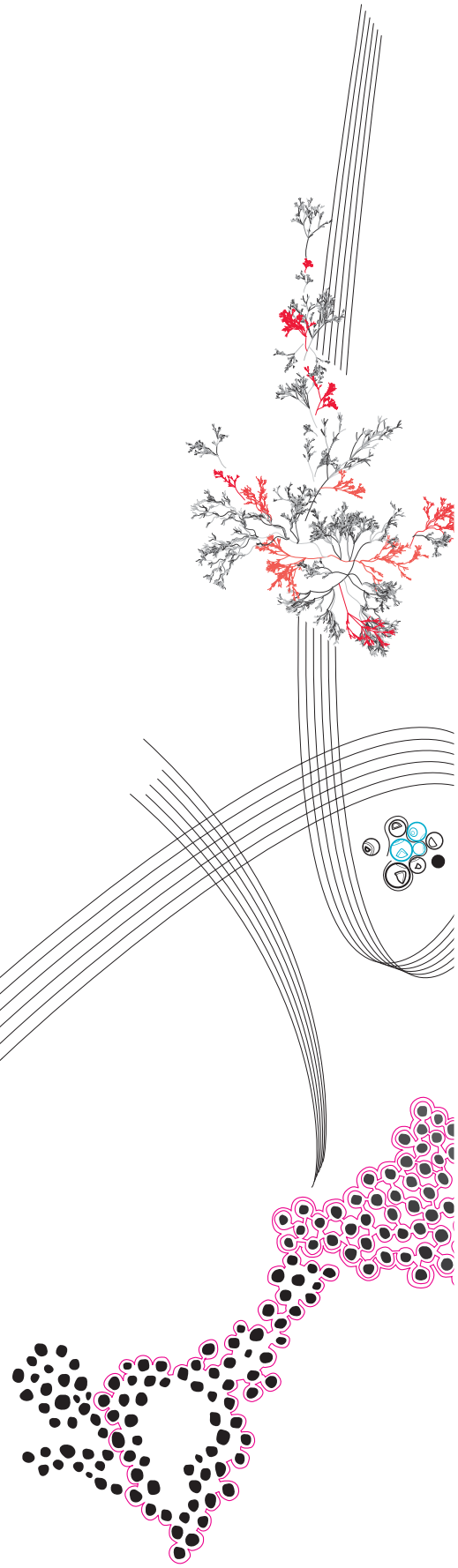

BSc Thesis Applied Mathematics
& Technical Computer Science

Multiscale Convolutions for an Artificial Neural Network

Ioannis Linardos

Supervisors:
Applied Mathematics: Yoeri Boink
Computer Science: Nirvana Meratnia & Jeroen Klein Brinke

July, 2019

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

UNIVERSITY OF TWENTE.

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance. I would like to thank my Applied Mathematics supervisor ir. Yoeri Boink and my Computer Science supervisors dr. Nirvana Meratnia and ir. Jeroen Klein Brinke. Their expertise was invaluable in formulating the research project and carrying out the study at each step. The patience and interdisciplinary competence they exhibited were of great importance because they were asked to co-supervise a research project that would fulfill the requirements of two degree programmes.

Multiscale Convolutions for an Artificial Neural Network

Linardos I. *

July, 2019

Abstract

The study investigates the possibility of using convolutional neural networks across input of different sampling rates, focusing on one-dimensional convolutions. This is an idea that has not been adequately studied although it may produce useful results that expand the usefulness of convolutional neural networks. The problem was approached from the perspective of algebraic multigrid. Three interpolation methods were tested on audio classification neural networks trained for input of different sampling rates: nearest neighbor, linear interpolation and inverse distance weighting. The approach was extended to pooling and fully connected layers. In the case of using a neural network trained for high sampling rate input with input of low resolution, the method of linear interpolation gave promising results. Moreover, the results hint that pooling layers should not be changed in the process of multiscaling. In the case of training for low sampling rate and testing with input of high sampling rate, there is no unique solution to the system of weight equations. In dealing with this problem, the approach of directly prolonging the convolution kernels was tried using the three interpolation methods that were explained above as well as the method of kernel dilation. The last method, kernel dilation, appeared to be considerably effective in upscaling.

Keywords: convolutional neural networks, algebraic multigrid, multiscale methods, nearest neighbor, linear interpolation, inverse distance weighting, kernel dilation

*Email: i.linardos@student.utwente.nl

1 Introduction

In this work, we consider the problem of multiscaling convolutional neural networks (CNN). CNNs are deep neural networks whose learned parameters are the values of discrete convolution kernels. They are mainly used in processing data that benefit from keeping their original spatial and structural information. For example, they are used in image, audio and video processing. The tasks that can be performed using CNNs include but are not limited to classification, denoising or labeling [1].

CNNs are trained to perform a specific task for input of a specific resolution/sampling rate. In case the same type of processing needs to be performed to input of different resolution/sampling rate, a new CNN is created and trained. This procedure is lengthy because discovering a functional network architecture usually requires trial and error and the research in optimal heuristic procedures is still in development [2]. Moreover, the computational cost of training CNNs is high [3]. Another option would be to resample the dataset to the resolution in which the CNN has been originally trained but this also adds a considerable computational overhead.

In order to avoid these costs, we examined whether it is possible to modify an already trained CNN for input of a different resolution. Modifying a CNN that has been trained for high resolution input to perform for low resolution is called downscaling the network while the opposite process is called upscaling.

By finding effective ways to multiscale (upscale and downscale) existing network, the CNNs will become more versatile; a network could be trained for one input resolution/sampling rate and then used for another. Furthermore, it is possible to optimize the training of CNNs by first training the network using input that minimizes the computational costs (i.e. lower resolution) and then scaling the network for the desired input. Moreover, even if the multiscaled network does not have adequate accuracy and retraining cannot be completely skipped, these methods may be used to initialize the training parameters (architecture and weights) of a neural network and shorten the development time.

Multiscaling methods for CNNs is a new field of research; consequently, there is very limited research on the subject. Haber et al. explored some ideas with considerable success in two-dimensional CNNs for image classification [4]. One of the methods they proposed was based on algebraic multigrid (AMG), a numerical method used to solve large systems of equations using a multilevel hierarchy. AMG has been used in varied fields of scientific research ranging from fluid mechanics [5] to queuing theory [6] but its relevance to CNNs just started being investigated.

The AMG approach requires the construction of different prolongation (interpolation) and restriction (coarsening) operators and the success of the multiscaling depends on that choice [7]. In this work, we explored three such methods: nearest neighbor, linear interpolation and inverse distance weighting. In contrast to the work of Haber et al. [4], the task at hand is audio classification performed by one-dimensional CNNs. Multiscaling techniques for one-dimensional CNNs is an area that remained up to this point unexplored. In this study, we delved into the connection between AMG and CNN multiscaling and explored the practical implementation of the prolongation and coarsening strategies in specific scenarios. Additionally, in dealing with some deficiencies of AMG in the case of upscaling, the strategy of directly prolonging the convolution kernels was examined. In order to test the efficacy of these methods, a dataset was collected and a number of CNN architectures were trained in two different sampling rates.

2 Methodology

In this section, we describe in detail the methods that were used to solve the problem. At first, the theory of AMG is explained focusing on why it is relevant to multiscaling CNNs. Then, we explain the specific prolongation and restriction methods that were used and how they were applied. Afterwards, we discuss how we dealt with the pooling and dense layers that are usually present in a CNN architecture. Last but not least, we explore a new strategy to upscaling to deal with a deficiency in the AMG approach.

2.1 Algebraic Multigrid and Multiscaling CNNs

As mentioned, the backbone of the approach in multiscaling followed in this work is the method of algebraic multigrid. Multigrid is a numerical method developed to solve large systems of discrete partial differential equations. It is based on the idea of coarsening the discretization grid based on a physical geometric interpretation of the problem using hierarchical algorithms (geometric multigrid). This inspired the development of algebraic multigrid (AMG) which is used to solve large (usually sparse) systems of linear equations using the same multigrid principles but without any references to a geometric origin of the problem; it is based only on the information contained in a given matrix K to construct the hierarchy of grids and the corresponding prolongation and restriction operators [8] [7].

In general, AMG methods are designed to solve linear systems of the form $Ku = f$ where K is a sparse matrix. This system is called "the finest grid problem" and the solution is found within a hierarchy of coarser grid problems. By finest scale we define the dimension of the (unknown) vector u. The method transitions to a sequence of coarser grids (grids of coarser scales) in which the sparse matrix K is transformed to a coarser grid operator using some intergrid transfer (prolongation and restriction) operators [8].

In order to make the connection with our research area, we need to interpret a CNN from an algebraic point of view where the application of a convolution kernel in an input array is represented by a matrix multiplication.

Let a coarse scale (low resolution/sampling rate) H and a fine scale (high resolution/sampling rate) h with $h > H$. Moreover, let s_H and s_h the convolution kernels that operate on the low sampling rate (coarse scale) input u_H and high sampling rate (fine scale) input u_h respectively. In the case of audio classification, which was the focus of this work, the input is an audio recording represented by a one-dimensional array. The sparse matrices K_H and K_h are the matrix representation of the coarse and fine scale convolution kernels respectively. The form of these sparse matrices will be explained in the next section.

In this framework, the finest scale problem is the application of a convolution kernel to a high sampling rate input represented by $K_h u_h$. In this approach, the purpose of applying AMG is not to solve the linear system $K_h u_h = f$ per se, since both K_h and u_h are taken to be known. In contrast, the main goal is to find a new sparse matrix K_H which is equivalent to applying the fine scale convolution to a coarser grid, meaning to an input of lower sampling rate. Through this process, given a fine scale convolution kernel, we can derive a coarse scale one, effectively downscaling the kernel.

AMG is a method used to downscale a problem in pursuance of a configuration that is easier to solve. However, in the case of CNNs, upscaling the operators is also a point of interest. In this case, the opposite procedure is followed with known K_H and unknown K_h. However, as we shall see, this is not as straightforward as the case of downscaling and it presents additional challenges.

In AMG, constructing a coarse scale operator K_H given a fine scale operator K_h can be done with many methods. In this work, the Galerkin method was used because of its purely alge-

braic nature. In this method, the coarse scale operator K_H is called Galerkin operator and is computed by $K_H = RK_hP$ where R and P denote the restriction and prolongation operators respectively. The common AMG practice is that the prolongation operator is chosen first and the restriction operator is adjusted to this choice. R and P should be linear transformations so that they can be represented in a matrix form. Moreover, it is assumed that R and P have full rank, which means that P should have linearly independent columns and R linearly independent rows. Finally, it should be $RP = I$, meaning that there is an adequate mapping from the fine scale to the coarse scale and conversely [9].

