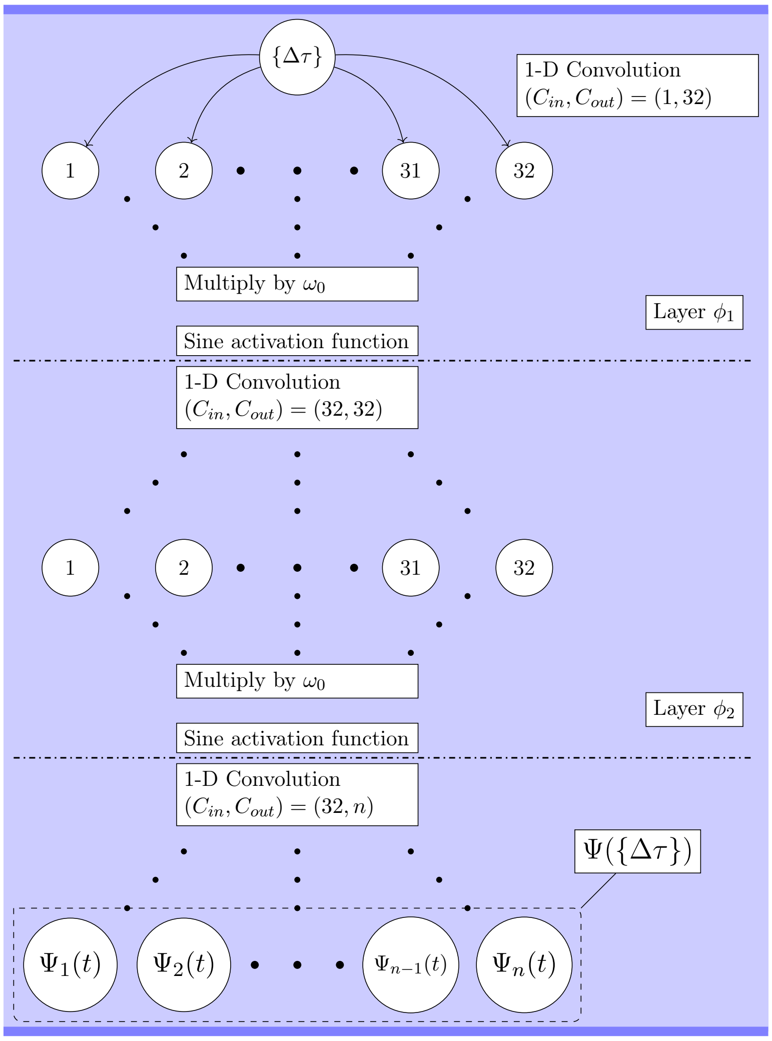


FIGURE 1: SIREN MLP^{Ψ} architecture used to represent the kernel

3.2 Sampling of the kernel

In order to sample the MLP^{Ψ} for a given convolutional layer, the real valued sequence $\{\Delta\tau\}$ is used as the input. The length of the sequence $|\{\Delta\tau\}| = K$ is equal to the kernel size for the convolutional layer and is fixed prior to training. The sequence $\{\Delta\tau\}$ is a linearly spaced vector from negative one to one. For example, if the kernel is of size five then $\{\Delta\tau\}^{N_k} = \{-1,\ -0.5,\ 0,\ 0.5,\ 1\}$ is the corresponding sequence.

The output from MLP^{Ψ} is a tensor of the shape $(1, C_{in} * C_{out}, K)$ and is reshaped into (C_{out}, C_{in}, K). This tensor is the kernel used when computing the convolution.

3.3 Convolution

3.3.1 Standard convolution

The convolution operation takes an input $\mathbf{X}$, a weight tensor $\mathbf{W}$, and a bias vector $\mathbf{b}$ in order to calculate the output $\mathbf{Y}$. Symbolically this would be $\mathbf{X} * \mathbf{W} + \mathbf{b} \rightarrow \mathbf{Y}$. The size of each are written below where N represents the batch size, C_{in} the number of input channels, C_{out} the number of output channels, and L the signal length. The signal length is reduced after the operation to $\tilde{L}$ which is present in the output $\mathbf{Y}$.

$$\mathbf{X} \in \mathbb{R}^{N \times C_{in} \times L}$$
$$\mathbf{W} \in \mathbb{R}^{C_{out} \times C_{in} \times K}$$
$$\mathbf{b} \in \mathbb{R}^{C_{out}}$$
$$\mathbf{Y} \in \mathbb{R}^{N \times C_{out} \times \tilde{L}}$$

The following notation $[\mathbf{X}]_{a,b,c}$ will denote the element (a, b, c) for the respective tensor. The value for each element $l \in \tilde{L}$ is equal to the sum of the product between elements of the weight tensor $\mathbf{W}$ and the input $\mathbf{X}$. This is done for each batch $N_i \in N$ and each output channel $C_{out_j} \in C_{out}$. The bias $\mathbf{b}_j$ is added corresponding to the output channel. The exact computation is formulated in equation 4, where $k = \frac{K}{2}$.

$$[\mathbf{Y}]_{N_i,C_{out_j},l} = \mathbf{b}_j + \sum_{c=1}^{C_{in}} \sum_{a=-k}^{k} [\mathbf{W}]_{C_{out_j},c,a} \cdot [\mathbf{X}]_{N_i,c,(l+a)} \tag{4}$$

In the case of a dilated convolution, with a dilation factor of d, the calculation in slightly different. It effectively spaces out the elements of the kernel by inserting holes between the elements. The exact computation is formulated in equation 5, where the only difference is which elements of $\mathbf{X}$ are used.

$$[\mathbf{Y}]_{N_i,C_{out_j},l} = \mathbf{b}_j + \sum_{c=1}^{C_{in}} \sum_{a=-k}^{k} [\mathbf{W}]_{C_{out_j},c,a} \cdot [\mathbf{X}]_{N_i,c,(l+d \cdot a)} \tag{5}$$

Calculating the convolution between the given weight tensors and the layer's input is a straightforward process. This is done with PyTorch's functional conv1d function which takes the input tensor and the weight tensor and calculates the convolution. It also has an argument to add dilation to the convolution. This convolution method is used when emulating the previous work on micro-bubble localization [2] which is a conventional convolutional neural network whoch uses kernels of size three, a depth of 12, and dilation increasing by a factor at each subsequent layer.

3.3.2 FFT Convolution

Due to the large size of some of the kernels used it is imperative to compute the convolution using the Fast Fourier Transform. This is due to the fact that the computation time is exceedingly long when using the standard convolution. This is possible by utilizing the convolution 6 where $\mathcal{F}$ is the Fourier Transform.

$$(x * w) = \mathcal{F}^{-1}\{\mathcal{F}\{x\} \cdot \mathcal{F}\{w\}\} \tag{6}$$

This process is straightforward when the two signals (x and w) are of equal length. Unfortunately this is not the case and the w is shorter than the input.

PyTorch has Fourier transform functions to facilitate this but does not have an function to easily compute the FFT convolution. Instead the python package [3] was used which adds a PyTorch FFT convolution function. The behaviour of the added function was tested using representative tensors (PyTorch *rand* tensors) to confirm that it operates like the standard convolution.

3.4 CKCNN architecture

Based on previous naming, [4] the term CKCNN will be used in place of *Continuous Kernel Convolutional Neural Networks.*

3.4.1 CKConv layer

The primary difference between the CKCNN architecture compared to a conventional CNN are the convolutional layers. The CKConv layer acts as the analogue to the standard convolutional layers. Each CKConv layer is defined by the number of input channels, the number of output channels, and the kernel size. Each CKConv layer uses its own SIREN MLP^{Ψ} in order to represent the continuous kernel function. During each forward pass the MLP^{Ψ} is sampled with the sequence $\{\Delta\tau\}$ in order to obtain the kernel used during the convolution calculation. The forward pass is shown in the diagram 2.