Given a known fine scale operator K_h, the Galerkin method allows for the immediate calculation of the coarse scale operator K_H. In the case K_H is known and K_h unknown, then K_h is a sparse matrix with some unknown variables. As we explain in the next section, the matrices K_H and K_h have a specific form. Thus, the matrix equation $K_H = RK_hP$ leads to a linear system of equations. If the size of the convolution kernels s_H and s_h is the same, then the system has a unique solution [4] but it will be shown that this is not always the case.

As explained above, the approach to AMG in this case follows a different path than in more traditional applications because the purpose is not to downscale the operator in order to solve a linear system but to downscale the operator for its own sake. In the traditional applications of AMG, the coarser grids are chosen freely so that the system becomes easier to solve while in the case presented here the different grids are a given to the problem; the purpose is to scale a CNN to a specific grid. Moreover, AMG includes considerations of other factors such as the error that appears between the solution that the method converges to and the actual solution of the system. This error also needs to be approximated to the coarser levels in an appropriate way to achieve convergence. This is the role of a smoothing operator which usually drives the choice of the intergrid operators R and P [9]. However, this does not seem to be relevant in the present case as it is not the unknown u that should be approximated.

The lack of error convergence considerations drove us to a different approach in regards to the choice of the intergrid operators. Although the Galerkin method is purely algebraic, its application to a specific case gives a physical meaning to the intergrid operators R and P. In order to comprehend this physical meaning, it is useful to look into the derivation of the Galerkin operator [4]. The restriction operator restricts a fine scale signal u_h to a coarse scale signal u_H through the relation $u_H = Ru_h$ while the prolongation operator prolongs the coarse scale signal u_h to the interpolated fine scale signal $\overline{u_h}$. When $RP = I$, then $\overline{u_h} = u_h$. Let w_h the output signal of applying a fine scale convolution operating on the fine scale signal $w_h = K_h u_h$. Then, we have $w_h = K_h P u_H$. Now, we want to construct a coarse scale convolution K_H operating on the coarse scale signal $w_H = K_H u_H$ which is consistent with applying K_h on u_h. Namely, we want to restrict the convoluted coarse scale signal so that $w_H = Rw_h \Rightarrow K_H u_H = RK_h P u_H$. Therefore, one way to construct the coarse scale operator is by $K_H = RK_hP$.

What should be taken out of this process is that R and P should be understood as restriction and prolongation applied to a signal, which in the cases examined is the input to the CNNs, that is audio recordings. Namely, R and P are in fact resampling methods that should follow the restrictions outlined above. In terms of signal processing, the restriction that $RP = I$ means that after upsampling the signal and then downsampling again, we should return to the original. The restriction that both matrices should have full rank means that all the points of the signal should be taken into account when resampling. This physical interpretation of the operators inspired the choices that were made and that are explained below.

2.2 Convolution as Matrix Multiplication

Before proceeding to the specific prolongation and restriction methods, it should be explained that applying a convolution to an array can be considered as a matrix multiplication in which

the convolution is represented by a sparse Toeplitz matrix [1].

In the case of one-dimensional input that was examined, the Toeplitz matrix is constructed so that the first column starts with the convolution kernel and is completed with zeros while the first row starts with the first element of the convolution and is completed with zeros. The rest of the matrix is completed so that each descending diagonal from left to right is constant.

In the neural networks that are examined, zero padding in the borders of the signal is implemented so that the convoluted output has the same length as the input. Then, the matrix should be diagonal $n \times n$ with n the length of the signal. Moreover, the first column starts with the middle element of the kernel and is completed as described above.

Since we deem it important to be consistent with the AMG bibliography, we should apply the convolution as a matrix multiplication from the left of the signal. Therefore, we used the transpose of the Toeplitz matrix in the applications. From now on, when the matrix representation of a convolution kernel is referred, it is implied the transpose of the zero padded Toeplitz matrix.

So, assuming we have a kernel $s = [x_1, x_2, \ldots, x_m]$ where m is an odd number (as commonly used in CNNs), then the Matrix is:

$$K = \begin{pmatrix} x_{\frac{m+1}{2}} & x_{\frac{m+1}{2}+1} & x_{\frac{m+1}{2}+2} & \cdots & x_m & 0 & \cdots & 0 \\ x_{\frac{m+1}{2}-1} & x_{\frac{m+1}{2}} & x_{\frac{m+1}{2}+1} & \cdots & x_{m-1} & x_m & 0 & \vdots \\ \vdots & x_{\frac{m+1}{2}-1} & \ddots & \ddots & \ddots & & & \vdots \\ x_1 & \ddots & & \ddots & \ddots & \ddots & & x_m \\ 0 & & \ddots & \ddots & \ddots & \ddots & & x_{m-1} \\ \vdots & \ddots & & \ddots & & & & \vdots \\ \vdots & & \ddots & \ddots & & & & \vdots \\ 0 & \cdots & \cdots & x_1 & \cdots & & x_{\frac{m+1}{2}-1} & x_{\frac{m+1}{2}} \end{pmatrix}$$

In the case m is even, then the middle element is $\frac{m}{2}$. This is rarely used in CNNs but it is relevant when a fully connected layer is represented as a convolutional layer as we shall see.

It is important to note that the matrix K presented above is the matrix representation of a kernel that is applied without the use of strides, or with $stride = 1$, as in the case of the networks examined in this study. The case of strided kernels will be briefly examined in the case of the average pooling layer below as well as in Appendix A.

It should be noted that, in CNNs, applying a convolution to a signal includes an activation function. However, not all activation functions are linear, meaning that they cannot be represented as a matrix operation. In fact, more often than not, the activation functions are not linear and so is the case in the CNNs that will be presented in the practical implementation. In the methods explored below, we shall deal with the weights of the kernels, leaving the activation functions as part of the architecture which remains unchanged.

2.3 Prolongation and Restriction Methods

As mentioned above, the success of the AMG depends on the choice of prolongation and restriction methods. In this section, the combinations of intergrid operators that were used are presented.

2.3.1 Nearest Neighbor Interpolation

Nearest neighbor is one of the simplest interpolation methods and entails upsampling the signal by interpolating the nearest sample [10]. The basic idea of the method is shown in Figure 1. Nearest neighbor interpolation can be used to prolong signals by integer factors.

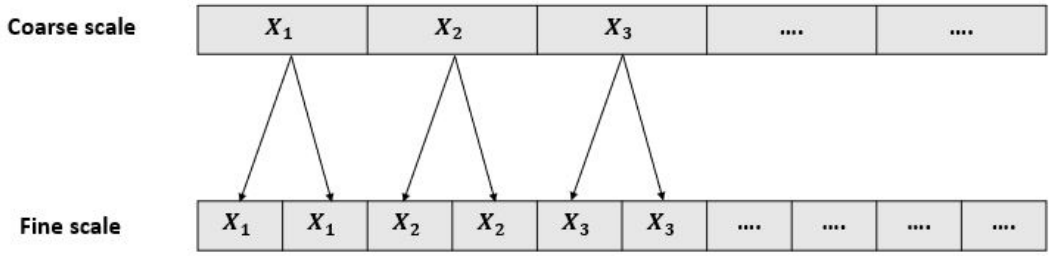

FIGURE 1: Nearest Neighbor Method

Let a one-dimensional array $u_H = [x_1, x_2, \ldots, x_m]^T$ that is to be upscaled to a new finer scale h by an integer factor $N = \frac{h}{H}$. Then, the length of u_h will be Nm and we have:

$$u_h = [\underbrace{x_1, \cdots, x_1}_{N}, \underbrace{x_2, \cdots, x_2}_{N}, \ldots, \underbrace{x_m, \cdots, x_m}_{N}]^T$$

2.3.1.1 Prolongation Matrix

The prolongation matrix P is the transformation matrix of the linear transformation that performs the operation described above. By following well-known methods of linear algebra, the columns of P are the images of the normal basis of $\mathbb{R}^m$ (see Appendix B) [11]. The dimensions of the matrix are $Nm \times m$. It can be seen that the columns of P are linearly independent. Therefore, the matrix has full rank.

$$P = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & 1 & \cdots & \vdots \\ 0 & \vdots & \cdots & 0 \\ \vdots & 1 & \cdots & 1 \\ 0 & 0 & \cdots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

2.3.1.2 Restriction Matrix

Since P is not square and hence not invertible, there is not a unique matrix R that complies to the requirement $RP = I$. The most intuitive choice is to restrict by averaging the neighboring sample points. The shape of the matrix is $m \times Nm$. We can see that the rows are linearly independent. Last but not least, the requirement $RP = I$ is also fulfilled.

$$R = \begin{pmatrix} N^{-1} & \cdots & N^{-1} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & N^{-1} \cdots & N^{-1} & 0 & \cdots & \\ \vdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & \underbrace{N^{-1}}_{N} & \cdots & N^{-1} \end{pmatrix}$$

2.3.1.3 Multiscaling

Let a fine scale kernel $s_h = [x_1, \cdots, x_m]$ and a coarse scale kernel $s_H = [y_1, \cdots, y_k]$ with K_h and K_H their matrix representations. Moreover, let a coarse scale signal with size n and the prolonged fine scale signal with size $2n$. Since $RP = I$, the relation can also be seen in reverse where the coarse scale signal is the restriction of the fine scale one. The two kernels are connected by the relation $K_H = RK_hP$, as explained above, where R and P are the restriction and prolongation operators respectively. In the cases examined here, the scaling ratio was $N = 2$. However, the nearest neighbor interpolation can be applied for any integer ratio.

The matrices K_h and K_H have a specific form as explained above. In addition, it can be proven that multiscaling should preserve the strides of the kernel (see Appendix A). Therefore, the equation $K_H = RK_hP$ can be solved in the general form for kernels of arbitrary length, something which returns a set of formulas that relate the weights of the fine scale kernel x_i with the weights of the coarse scale kernel y_i. It can be shown that there are four cases depending on $max(m,k)$.

From the way a convolution kernel is applied in CNNs, it can be seen that padding any number of zeros at the borders of the kernel does not change the operation. This means that, in the case $k \neq m$, the smallest kernel can be extended by zero padding so that $k = m$. Therefore, in the general case it is assumed that $k = m$, namely that the coarse and fine scale kernels have the same length. If it turns out that one kernel is smaller than the other, this will become apparent in the formulas as the first and/or the last elements of the smaller kernel will be zero. It turns out that this is indeed the case; the coarse scale kernel is in fact smaller than the fine scale one for $k, m > 3$ which includes all the kernels of interest (kernels with length less than three were not effective as we shall see when discussing the practical implementation). This result is repeated in all the methods that were examined and has important consequences in the case of upscaling because it means that there is no unique fine scale convolution given a coarse scale one.