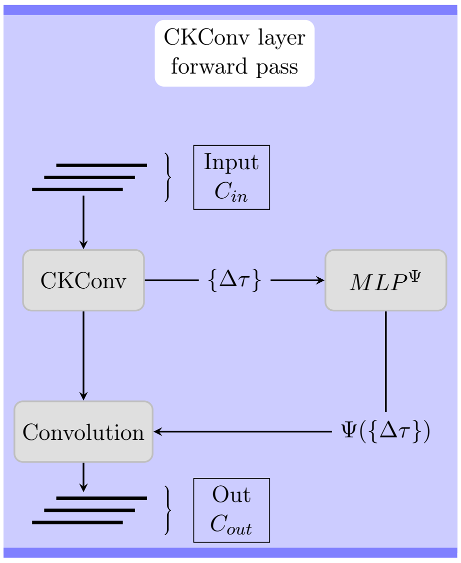


FIGURE 2: CKConv forward pass

3.4.2 Kernel of size 3 with increasing dilation

The work done by the Physics of Fluids group on micro-bubble localisation [2] utilizes a convolutional neural network. The architecture is depicted in figure 3 and is composed of conventional convolution layers, batch normalization, and ReLu activation functions.

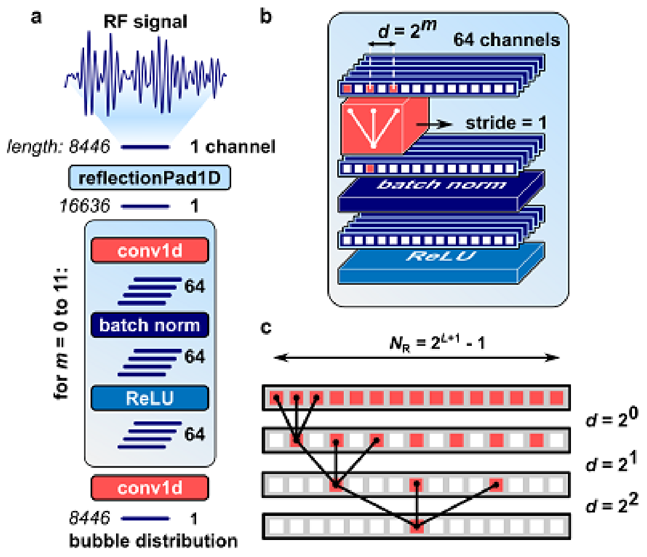


FIGURE 3: Figure from paper [2] "**Architecture of the dilated convolutional neural network.** **a)** Architecture overview, showing a 1D reflection padding layer, 1D convolutional layers, batch normalization layers and Rectified Linear Unit (ReLU) activation layers. **b)** Detailed illustration of the three-layer block outlined in a for $\mathbf{m = 1}$, showing the convolution kernel, the stride, and the dilation rate d. **c)** Stack of convolutional layers illustrating the exponential expansion of the receptive field NR as a function of the number of convolutional layers **L**."

In order to make as direct of a comparison with to this model, the general layout of is kept the same. The only change made is replacing the convolution layers with CKConv layers. This is simple to do as the CKConv layer operates in the same way within the overall network. An important note is that representing small kernels (3 in this case) through the MLP^{Ψ} is not advantageous. The original CNN model consists of $137,729$ trainable parameters while the CKCNN model consists of $636,497$ trainable parameters.

3.4.3 CKCNN architecture for large kernels

Initially, the general architecture of the neural network would follow a similar structure to the previous work regarding Continuous Kernel Convolutional Networks [4]. To summarize, the architecture utilized a block structure where the CKBlock was the primary element. Each CKBlock was composed of two CKConv layers with each followed by layer normalization, a ReLu activation function, and optional dropout. Also included was a residual skip around the CKBlock.

The CKCNN architecture used is instead considerably simpler. It consists of an initial reflection padding layer which is based the the kernel size and pads the input such that the output signal length remains the same as the input. Afterwards there is a CKConv layer, batch normalization, and a ReLu activation function. Those three layers are repeated four times. Finally, a regular Conv1d with a kernel size of one consolidates the hidden channels of the network into a single output channel. The architecture is shown in figure 4. This model consists of $28,392$ trainable parameters.