In the formulas that are presented below, the length $k < m$ of the coarse scale kernel corresponds to the length after subtracting all the consecutive zero elements from the borders that appear in the formulas if equal length is assumed. Moreover, a relation between the lengths of the kernels is also given.

Given a fine scale kernel, a unique coarse scale kernel can be computed using these formulas by substituting the known x_i's. However, given a coarse scale kernel, the substitution of the known y_i's to the formulas returns a linear system with more unknown x_i's than equations. In particular, there are k equations for m unknowns with $k < m$. By examining the coefficient matrices of the systems in all cases, it can be shown that they have an infinite solution set.

When m is odd

When $\frac{m+1}{2}$ is even
Then $k = \frac{m+3}{2}$ and:

$$\begin{aligned} y_1 &= \frac{1}{2}x_1 \\ y_2 &= \frac{1}{2}x_1 + x_2 + \frac{1}{2}x_3 \\ &\vdots \\ y_{k-1} &= \frac{1}{2}x_{m-2} + x_{m-1} + \frac{1}{2}x_m \\ y_k &= \frac{1}{2}x_m \end{aligned}$$

When $\frac{m+1}{2}$ is odd
Then $k = \frac{m+1}{2}$ and:

$$\begin{aligned} y_1 &= x_1 + \frac{1}{2}x_2 \\ y_2 &= \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_4 \\ &\vdots \\ y_{k-1} &= \frac{1}{2}x_{m-3} + x_{m-2} + \frac{1}{2}x_{m-1} \\ y_k &= \frac{1}{2}x_{m-1} + x_m \end{aligned}$$

When m is even

When $\frac{m}{2}$ is even
Then $k = \frac{m}{2} + 1$ and:

$$\begin{aligned} y_1 &= x_1 + \frac{1}{2}x_2 \\ y_2 &= \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_4 \\ &\vdots \\ y_{k-1} &= \frac{1}{2}x_{m-2} + x_{m-1} + \frac{1}{2}x_m \\ y_k &= \frac{1}{2}x_m \end{aligned}$$

When $\frac{m}{2}$ is odd
Then $k = \frac{m}{2} + 1$ and:

$$\begin{aligned} y_1 &= \frac{1}{2}x_1 \\ y_2 &= \frac{1}{2}x_1 + x_2 + \frac{1}{2}x_3 \\ &\vdots \\ y_{k-1} &= \frac{1}{2}x_{m-3} + x_{m-2} + \frac{1}{2}x_{m-1} \\ y_k &= \frac{1}{2}x_{m-1} + x_m \end{aligned}$$

2.3.2 Linear Interpolation

There are many definitions of linear interpolation. In this work, it was defined as the interpolation between two sample points [12]. The method is explained visually in Figure 2.

Let a discrete signal $u_H = \left[x_1, x_2, \ldots, x_m\right]^T$ that is to be upsampled to a new signal u_h by a factor $N = 2$. Then, the length of u_h will be $2m$ and we have $u_h = \left[x_1, \frac{x_1+x_2}{2}x_2, \ldots, x_m, \frac{x_m}{2}\right]^T$. As shown in Figure 2, the first point of the prolonged signal is the same as the first of the coarse scale signal. However, when it comes to the last point of the prolonged signal, there is no point "to the right" in order to interpolate. It was chosen to deal with this by padding a zero as a new element in the coarse scale signal and calculate the last element of the fine scale signal by $\frac{x_m+0}{2}$.

This method can be used to upscale by a factor of 2. Iteratively, it can be used to upscale by a factor 2^j, with j the number of iterations.

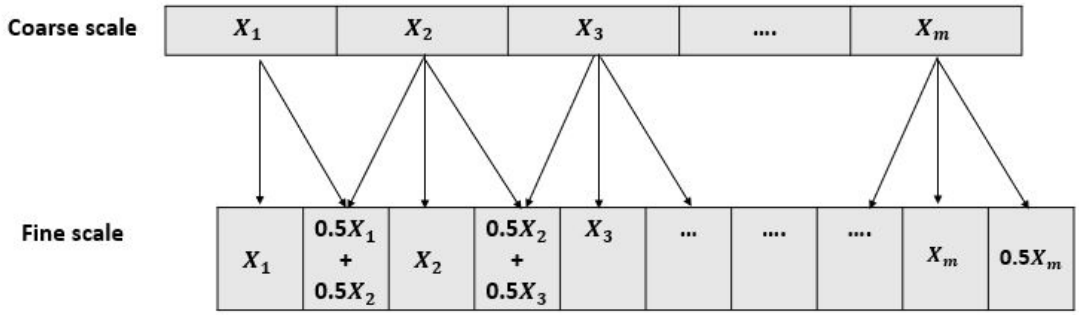

FIGURE 2: Linear Interpolation Method

2.3.2.1 Prolongation Matrix

The matrix representation of this method is calculated by using methods of linear algebra as explained in the case of nearest neighbor (see Appendix B). We can verify that the matrix has full rank as it is required by the AMG method.

$$P = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \frac{1}{2} & \frac{1}{2} & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \frac{1}{2} & \frac{1}{2} & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \frac{1}{2} \\ 0 & 0 & \cdots & 1 \\ 0 & \cdots & 0 & \frac{1}{2} \end{pmatrix}$$

We can see that the first row and column of the matrix do not follow the same pattern with the last row and column. The reason for this is the different treatment of the border elements of the array that was presented in Figure 2. When applying the Galerkin method $K_H = RK_hP$, this discrepancy creates an inconsistent system of equations. In particular, there are two inconsistent formulas for some coarse scale weights. In order to solve this problem, it was decided to ignore the first row and column of P when multiscaling. The physical meaning of this choice is that the upscaled signal has one element less, the first element is missing (equivalently, it could be chosen that the last element would be ignored). Since the signals are in the range of tens of thousands, this is not expected to have a great influence on the whole method.

2.3.2.2 Restriction Matrix

There are many possible matrices to perform the opposite transformation. The range of choices is restricted by the fact that the rows should be linearly independent.

Let the coarse scale signal be $u_H = [a_1, a_2, \cdots a_m]$ and the interpolated $u_h = [A_1 = a_1, A_2 = \frac{a_1+a_2}{2}, A_3 = a_3, \cdots A_{2m-3} = a_{m-1}, A_{2m-2} = \frac{a_{m-1}+a_m}{2}, A_{2m-1} = a_m, A_{2m} = \frac{a_m}{2}]$. One solution to the problem of restriction is the formula $a_i = 2A_{2i} - A_{2i+1}, 1 \leq i \leq m$, with $A_{2m+1} = 0$. It can be shown that the corresponding matrix R has full rank. The shape of the matrix is $m \times 2m$. It is not difficult to verify that $RP = I$.

$$R = \begin{pmatrix} 0 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -1 & 0 & \cdots & 0 \\ \vdots & 0 & 0 & \cdots & 0 & 2 & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Following the same reasoning as in the case of the prolongation matrix, the first column and row are subtracted from the matrix when multiscaling since this would create a matrix with a distinct pattern while keeping the requirement that $RP = I$.

2.3.2.3 Multiscaling

By following the same process that was explained in the section of the nearest neighbor, we can derive the formulas that relate the weights of the coarse scale kernel $s_H = [y_1, \cdots, y_k]$ and the fine scale kernel $s_h = [x_1, \cdots, x_m]$. Once more, we shall see that given a fine scale kernel it is possible to calculate a unique coarse scale one while in the opposite case we have an underdetermined system of linear equations with an infinite solution set. There are again four cases depending on the size of the fine scale kernel m.

When m is odd

When $\frac{m+1}{2}$ is even
Then $k = \frac{m+3}{2}$ and:

$$\begin{aligned} y_1 &= \frac{3}{2}x_1 + x_2 \\ y_2 &= x_4 + \frac{3}{2}x_3 - \frac{1}{2}x_1 \\ &\vdots \\ y_{k-2} &= x_{m-1} + \frac{3}{2}x_{m-2} - \frac{1}{2}x_{m-4} \\ y_{k-1} &= \frac{3}{2}x_m - \frac{1}{2}x_{m-2} \\ y_k &= -\frac{1}{2}x_m \end{aligned}$$

When $\frac{m+1}{2}$ is odd
Then $k = \frac{m+3}{2}$ and:

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_3 + \frac{1}{2}x_2 \\ y_3 &= x_5 + \frac{3}{2}x_4 - \frac{1}{2}x_2 \\ &\vdots \\ y_{k-1} &= x_m + \frac{3}{2}x_{m-1} - \frac{1}{2}x_{m-3} \\ y_k &= -\frac{1}{2}x_{m-1} \end{aligned}$$

When m is even

When $\frac{m}{2}$ is even
Then $k = \frac{m}{2} + 2$ and:

$$
\begin{aligned}
y_1 &= x_1 \\
y_2 &= x_3 + \frac{1}{2}x_2 \\
y_3 &= x_5 + \frac{3}{2}x_4 - \frac{1}{2}x_2 \\
&\vdots \\
y_{k-2} &= x_{m-1} + \frac{3}{2}x_{m-2} - \frac{1}{2}x_{m-4} \\
y_{k-1} &= \frac{3}{2}x_m - \frac{1}{2}x_{m-2} \\
y_k &= -\frac{1}{2}x_m
\end{aligned}
$$

When $\frac{m}{2}$ is odd
Then $k = \frac{m}{2} + 1$ and:

$$
\begin{aligned}
y_1 &= \frac{3}{2}x_1 + x_2 \\
y_2 &= x_4 + \frac{3}{2}x_3 - \frac{1}{2}x_1 \\
&\vdots \\
y_{k-1} &= x_m + \frac{3}{2}x_{m-1} - \frac{1}{2}x_{m-3} \\
y_k &= -\frac{1}{2}x_{m-1}
\end{aligned}
$$

2.3.3 Inverse Distance Weighting

The third multiscale method that was explored in the paper was inverse distance weighting. In this technique, the points of the fine scale signal are produced by the weighted average of the two closest points in the coarse grid [10]. In contrast to the previous two methods, the points of the coarse scale signal are not a subset of the points of the fine scale one.

In Figure 3, it is shown how the distances are defined when upscaling by a factor of 2. The method can be theoretically used to upscale for any ratio, even non-integer ones. Every box represents one sample point and it has a center. The distance is measured from the midpoint of the interpolated sample to the midpoints of the two closest coarse scale samples. The unit of distance is half the length of the interpolated points. The value of the new sample is calculated as the weighted average of the two original samples, where the weights are the distances.