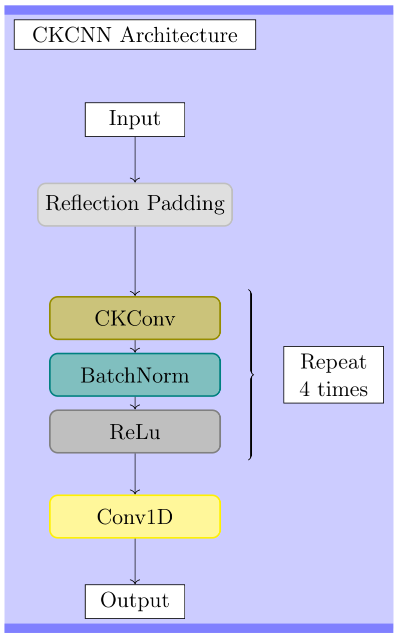


FIGURE 4: NEEDS UPDATE

3.5 Training

The loss function used is the dual-loss function described in [2] which is a linear combination of two separate losses. The first is a form of soft label training. It does so by by calculating the convolution between the model output with a Gaussian kernel $G_\alpha = e^{-\alpha x^2}$ where a decrease in α widens the width of the Gaussian kernel. After the Gaussian kernel is applied to both the prediction and the ground truth, L_1 loss is applied to the soft labels. The second is a form of hard label training which is the dice loss. The linear combinations coefficients are defined as ϵ_1 and ϵ_2 respectively.

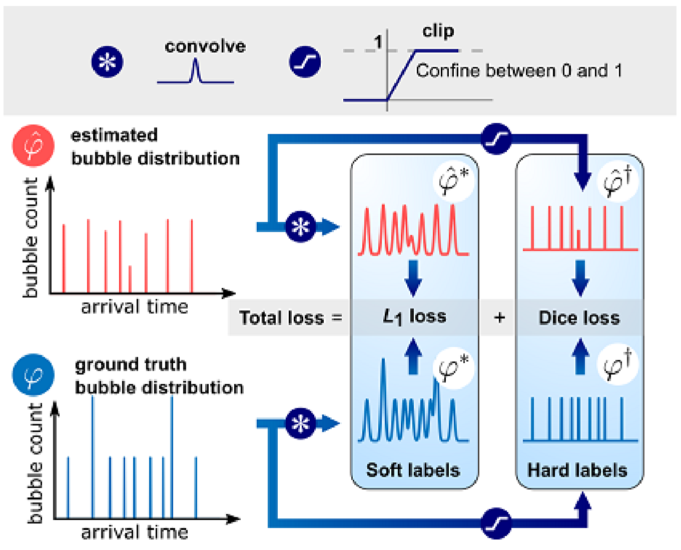


FIGURE 5: Dual loss diagram [2]

During training the value of α remains the same at $\alpha = 0.1$ and the values of ϵ_1 and ϵ_2 are set to $\epsilon_1 = 1$ and $\epsilon_2 = 1.6$ based on previous findings [2].

In order to optimize the model parameters the Adam algorithm is used. The total number of epoch is 1250 with an initial learning rate of 0.001 and the learning rate is decayed instantly by a factor of 0.1 at epoch 1000.

4 Results

In order to determine evaluate accuracy of the prediction the F_1 score is used, which is the harmonic mean between the precision and recall. TP, FP, and FN represent true positives, false positives, and false negatives respectively.

$$P = \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

$$R = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \tag{9}$$

The F_1 score calculated is dependent dependent on a localization tolerance t_{tol} and the threshold φ_{th} which is visualized in figure 6.

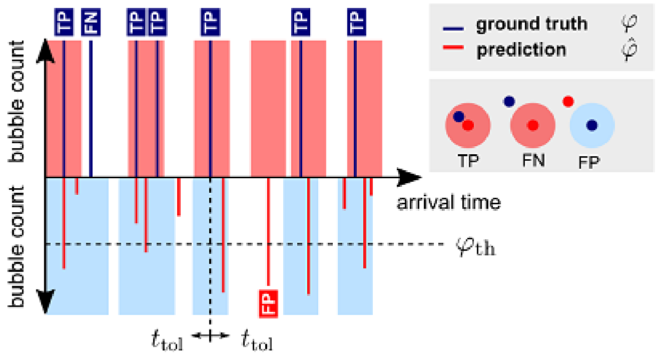
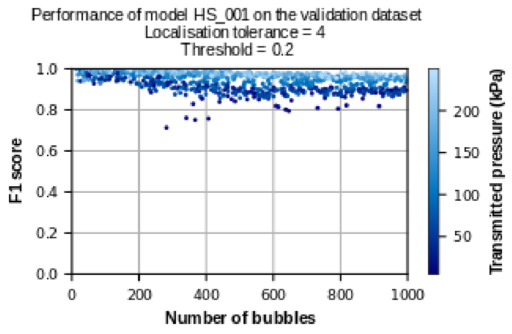