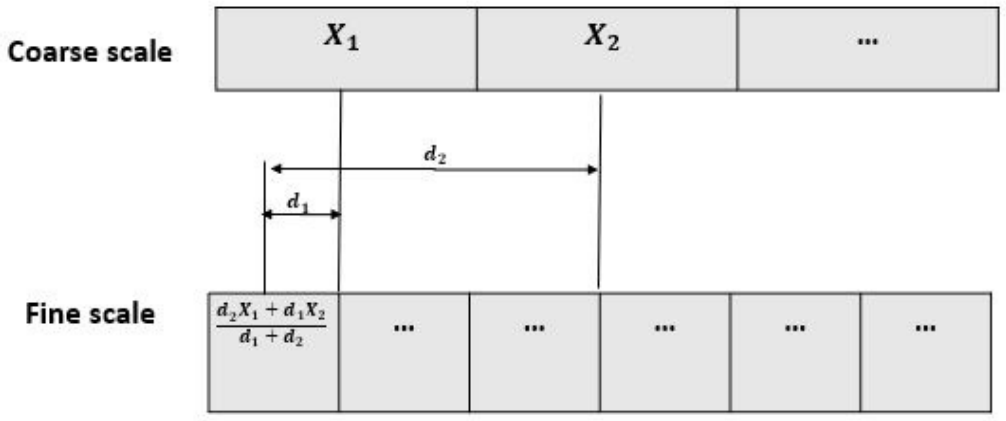

FIGURE 3: Inverse Distance Weighting Method

2.3.3.1 Prolongation Matrix

The matrix representation of this prolongation approach is presented below (see Appendix B for how it was derived). It can be proven to have full rank.

$$P = \begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 & \cdots & \cdots & 0 \\ \frac{3}{4} & \frac{1}{4} & 0 & \cdots & \cdots & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & \cdots & \cdots & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & \ddots & \ddots & \ddots \\ 0 & \frac{1}{4} & \frac{3}{4} & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & \cdots & \frac{3}{4} & \frac{1}{4} & 0 \\ \vdots & \cdots & \cdots & \frac{1}{4} & \frac{3}{4} & 0 \\ \vdots & \cdots & \cdots & 0 & \frac{3}{4} & \frac{1}{4} \\ \vdots & \cdots & \cdots & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & \cdots & \cdots & 0 & \frac{1}{6} & \frac{5}{6} \end{pmatrix}$$

The first and last row of P follow a different pattern. This is because the first and last element of the interpolated signal should be calculated using coarse scale sample points that are further than the points in the middle of the signal. This is similar to the case of the last element in the linear interpolation. When P is used in the Galerkin method, the difference in these two rows created an inconsistent system of equations. Therefore, it was decided that they should not be taken into consideration. The physical meaning of this choice is that the two end points of the signal are not interpolated. Since the length of the signals at hand is in the range of tens of thousands, it is not expected to cause problems.

2.3.3.2 Restriction Matrix

Once more, there are many possibilities in picking a restriction matrix that performs the inverse transformation. In a manner similar to the case of the linear interpolation the choice was:

$$R = \begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & \frac{3}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & -\frac{1}{2} & \frac{3}{2} & 0 & 0 \end{pmatrix}$$

R conforms to the restrictions set by AMG. It should be noted that R was chosen to correspond to P after the deletion of the first and last column.

2.3.3.3 Multiscaling

Once more, following the Galerkin method we can derive the formulas that relate the weights of the coarse scale kernel $s_H = [y_1, \cdots, y_k]$ with those of the fine scale kernel $s_h = [x_1, \cdots, x_m]$. Again, we see that given a fine scale kernel we can find a unique coarse scale kernel while the opposite is not the case; there are infinite solutions to the problem of finding a fine scale kernel given a coarse scale one. There are four cases depending on the size of the fine scale kernel m.

When m is odd

When $\frac{m+1}{2}$ is even
Then $k = \frac{m+5}{2}$ and:

$$
\begin{aligned}
y_1 &= -\frac{1}{8}x_1 \\
y_2 &= -\frac{1}{8}x_3 + \frac{3}{4}x_1 \\
y_3 &= -\frac{1}{8}x_5 + \frac{3}{4}x_3 + x_2 + \frac{3}{8}x_1 \\
&\vdots \\
y_{k-2} &= -\frac{1}{8}x_m + \frac{3}{4}x_{m-2} + x_{m-3} + \frac{3}{8}x_{m-4} \\
y_{k-1} &= \frac{3}{4}x_m + x_{m-1} + \frac{3}{8}x_{m-2} \\
y_k &= \frac{3}{8}x_m
\end{aligned}
$$

When $\frac{m+1}{2}$ is odd
Then $k = \frac{m+3}{2}$ and:

$$
\begin{aligned}
y_1 &= -\frac{1}{8}x_2 \\
y_2 &= -\frac{1}{8}x_4 + \frac{3}{4}x_2 + x_1 \\
y_3 &= -\frac{1}{8}x_6 + \frac{3}{4}x_4 + x_3 + \frac{3}{8}x_2 \\
&\vdots \\
y_{k-2} &= -\frac{1}{8}x_{m-1} + \frac{3}{4}x_{m-3} + x_{m-4} + \frac{3}{8}x_{m-5} \\
y_{k-1} &= \frac{3}{4}x_{m-1} + x_{m-2} + \frac{3}{8}x_{m-3} \\
y_k &= x_m + \frac{3}{8}x_{m-1}
\end{aligned}
$$

When m is even

When $\frac{m}{2}$ is even
Then $k = \frac{m}{2} + 2$ and:

$$
\begin{aligned}
y_1 &= -\frac{1}{8}x_2 \\
y_2 &= -\frac{1}{8}x_4 + \frac{3}{4}x_2 + x_1 \\
y_3 &= -\frac{1}{8}x_6 + \frac{3}{4}x_4 + x_3 + \frac{3}{8}x_2 \\
&\vdots \\
y_{k-2} &= -\frac{1}{8}x_m + \frac{3}{4}x_{m-2} + x_{m-3} + \frac{3}{8}x_{m-4} \\
y_{k-1} &= \frac{3}{4}x_m + x_{m-1} + \frac{3}{8}x_{m-2} \\
y_k &= \frac{3}{8}x_m
\end{aligned}
$$

When $\frac{m}{2}$ is odd
Then $k = \frac{m}{2} + 2$ and:

$$
\begin{aligned}
y_1 &= -\frac{1}{8}x_1 \\
y_2 &= -\frac{1}{8}x_3 + \frac{3}{4}x_1 \\
y_3 &= -\frac{1}{8}x_5 + \frac{3}{4}x_3 + x_2 + \frac{3}{8}x_1 \\
&\vdots \\
y_{k-2} &= -\frac{1}{8}x_{m-1} + \frac{3}{4}x_{m-3} + x_{m-4} + \frac{3}{8}x_{m-5} \\
y_{k-1} &= \frac{3}{4}x_{m-1} + x_{m-2} + \frac{3}{8}x_{m-3} \\
y_k &= x_m + \frac{3}{8}x_{m-1}
\end{aligned}
$$

2.4 Pooling Layers

There are two types of pooling layers that are used in CNNs: max pooling and average pooling [13].

2.4.1 Average Pooling

There are two ways to interpret an average pooling layer. In the first interpretation, the pooling layer is interpreted as a way to reduce the size of the representation and consequently to reduce the learned parameters[1]. Following this line of thought, a multiscaling algorithm would treat the pooling layers as part of the architecture, as it does with the number of layers and the number of nodes per layer. The algorithms that were examined in this work left the architecture unaffected and focused on scaling the learned parameters. Hence, the pooling layers would also remain unaffected.

In the second approach, an average pooling layer can be seen as a fixed strided convolution kernel s_p that operates on the data. As a matter of fact, an average pooling layer of size n is equivalent to a strided convolutional layer with $stride = n$ and kernel

$$k = [\underbrace{n^{-1}, n^{-1}, \cdots, n^{-1}}_{\text{n}}]$$

The activation function is the unit function and no zero padding in the borders is used. Moreover, the application of this operator to a signal of size N can be represented as a matrix multiplication from the left with the matrix K_p with dimensions $\frac{N}{n} \times N$, where $\frac{N}{n}$ is rounded down when it is not an integer. In the case of an average pooling layer of size 2, as is the case in the CNNs that were used, the matrix is:

$$K_p = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Therefore, when this layer is analyzed from the multigrid perspective, there is no restriction in applying the same multiscale methods that were used in the convolutional layers. That is, let s_p be the kernel applied to the fine scale data and s_P the kernel for the coarse scale. Then, with R and P the restriction and prolongation matrices that were explained above, we have $K_P = RK_pP$. However, since the application of the kernel reduces the size of the data, the dimensions of the restriction and prolongation matrices should be adjusted accordingly.

2.4.1.1 Downscaling

Applying the Galerkin method in the average pooling layer with size 2, we have:

Nearest neighbor interpolation: The kernel remains the same when downscaling.

Linear interpolation: When downscaled, the average pooling layer is transformed to a convolutional layer with $stride = 2$ and kernel $[\frac{3}{2}, -\frac{1}{4}, -\frac{1}{4}]$, no zero padding in the borders.

Inverse Distance Weighting: When downscaled, the average pooling layer is transformed to a convolutional layer with $stride = 2$ and kernel $[-\frac{1}{4}, \frac{1}{2}, \frac{3}{4}]$, no zero padding in the borders.

2.4.2 Upscaling

Nearest neighbor interpolation: The new kernel is a convolution with $stride = 2$ and $s_p = [a, 1-a], a \in R$. We see that when upscaling, there is no unique solution. However, the original kernel $s_P = [\frac{1}{2}, \frac{1}{2}]$ is part of the solution set.

Linear interpolation: The system that arises from the matrix equation $K_P = RK_pP$ is inconsistent.

Inverse Distance Weighting: The system that arises from the matrix equation $K_P = RK_pP$ is again inconsistent.

2.4.3 Max Pooling

A max pooling layer performs a non-linear transformation to the input. The max function cannot be represented in a matrix form. Consequently, when multiscaling a max pooling layer was left unchanged.

2.5 Dense Layers

A dense (fully connected) layer can be viewed as a convolutional layer with kernel length equal to the length of the input. Then, the same multiscaling methods that were applied in the convolutional layers can also be applied to the fully connected layers.

The resulting "kernel" should then have length equal to the new input signal (bigger when upscaling and smaller when downscaling). In the cases examined above, the scaling ratio was set to 2, which means that the upscaled dense layer "kernels" should have twice the length in the case of upscaling and half in the case of downscaling.

As it was presented above, when a kernel of length m is downscaled, the new kernel has length $\frac{m}{2} + c_1$ where the small constant c_1 depends on the method. Similarly, in the case of upscaling the new kernel has length $2m + c_2$. In all the cases, we had with $-3 \leq c_1, c_2 \leq 3$. Therefore, the length of the multiscaled "kernel" of the dense layer has length equal to the size of the input signal plus or minus a small integer. This constants means that the new multiscaled fully connected layer will either be slightly smaller or slightly bigger than the new input length.

In the case it is larger, the "convolution kernel" has length larger than the input in which it should be applied, so it was applied by ignoring the extra weights as it would be if it was indeed a convolutional layer.

In the case it is smaller, we have a more traditional application of a convolution kernel to a signal. However, hardware limitations did not allow for that implementation. Instead, an appropriate number of zeros was padded at the end of the "kernel" to reach the appropriate length. Considering that the input size to the dense layer, and therefore the number of weights of the dense layer, is on the scale of tens of thousands while the constants c_1, c_2 are small, the effect of this choice is expected to be small.

2.6 Direct Kernel Prolongation

The main approach to the topic of multiscaling followed in this study was based on AMG. However, as it was presented above, this strategy failed to return a unique weight configuration in the case of upscaling. This led to the consideration of a different strategy.

As it was explained in the relevant section, AMG focuses on constructing an operator in a new grid that is consistent with applying a known operator in the original grid. This is done by changing the convolution kernels in a way that incorporates the signal resamlping. In other words, the prolongation and restriction operators should be thought as being implicitly applied to the signal by becoming part of the new kernels.

A different approach to upscaling would be to explicitly prolong the convolutional operators. Let $w_H = K_H u_H$ the convoluted signal in the coarse scale. Now we set out to prolong w_H itself in order to arrive to a finer scale convoluted signal $w_h = Pw_H = PK_H u_H$. Therefore, the fine scale operator could be calculated as $K_h = PK_H$.

On the one hand, this strategy lacks the rigid mathematical foundation of AMG. In particular, the main deficiency of the strategy is that it prolongs the coarse scale operator so that it can be applied to the fine scale signal without accounting for an adequate mapping back to the coarse scale. In mathematical terms, there is no restriction operation R with $RP = I$ included in this method. On the other hand, this method ensures that there is always a unique prolonged kernel for each coarse scale one.

Practically, this strategy entailed directly applying the three interpolation methods that were explained above to the coarse scale kernels. In addition, a fourth method was included in the experiments, that of dilating the kernels. Dilating can be applied by interpolating zeros between the weights as shown in Figure 4; it is a strategy that is frequently used while training CNNs. All these methods produce fine scale kernels with twice the length of the coarse ones when applied in the case of scaling with ratio $N = 2$, as in the experimental design of this study.

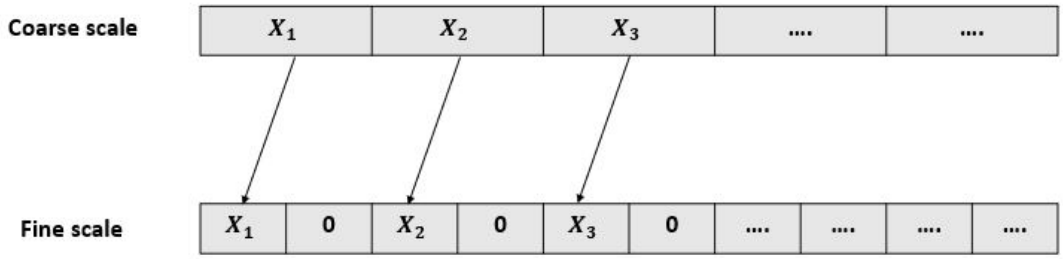

FIGURE 4: Dilating a convolution kernel

3 Practical Implementation

3.1 The dataset

The dataset consists of audio recordings from five devices that can be found in an average household or around it, they are sound sources that most people would recognize as familiar. These five classes are: a vacuum cleaner, a microwave oven, a truck, a sewing machine and a mixer. The duration of each audio track is 4 seconds and they were recorded using the "MP3 Recorder" app for Android [14] in the WAV file audio format with 320 Kbps bitrate and 48 kHz sampling rate (the highest quality possible).

In general, the sound produced by these devices is largely repetitive. Although they are generally distinct to a human ear, they also share similarities. The main sound source in each class was some moving parts (even in the microwave the main sound source was the ventilation). Additionally, with the exception of the truck which used a diesel internal combustion engine, the source of motion was an electric motor. These similarities created a challenge to the neural network.

In order to create some variability within the tracks that belong to the same class, the recordings were conducted from various distances from the source and the machines were used in different intensities. The recordings took place in relative sound insulation to avoid noise in the dataset.

The dataset consists of 750 tracks, 150 tracks for each class. The tracks are equally divided between classes to prevent the dataset from becoming biased. The training set consisted of 650 audio tracks (130 tracks from each class) and the testing set of 100 tracks (20 per class). The training testing split was done randomly and after shuffling the trucks. Moreover, a part of the

training set was used as a validation set during training. The training-testing-validation split was 70/15/15, 70% of the dataset was the training set while the testing and validation sets were 15% each, which is a common rule of thumb for small datasets and the default split used in MATLAB [15].

The dataset was downsampled to two different datasets, one with 12 kHz sampling rate and one with 24 kHz. Hardware limitations did not allow for higher sampling rates to be used in training the neural networks. The choice of 12 kHz and 24 kHz was made so that the different sampling rates have an integer ratio because this would make the implementation of the multiscale methods simpler. The downsampling was done using the librosa Python library for audio signal analysis [16]. The default resampling method in librosa is Kaiser's best [17].

The testing sets will have a dual function in this work, the usual function of testing the efficacy of the training and comparing the success of the multiscaling methods. In order to eliminate the possibility that the differences depend on the testing set variability, the training and testing data in the two datasets (12 and 24 kHz) consist of exactly the same audio recordings in different sampling rates.

3.2 The convolutional neural networks

Ten one-dimensional CNNs were developed in order to test the different multiscale techniques, five for the 12 kHz sampling rate dataset and five for the 24 kHz one. They were developed using the Keras API [13] with TensorFlow backend [18]. The architectures of the CNNs trained on the 12kHz dataset are shown in Appendix C and on the 24kHz dataset in Appendix D.

In order to determine the appropriate architecture of the CNNs, a heuristic iteration over the parameter space was performed. The parameters that were fine-tuned with this approach were: the number and size of the convolutional layers, the kernel size, the size and type of the pooling layer, the dropout rate, the number and size of the dense layers, the activation functions, the batch size and the number of training epochs. Hardware limitations also affected the choice of architecture. The optimal networks were chosen based on the categorical accuracy metric.

For the sake of facilitating the implementation of the multiscale methods, some simplifications were made. All the convolutional layers of each CNN were formulated with equal kernel size and with the same activation function. The convolutions were applied with zero padding in the borders to make the length of the output of each layer equal to the input. The biases were ignored in the multiscale process.

With a view to preventing overfitting, dropout was used during training with a rate that was determined by the heuristic described above. This was important because the dataset was rather small. Since it was only used during training, it will not be considered in the phase of multiscaling which is conducted post-training. In all cases, the Adam optimization algorithm was used because it is computationally efficient and it has low memory requirements [19]. The loss function was categorical cross-entropy which is commonly used in classification tasks.

4 Results and Discussion

The different multiscale methods that were explained in the methodology section were implemented in the target CNNs and the resulting networks were tested using the appropriate testing set. In particular, the 24kHz CNNs were downscaled and tested with the 12kHz testing set and conversely for the 12kHz CNNs.

As it is explained by Stuben [8], the implementation of an AMG method can take a considerable amount of human effort but the computational complexity is small. The human ef-

fort consists of formulating the appropriate prolongation and restriction strategies and solving large sparse systems of linear equations. In this work, the Galerkin method was applied for the general form of the fine and coarse scale operators (Toeplitz matrices) and a number of formulas were extracted that connect the weights of the fine scale kernels to the weights of the coarse scale kernels.

Given the fine scale kernels, the unique corresponding coarse scale kernels can be computed by substitution to these formulas. Therefore, the computational cost of the implementation is negligible. However, given the coarse scale kernels, the fine scale ones are the solution of an underdetermined system with an infinite solution set. In order to determine a particular numerical solution to this problem, the method of least squares was used [20].

In order to deal with the problems arising from the lack of a unique solution when applying AMG in upscaling, the kernels were directly prolonged using the three interpolation methods that were explained above as well as the method of kernel dilation.

As a baseline of comparison, the CNNs that was trained on the 24kHz dataset were tested for the 12kHz dataset and conversely without the application of any multiscale method. In this case, the dense layer was treated as a convolutional layer.

The only other related work that these results can be compared with is the work of Haber et al [4]. However, such a comparison may not be appropriate because they focused on image classification networks with larger architectures and datasets. Moreover, they did not use the three interpolation methods described in this study.

4.1 Downscaling

The dual interpretation of the pooling layers gave rise to two different cases in downscaling the kernels. In the first approach, the average pooling was viewed as a strided convolutional layer and was downscaled as such while in the second it was viewed as part of the architecture and left unaffected. In the networks that used max pooling, there was only one choice, to leave the layers unaffected . The categorical accuracy rates of the CNNs over the 12kHz testing set are shown in Table 1. For the sake of comparison, the accuracy over the 24kHz dataset for which they were trained is also presented in parentheses beside the model name.

The first noticeable result is that the accuracy of the CNNs in which the average pooling layers remained unaffected was higher. In particular, when downscaled using linear interpolation or inverse distance weighting, the accuracy was around 20%. Since there are five classes, a result in the neighborhood of 20% indicates random prediction. In the case of the nearest neighbor method, the result is the same in the two cases, something which was expected as this method leaves the average pooling layer unaffected when applied to it.

This result does not support the interpretation of average pooling as a convolutional layer. This explanation seems plausible as, in contrast to convolutional and fully connected layers, the pooling layers do not contain learned parameters which means that they cannot be trained to detect features of the input.

Interestingly, when the CNNs were not changed, the accuracy rate on the 12kHz testing set was high (15% to 20% lower than the accuracy in the original 24kHz dataset), meaning that the CNN configuration is transferable across different scales. This puts the bar of success of a downscale method high since for a strategy to be considered useful it should at least perform better than applying no method at all. It should be noted that in the work of Haber et al. in image classification CNNs, the original architecture also gave a high accuracy rate [4].

Another important finding is that, when the pooling layer remained unaffected, linear interpolation performed considerably higher than the other downscaling strategies (ranging from 15% to 39% higher) and slightly higher than the unchanged network (ranging from 8% to 20%), with the exception of network A24 where the linear interpolation scored significantly low (33%).