FIGURE 6: Visualization of model evaluation [2]

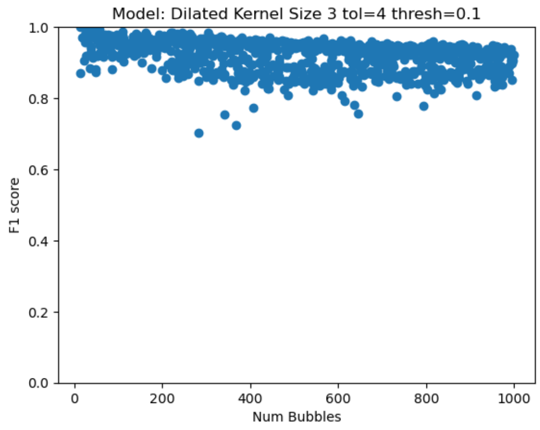
4.1 CKCNN with Dilated Kernels of size three

The overall model structure was described in section 3.4.2 where the only remaining hyper-paramters pertain to the MLP^{Ψ} and the CKConv layer. In this case that only involves the hyper-parameter ω_0. The model was not overly sensitive to the value of ω_0, and the final value used was $\omega_0 = 8$.

After training the model, the optimal threshold for each tolerance ($\{0, 1...8, 9\}$) is calculated based on the data used during training. The optimal tolerance in this case means which tolerance leads to the highest average F_1 score for all the data. In order to test the accuracy of the model, the trained model predicts the micro-bubble location based on 960 new data sets using a tolerance of 4 and the corresponding optimal threshold of 0.1. In order to visualize the data, the F_1 score of each data set is displayed in a scatterplot against the number of micro-bubbles in each data set. Comparing the graphs (figure 7) the result is comparable to the conventional CNN.



(A) Conventional CNN



(B) CKCNN

FIGURE 7: Comparison of results

4.2 CKCNN with Large Kernels

When constructing the CKCNN's for large kernels there is a large amount of flexibility in the architecture. The main points investigated are the size of the kernels and the selection of ω_0. The total number of CKConv layers was kept at four throughout testing.

The number of hidden channels within the CKCNN's was set to 22, as high as possible. The number of hidden channels could not be increased due to running out of memory on the GPU (24 GB). This was likely due to the FFT convolution calculation, and is mentioned in the issues on the GitHub repository [3]. The hidden channels were maximized in order to add more complexity to the model as the CKCNN architecture suffered from under-fitting.

4.2.1 Experiments on kernel size and ω_0

In order to get an idea of the appropriate kernel size CKCNN models were trained for 1250 epochs with the kernels of size: 65, 131, 263, 527, 1055, and 2111 with the fixed $\omega_0 = 30$ hyper-parameter. In order to get an approximation of the accuracy of these models quickly the optimal threshold φ_{th} for each t_{tol} was calculated and stored with the average F_1 score. This was done on the same data used during training.

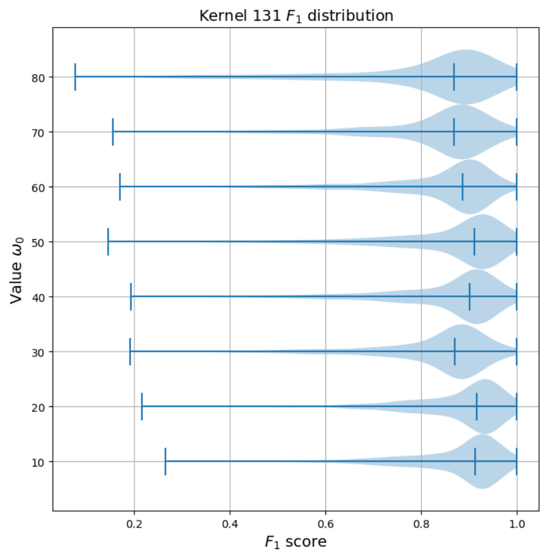
The optimal φ_{th} and average F_1 score for $t_{tol} = 4$ is shown for each model in table 1.