Model (accuracy in original dataset)	Pooling	No Downscaling	Nearest Neighbor	Linear Interpolation	Inverse Distance Weighting
A24 (100%)	Pooling as Convolution	-	80%	20%	20%
	Pooling Unaffected	86%	80%	33%	77%
B24 (93%)	Pooling as Convolution	-	80%	15%	1%
	Pooling Unaffected	79%	80%	95%	78%
C24 (95%)	Pooling as Convolution	max pooling	max pooling	max pooling	max pooling
	Pooling Unaffected	80%	67%	90%	53%
D24 (97%)	Pooling as Convolution	-	40%	20%	20%
	Pooling Unaffected	70%	40%	78%	44%
E24 (78%)	Pooling as Convolution	max pooling	max pooling	max pooling	max pooling
	Pooling Unaffected	57%	40%	77%	38%

TABLE 1: The categorical accuracy of the downscaled CNN with the presented methods

In fact, it is the only method that may be of some use since it clearly surpasses the accuracy rate of leaving the weights unaffected in most cases.

Moreover, with the exception of network D24, the nearest neighbor method gave better results than the inverse distance weighting, although by a small margin (2% to 3% higher apart from C24 where it was 14% higher), and even in the case of D24 the difference was very small (the nearest neighbor scored 4% lower). However, in all but one cases, the CNN downscaled with the nearest neighbor interpolation had lower accuracy rate than applying no method at all (ranging from 6% to 30%) and in the exceptional case of B24 the accuracy was practically the same (79% for nearest neighbor and 80% for no downscaling).

When a CNN is trained to perform a specific task, the different components of the network architecture acquire a specific function in the process of training. This is easier to visualize in the case of image processing networks but arguably more difficult in the case of audio, as is the case here. For example, a node in a layer may be trained to detect edges (in image processing) or a specific frequency (in audio processing). Hypothetically, a robust multiscaling strategy would preserve the function of the edges across different scales. It would be difficult for a component of the network to adjust to a new function without further training.

A possible explanation for the fact that using the same network is rather successful with the new data is that the features of the signal that are detected by the different components of the network can be transferred to a signal of a different resolution. Namely, an audio recording of the same source in two different sampling rates keeps some of its spatial, temporal and structural information that can be detected by the same component (i.e. node in a layer) of the

network.

Using this line of thought, the failure of the nearest neighbor and inverse distance weighting may be interpreted as a failure to transfer these acquired functions. The nearest neighbor interpolation is the simplest interpolation method and it usually does not have the desired effect in signal processing [10]. It should not come as a surprise that it did not give useful results in this case either. When it comes to the inverse distance weighting method, as we mentioned in the relevant section, the sample points of the coarse scale array are not a subset of the sample points of the fine scale one. Moreover, the new sample points come from a weighted average formula meaning that the information contained in the new sample is not a balanced sum of the information in the original, based on a distance which is defined in a specific yet not unique way. The combination of these two factors possibly destroys the function of the CNN components when transferred to the coarse scale.

The method of linear interpolation seems to be better in preserving the functionality of the network. A possible explanation is that the reasons that make inverse distance weighting a problematic strategy are the reasons that make the linear interpolation successful. Namely, the method preserves the original sample points while interpolating a balanced average in between. The success of the unchanged architecture hints that the original sample points should be preserved instead of discarded in a successful method, which is what the linear interpolation does. This view is reinforced by the fact that the nearest neighbor interpolation gave slightly better results than inverse distance weighting since it is also a method in which the original sample points are a subset of the interpolated signal.

A note of caution is due here since the results do not appear to be universal. For example, the accuracy rates of nearest neighbor and inverse distance weighting were relatively high in the networks A24, B24 and C24 but quite low in the networks D24 and E24. After examining the architectures of the CNNs closer (see Appendix D), it can be seen that there is a difference between the two groups. Namely, network A24, B24 and C24 consist of a number of convolutional layers with pooling layers between them while the networks D24 and E24 consist of consecutive convolutional layers and one pooling layer in the end. This indicates a relation between the appropriate downscaling method and the network architecture. Furthermore, the general pattern of the success of linear interpolation is disturbed by the network A24 in which it gave the lowest accuracy rate. In fact, the accuracy rate was in the neighborhood of 20%, indicating random classification. This exception shows that the method is not universally valid and the conditions of its success can inform a new study.

In the work of Haber et al. we can see that their implementation of the AMG method gave better results in comparison to the present study. In fact, the accuracy rates of the downscaled networks were close to the accuracy rates of the networks that have been trained for the target dataset [4]. However, as mentioned above, a more detailed comparison is not possible because the prolongation and restriction operators that we used were different since we focused on the one-dimensional case.

4.2 Upscaling

As it was explained in the relevant section, the application of the AMG multiscaling methods to the average pooling layer with the purpose of upscaling returned an inconsistent linear system. Therefore, the upscaling strategies that were tried left the pooling layers unaffected. It should be noted though that the results of the previous section seem to support the idea that the pooling layers should remain unchanged across scales. The categorical accuracy over the testing set using AMG with the three interpolation methods is shown in Table 2.

Table 2 shows that the accuracy of all the upscaling methods is on the neighborhood of 20% which can be interpreted as a random prediction in the present case.

Model (accuracy in original dataset)	No Upscaling	Nearest Neighbor	Linear Interpolation	Inverse Distance Weighting
A12 (100%)	80%	20%	20%	20%
B12 (99%)	86%	20%	20%	1%
C12 (98%)	78%	19%	19%	19%
D12 (98%)	83%	20%	20%	20%
E12 (98%)	80%	20%	20%	30%

TABLE 2: The categorical accuracy of the upscaled CNN with the presented prolongation methods using AMG

A possible explanation for these results may be the lack of a unique solution to the upscaling problem. As explained before, an effective multiscaling strategy may be preserving the function of the CNN components across different scales. However, an infinite solution set would possibly also contain solutions that alter the nature of the learned parameters. As mentioned above, the least square algorithm was used to calculate a particular solution. This algorithm, although of proven general efficiency, may not be appropriate in identifying a fitting solution in this particular problem.

Another important result is that leaving the network unchanged performs with considerable success (with accuracy rates in the neighborhood of 80%). This indicates that a solution to the problem of upscaling is indeed possible, although none of the suggested methods seem to work.

In Table 3, the categorical accuracy rates of applying direct kernel prolongation are presented. It can be seen that the method of kernel dilation consistently outperforms all the other methods as well as leaving the kernels unaffected, having accuracy rates that range between 80% and 90%. The other three interpolation methods create networks with accuracy rates considerably lower in comparison to not changing the weights. In particular, leaving the weights unchanged leads to accuracy rates in the neighborhood of 80% while the three methods have results that stretch from 28% to 69%. Therefore, they do not seem to be of some use. Comparing the three interpolation methods, linear interpolation and inverse distance weighting have similar results while the nearest neighbor generally scores relatively lower. However, finding a general pattern is not as clear as in the previous cases since the differences are relatively small (around 10%). Finally, it can be seen once more that treating the average pooling layers as convolutional does not work since it practically leads to CNNs that classify randomly.

The results of applying direct kernel prolongation appear to reinforce the idea that the original CNN configuration should not be overly altered. Kernel dilation, a method which simply interpolates zeros between the original weights, seems to be a quite effective strategy in upscaling albeit the simplest one.

In the case of upscaling, Haber et al. managed to upscale the kernels of their image classification CNNs with a high accuracy rate. However, they did so by using an approach of CNNs as an optimal control problem, something that went beyond the scope of the present work.

Model (accuracy in original dataset)	Pooling	No Upscaling	Nearest Neighbor	Linear Interpolation	Inverse Distance Weighting	Kernel Dilation
A12 (100%)	Pooling as Convolution	20%	20%	20%	20%	20%
	Pooling Unaffected	80%	43%	54%	49%	90%
B12 (99%)	Pooling as Convolution	20%	20%	20%	20%	20%
	Pooling Unaffected	86%	36%	28%	28%	91%
C12 (98%)	Pooling as Convolution	max pooling	max pooling	max pooling	max pooling	max pooling
	Pooling Unaffected	78%	59%	61%	68%	84%
D12 (98%)	Pooling as Convolution	max pooling	max pooling	max pooling	max pooling	max pooling
	Pooling Unaffected	83%	40%	40%	40%	93%
E12 (98%)	Pooling as Convolution	20%	20%	20%	20%	20%
	Pooling Unaffected	80%	59%	69%	69%	80%

TABLE 3: The categorical accuracy of the upscaled CNN with the presented interpolation methods using kernel prolongation

5 Conclusion

This study set out to discover whether it is possible to use CNNs across resolution scales different from the ones for which they were trained. The focus was on one-dimensional CNNs. The approach to multiscaling was inspired by algebraic multigrid, following a matrix interpretation of applying the convolution kernels. Based on this theory, three combinations of prolongation and restriction operators were investigated: nearest neighbor, linear interpolation and inverse distance weighting. Moreover, the analysis was extended to include the non-convolutional layers of CNNs, the pooling and fully connected layers. In order to test the efficacy of the proposed methodology, a dataset was collected and used to train CNNs in audio classification across two different sampling rates.

One of the interpolation methods that were used, linear interpolation, seems to be useful in the case of downscaling. In particular, the accuracy rate of applying linear interpolation was higher than leaving the network unaltered in all but one experiment. Moreover, there were some indications regarding the correct approach in treating the pooling layers in downscaling. In particular, it seems that they should remain unchanged. In the case of upscaling, the three methods that were applied in the framework of algebraic multigrid failed to give a unique weight configuration. When one possible solution was tried, the results were not promising. However, when the kernels were directly prolonged, kernel dilation was considerably effective. It is worth noting that using the same network across scales without changing the weights was

rather successful.

6 Limitations and Future Work

It is important to state that the study had many limitations. Firstly, the experiments were conducted using a limited dataset and a small number of neural networks. In order to verify the results, more research is needed using larger datasets, different network architectures and different processing tasks (e.g. image classification, denoising, etc).

In addition, the study was limited to a small number of interpolation methods, meaning that many more prolongation and restriction strategies may be tested. It should be noted that the interpolation methods are restricted to integer scaling ratios in the case of nearest neighbor and kernel dilation and ratios that are powers of 2 in the case of linear interpolation. Inverse distance weighting can be used for non-integer ratios but it did not seem to be successful in the experiments. This means that there is a gap for multiscaling methods that could successfully multiscale CNNs without ratio limitations, something that could be further investigated.

Furthermore, in the implementation of AMG, the smoothing operation was ignored, an assumption which may need reconsidering. Especially in the case of upscaling, there may be strategies to restrict the solution sets or iterative methods that converge to an appropriate solution as suggested in [4]. Moreover, algebraic multigrid and direct kernel prolongation are not the only potential strategies that can be used in multiscaling CNNs and different approaches may render better results.