Kernel Size	φ_{th}	Mean F_1
65	0.15	0.866
131	0.15	0.835
263	0.15	0.828
527	0.15	0.809
1055	0.15	0.676
2111	0.1	0.364

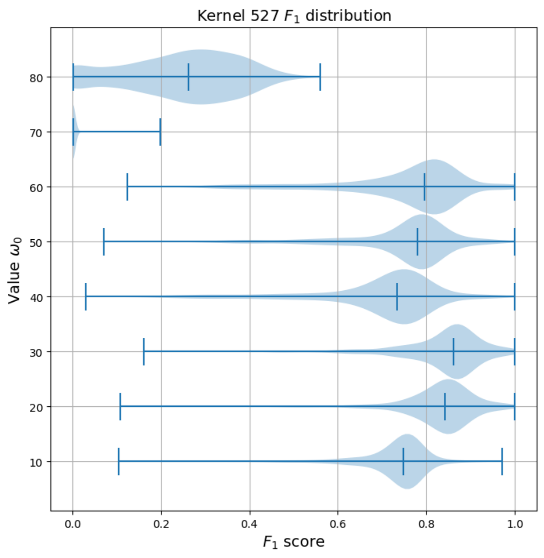
TABLE 1: Optimal φ_{th} and mean F_1 score for tolerance $t_{tol} = 4$. The values are derived from the data used during training.

The kernels of size 131 and 527 were tested further with several values of the hyper-parameter ω_0, namely $\omega_0 \in \{10, 20, 30, 40, 50, 60, 70, 80\}$. The models were trained in the same fashion, along with computing the optimal φ_{th} values. The trained models had the same optimal φ_{th} (for $t_{tol} = 4$) when tested on the training data for all the ω_0 values and will be used when evaluating the accuracy on new testing data.

Afterwards the trained models predicted the micro-bubble locations on new testing data (960 signals) using the tolerance $t_{tol} = 4$ and the optimal φ_{th} of the respective model. The F_1 score is calculated for each prediction, and the collection of the scores for each CKCNN comprises that models F_1 score distribution. In order to visualize the F_1 scores for each model the distributions are shown through the violin plots of said distributions 8 where the middle hash marks represent the median of each distribution. The mean and median values are provided in the table 2.



(A) Violin plot for kernel size 131



(B) Violin plot for kernel size 527

FIGURE 8: Violin plots of the F_1 distributions

ω_0	Kernel 131		Kernel 527	
	Median	Mean	Median	Mean
10	0.912	0.869	0.748	0.723
20	0.917	0.870	0.843	0.822
30	0.871	0.821	0.861	0.829
40	0.902	0.842	0.733	0.686
50	0.911	0.847	0.781	0.737
60	0.886	0.826	0.796	0.746
70	0.868	0.822	0.001	0.007
80	0.870	0.803	0.262	0.247

TABLE 2: Median and mean of the F_1 distributions

5 Discussion

The results from the CKCNN model for the dilated kernels of size 3 matched those for the conventional CNN. This does not take full advantage of the continuous kernel representation as a major benefit of this method is the capability for large kernels. These results do support the success of the SIREN MLP^{Ψ} architecture in representing the continuous kernel, and their ability to model arbitrary functions.

The results from the shallow CKCNN models with large kernels are more convoluted. The CKCNN with the kernel size of 131 performed better than when the kernel was of size 527. This seemed to be the case in general, where increasing the kernel size led to worse results. One explanation for this is that the most important information for detecting a micro-bubble at a given point in the signal is the information in close proximity of said point. This seems to be the opposite of other problems such as image classification (such as pMNIST), where the neural network benefits from incorporating all of the data to make a final result.

6 Conclusion

The results show that incorporating long-term dependencies is not beneficial for every task with one of those being micro-bubble localization. Incorporating long-term dependencies may be beneficial in order to filter out noise patterns present across the entire signal.

References

[1] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

[2] Nathan Blanken, Jelmer M. Wolterink, Hervé Delingette, Christoph Brune, Michel Versluis, and Guillaume Lajoinie. Super-resolved microbubble localization in single-channel ultrasound rf signals using deep learning. *IEEE Transactions on Medical Imaging*, pages 1–1, 2022.

[3] Frank Odom Frank Odom, III. fkodom/fft-conv-pytorch.

[4] David W. Romero, Anna Kuzina, Erik J. Bekkers, Jakub M. Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data, 2021.

[5] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020.

[6] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *CoRR*, abs/2111.11426, 2021.