In the preceding analysis, the focus was on multiscaling the weights of the convolution kernels while the network architecture was left unchanged. An interesting idea that was discussed was the possible relationship between different network architectures and multiscaling methods, something that could inspire further study. Finally, a natural progression of this work would be to investigate strategies in which the architecture itself is altered along with the learned parameters.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[2] Claudio Ciancio, Giuseppina Ambrogio, Francesco Gagliardi, and Roberto Musmanno. Heuristic techniques to optimize neural network architecture in manufacturing applications. *Neural Computing and Applications*, 27(7):2001–2015, 2016.

[3] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pages 855–863, 2014.

[4] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales—multiscale methods for convolution neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] RD Lonsdale. An algebraic multigrid solver for the navier stokes equations on unstructured meshes. *International Journal of Numerical Methods for Heat & Fluid Flow*, 3(1):3–14, 1993.

[6] Qianshun Chang, Shuqing Ma, and Guangyao Lei. Algebraic multigrid method for queueing networks. *International journal of computer mathematics*, 70(3):539–552, 1999.

[7] Klaus Stüben. A review of algebraic multigrid. In *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359. Elsevier, 2001.

[8] Klaus Stüben. Algebraic multigrid (amg): experiences and comparisons. *Applied mathematics and computation*, 13(3-4):419–451, 1983.

[9] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[10] Mathieu Lepot, Jean-Baptiste Aubin, and François Clemens. Interpolation in time series: An introductive overview of existing methods, their performance criteria and uncertainty assessment. *Water*, 9(10):796, 2017.

[11] Stephen H Friedberg, Arnold J Insel, and Lawrence E Spence. *Algebra lineal*. Publicaciones Cultural, 1982.

[12] Steven Schlegel, Nico Korn, and Gerik Scheuermann. On the interpolation of data with normally distributed uncertainty for visualization. *IEEE transactions on visualization and computer graphics*, 18(12):2305–2314, 2012.

[13] François Chollet et al. Keras. `https://keras.io`, 2015.

[14] Smart Mobile Tools. Mp3 recorder. `https://play.google.com/store/apps/details?id=com.fragileheart.recorder&hl=en`, 2019.

[15] Divide data for optimal neural network training. `https://nl.mathworks.com/help/deeplearning/ug/divide-data-for-optimal-neural-network-training.html`. Accessed: 2019-26-06.

[16] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 18 – 24, 2015.

[17] J. Kaiser and R. Schafer. On the use of the i0-sinh window for spectrum analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):105–107, February 1980.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

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

[20] Christopher C Paige and Michael A Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.

A Strided Convolution Kernels

In this section, we shall prove that, when using the Galerkin method in algebraic multigrid to multiscale convolution kernels using the interpolations outlined in the main text, the strides of the convolutions cannot change. During the analysis, we had to solve the matrix equation $K_H = RK_hP$ in the general form. Therefore, it was important to know the structure of the matrices K_h and K_H. In particular, given a convolution kernel, we should examine whether the strides would change across scales.

When using strides, the matrix representation of the convolution operation changes. Specifically, the diagonals of the matrix are not fixed. In comparison to the matrix representation of a convolution with $stride = 1$, each row is shifted to the right by the number of strides. Namely, the matrix representation of the kernel $s = [x_1, x_2, \cdots, x_n]$ with $stride = m$ without zero padding in the borders is:

$$K_h = \begin{pmatrix} x_1 & x_2 & \cdots & x_n & 0 & \cdots & 0 & 0 \\ \underbrace{0 \cdots 0}_{\text{m}} & x_1 & x_2 & \cdots & x_n & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix}$$

The number of columns should equal the size of the input signal while the number of rows should equal the size of the output signal, which will be smaller than the input depending on the number of strides.

When two matrices P and R are the prolongation and restriction matrices explained in the main text, the product RK_hP should also have the form of the above matrix, it will have the form of a matrix representation of a convolution operation with $strides = m$. Namely, the prolongation and restriction strategies that were examined do not change the strides of the kernel. Therefore, when moving from the fine scale to the coarse scale, the coarse scale will have the same number of strides.

Conversely, when moving from the coarse scale to the fine scale where the coarse scale operation has $stride = m$, the matrix will be:

$$K_H = \begin{pmatrix} y_1 & y_2 & \cdots & y_n & 0 & \cdots & 0 & 0 \\ \underbrace{0 \cdots 0}_{\text{m}} & y_1 & y_2 & \cdots & y_n & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix}$$

If the fine scale operation has $strides = k \neq m$, then the product RK_hP will have the form:

$$RK_hP = \begin{pmatrix} w_1 & w_2 & \cdots & w_n & 0 & \cdots & 0 & 0 \\ \underbrace{0 \cdots 0}_{\text{k}} & w_1 & w_2 & \cdots & w_n & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix}$$

When equating the two matrices, we have two cases:
Case 1: $k > m$

Then, we will have $y_j = w_j, 0 < j < m$ from the first row but $y_j = 0, 0 < j \leq k - m$ and $y_j = w_{j+m-k}, k - m < j \leq m$ from the second row. Therefore, we have $y_j = w_j = 0, 0 < j \leq k - m$ and $y_j = w_i, k - m < j \leq m, 0 < i \leq k - m$. Therefore, the only solution is for the zero matrices.

Case 2: $k < m$

The same result follows by a similar reasoning.

B Matrix Representations of the Prolongation Operators

B.1 Nearest Neighbor

Let $F : \mathbb{R}^n \longrightarrow \mathbb{R}^{Nn}$ be the nearest neighbor transformation. The transformation matrix of the operation is the prolongation matrix P. The shape of the matrix is $Nn \times n$ and the columns are the vectors $a_i = F(e_i), 1 \leq i \leq n$ where e_i the normal basis of $\mathbb{R}^n$ [11].

$$
\begin{aligned}
a_1 &= F(\big[1,0,\cdots,0\big]^T) = \big[1,\cdots,1,0,\cdots 0\big]^T \\
a_2 &= F(\big[0,1,0,\cdots,0\big]^T) = \big[0,0,1,\cdots 1,0\cdots,0\big]^T \\
&\vdots \\
a_n &= F(\big[0,\cdots,0,1\big]^T) = \big[0,0,\cdots,0,\underbrace{1,\cdots 1}_{\text{N}}\big]^T
\end{aligned}
$$

If we create a matrix with rows the vectors a_i, we can see that it will be in the echelon form and prove that $a_1,\ldots,a_n$ are linearly independent from the relevant theory of linear algebra [11].

B.2 Linear Interpolation

Let now $F : \mathbb{R}^n \longrightarrow \mathbb{R}^{2n}$ be the linear interpolation transformation. We need to find the transformation (prolongation) matrix P. The shape of the matrix will be $2n \times n$ and the columns are the vectors $a_i = F(e_i), 1 \leq i \leq n$ where e_i the normal basis of $\mathbb{R}^n$ [11].

$$
\begin{aligned}
a_1 &= F(\big[1,0,\cdots,0\big]^T) = \big[1,\frac{1}{2},0,\cdots 0\big]^T \\
a_2 &= F(\big[0,1,0,\cdots,0\big]^T) = \big[0,\frac{1}{2},1,\frac{1}{2},0,\cdots,0\big]^T \\
&\vdots \\
a_n &= F(\big[0,\cdots,0,1\big]^T) = \big[0,\cdots,0,\frac{1}{2},1,\frac{1}{2}\big]^T
\end{aligned}
$$

If we create a matrix with rows the vectors a_i, we can see that it will be in the echelon form and that $a_1,\ldots,a_n$ are linearly independent [11].

B.3 Inverse Distance Weighting

Let now $F : \mathbb{R}^n \longrightarrow \mathbb{R}^{2n}$ be the inverse distance weighting transformation applied with scaling ratio 2. We need to find the transformation (prolongation) matrix P. The shape of the matrix will be $2n \times n$ and the columns are the vectors $a_i = F(e_i), 1 \leq i \leq n$ where e_i the normal basis of $\mathbb{R}^n$ [11].

$$
\begin{aligned}
a_1 &= F(\big[1,0,\cdots,0\big]^T) = [\frac{5}{6},\frac{3}{4},\frac{1}{4},0,\cdots 0]^T \\
a_2 &= F(\big[0,1,0,\cdots,0\big]^T) = [\frac{1}{6},\frac{1}{4},\frac{3}{4},\frac{3}{4},\frac{1}{4},0,\cdots,0]^T \\
a_3 &= F(\big[0,0,1,\cdots,0\big]^T) = \big[0,0,0,\frac{1}{4},\frac{3}{4},\frac{3}{4},\frac{1}{4},0,\cdots,0\big]^T \\
&\vdots \\
a_{n-2} &= F(\big[0,\cdots,0,1,0,0\big]^T) = \big[0,\cdots,0,\frac{1}{4},\frac{3}{4},\frac{3}{4},\frac{1}{4},0,0,0\big]^T \\
a_{n-1} &= F(\big[0,\cdots,0,1,0\big]^T) = \big[0,\cdots,0,\frac{1}{4},\frac{3}{4},\frac{3}{4},\frac{1}{4},\frac{1}{6}\big]^T \\
a_n &= F(\big[0,\cdots,0,1\big]^T) = \big[0,\cdots,0,\frac{1}{4},\frac{3}{4},\frac{5}{6}\big]^T
\end{aligned}
$$

By creating a matrix with a_i as rows and bringing it to the echelon form using elementary row operations, we can see that $a_1,\ldots,a_n$ are linearly independent [11].

C 12 kHz CNNs

The architectures of the CNNs for the 12 kHz dataset are shown in Figures 5, 6, 7, 8 and 9. The input size of the networks was $input\_size = duration \cdot sampling\_rate = 4 \cdot 12000 = 48000$. The optimal batch size was 3 and training lasted netween two and three epochs, afterwards it started overfitting possibly because of the small dataset. The dropout rate that was used was 0.5.

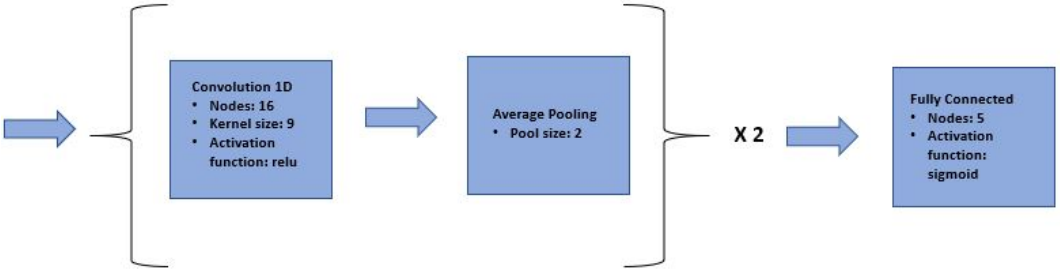

FIGURE 5: Model A12 for the 12kHz dataset. The categorical accuracy over the testing set is 100%. The training was performed with batch size was 1 and it lasted for two epochs, before overfitting. The dropout rate that was used was 0.5.

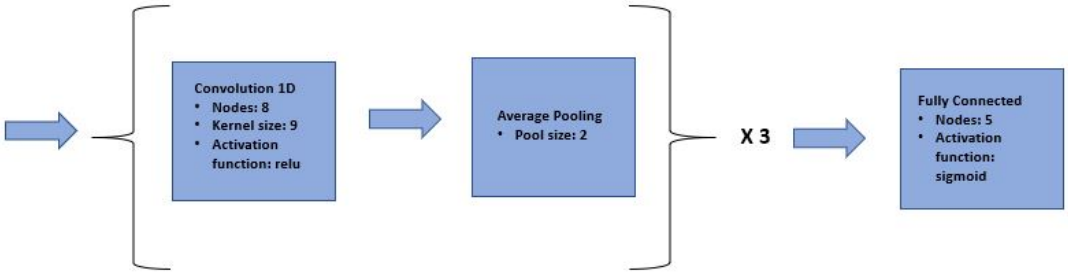

FIGURE 6: Model B12 for the 12kHz dataset. The categorical accuracy over the testing set is 99%, The training was performed with batch size was 1 and it lasted for two epochs, before overfitting. The dropout rate that was used was 0.5.

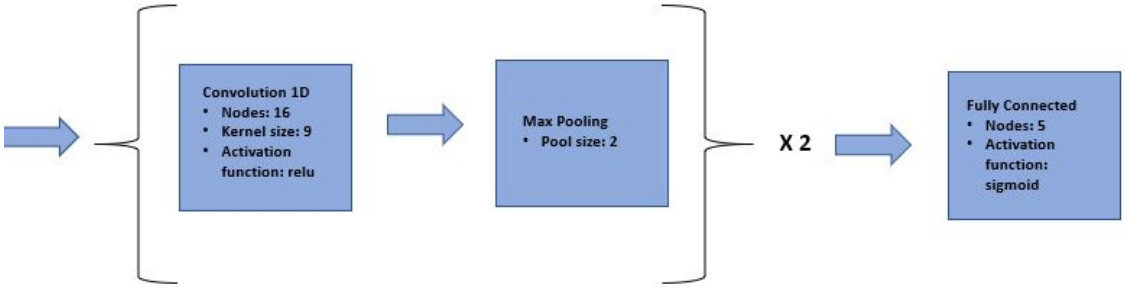

FIGURE 7: Model C12 for the 12kHz dataset. The categorical accuracy over the testing set is 98%. The training was performed with batch size was 1 and it lasted for two epochs, before overfitting. The dropout rate that was used was 0.5.

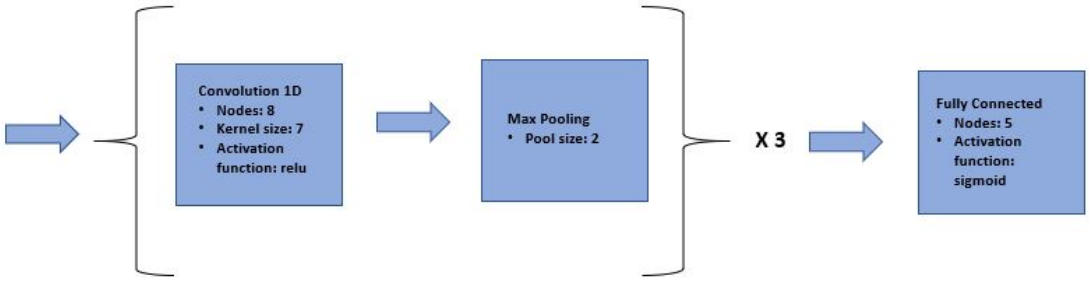

FIGURE 8: Model D12 for the 12kHz dataset. The categorical accuracy over the testing set is 98%. The training was performed with batch size was 3 and it lasted for three epochs, before overfitting. The dropout rate that was used was 0.5.

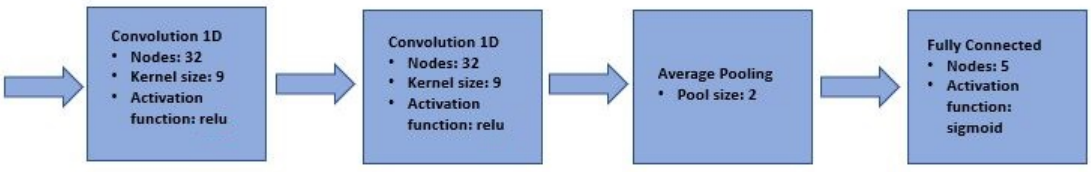

FIGURE 9: Model E12 for the 12kHz dataset. The categorical accuracy over the testing set is 98%. The training was performed with batch size was 3 and it lasted for three epochs, before overfitting. The dropout rate that was used was 0.5.

D 24 kHz CNNs

The CNNs for the 24 kHz dataset is shown in Figures 10, 11, 12, 13 and 14. The input size of the networks was $input\_size = duration \cdot sampling\_rate = 4 \cdot 24000 = 96000$. The batch size was between 1 and 3 and training lasted between two and three epochs, before starting overfitting. The optimal dropout rate was 0.5. The accuracy over the training set was 97%.

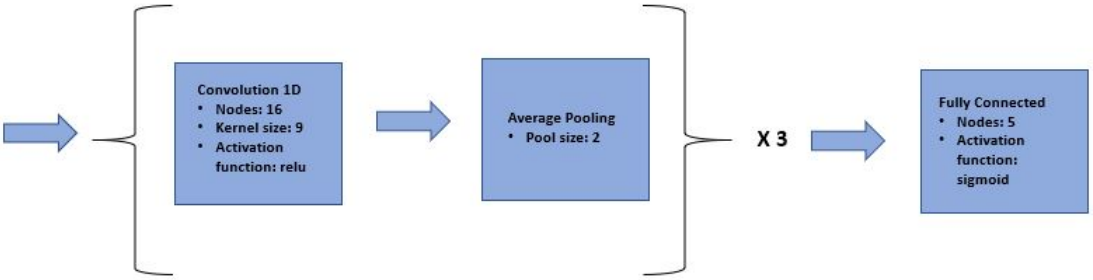

FIGURE 10: Model A24 for the 24kHz dataset. The categorical accuracy over the testing set is 100%. The training was performed with batch size was 1 and it lasted for tweo epochs, before overfitting. The dropout rate that was used was 0.5.

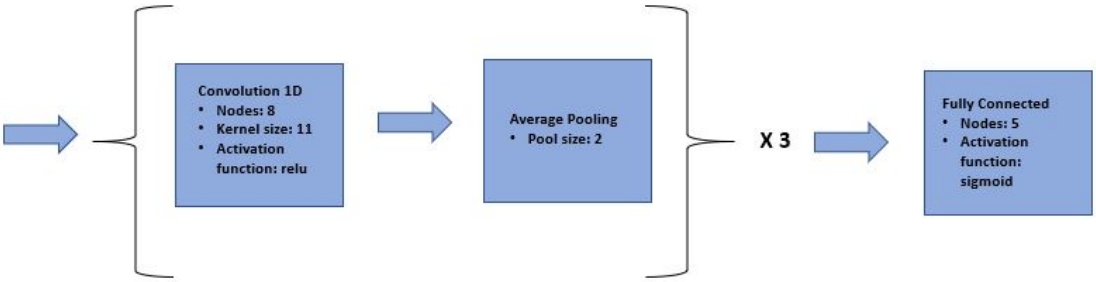

FIGURE 11: Model B24 for the 24kHz dataset. The categorical accuracy over the testing set is 93%. The training was performed with batch size was 1 and it lasted for two epochs, before overfitting. The dropout rate that was used was 0.3.

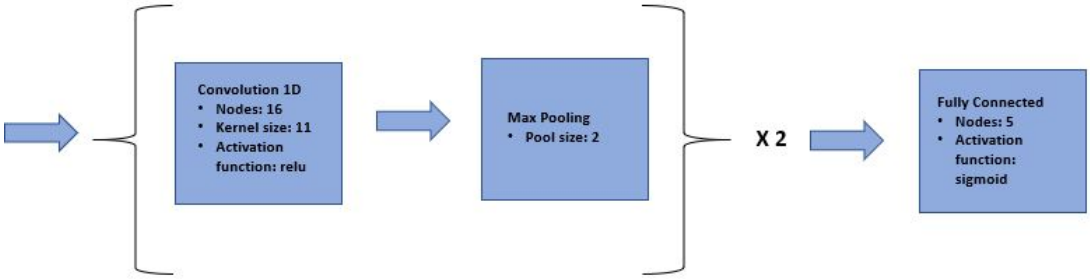

FIGURE 12: Model C24 for the 24kHz dataset. The categorical accuracy over the testing set is 95%. The training was performed with batch size was 1 and it lasted for two epochs, before overfitting. The dropout rate that was used was 0.4.

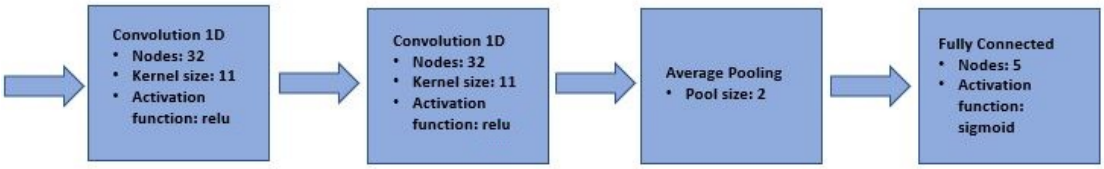

FIGURE 13: Model D24 for the 24kHz dataset. The categorical accuracy over the testing set is 97%. The training was performed with batch size was 3 and it lasted for three epochs, before overfitting. The dropout rate that was used was 0.5.

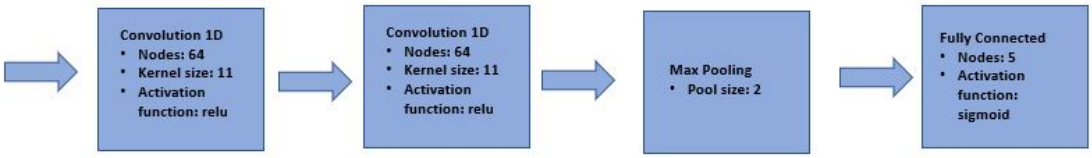

FIGURE 14: Model E24 for the 24kHz dataset. The categorical accuracy over the testing set is 78%. The training was performed with batch size was 3 and it lasted for three epochs, before overfitting. The dropout rate that was used was 0.5.