T.R.
SAKARYA UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

A NEW VOLTERRA NEURAL NETWORK LAYER LIBRARY USING TENSORFLOW

MSc THESIS

Zakaria ALYAFAWI

Computer and Information Engineering Department

Computer Engineering Program

JANUARY 2023

T.R.
SAKARYA UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

A NEW VOLTERRA NEURAL NETWORK LAYER LIBRARY USING TENSORFLOW

MSc THESIS

Zakaria ALYAFAWI

Computer and Information Engineering Department

Computer Engineering Program

Thesis Advisor: Doç. Dr. Devrim AKGÜN

JANUARY 2023

The thesis work titled “A New Volterra Neural Network Layer Library using TensorFlow” prepared by Zakaria Alyafawi was accepted by the following jury on 01/20 /2023 by unanimously/majority of votes as a MSc THESIS in Sakarya University Graduate School of Natural and Applied Sciences, Computer and Informatıon Engıneerıng department, Computer and Informatıon Engıneerıng program.

Thesis Jury

Head of Jury : **Prof. Dr. Ahmet ZENGIN**
Sakarya University

Jury Member : **Prof. Dr Devrim AKGÜN**
Sakarya University

Jury Member : **Dr. Öğr Üyesi Selman HIZAL**
Sakarya University of Applied Sciences

STATEMENT OF COMPLIANCE WITH THE ETHICAL PRINCIPLES AND RULES

I declare that the thesis work titled "A New Volterra Neural Network Layer Library using Tensorflow", which I have prepared in accordance with Sakarya University Graduate School of Natural and Applied Sciences regulations and Higher Education Institutions Scientific Research and Publication Ethics Directive, belongs to me, is an original work, I have acted in accordance with the regulations and directives mentioned above at all stages of my study, I did not get the innovations and results contained in the thesis from anywhere else, I duly cited the references for the works I used in my thesis, I did not submit this thesis to another scientific committee for academic purposes and to obtain a title, in accordance with the articles 9/2 and 22/2 of the Sakarya University Graduate Education and Training Regulation published in the Official Gazette dated 20.04.2016, a report was received in accordance with the criteria determined by the graduate school using the plagiarism software program to which Sakarya University is a subscriber, I have received an ethics committee approval document I accept all kinds of legal responsibility that may arise in case of a situation contrary to this statement.

(10/01/2023)

Zakaria Alyafawi

To my parent's brothers, and sisters I am extremely grateful each one of you, for their unwavering support and encouragement throughout my academic journey.

ACKNOWLEDGMENTS

I am extremely grateful to my supervisor, Dr. Devrim, for his valuable guidance, insights, and encouragement throughout this project.

I am deeply indebted to my mentors and professors at the university who have imparted their knowledge and expertise to me and have played a crucial role in shaping my academic and professional career. I am also grateful to my colleagues and friends for their support and camaraderie during this project. Finally, I would like to express my sincere appreciation to the university, the college, and all the staff members for providing me with the necessary resources and facilities to complete this project successfully.

Zakaria Alyafawi

TABLE OF CONTENTS

ABBREVIATIONS

ANN	: Artificial Neural Networks
CE	: Cross-entropy error function
CNN	: Convolutional neural network
CPU	: Central processing unit
GPU	: Graphics processing unit
ReLU	: Rectified linear unit
Conv1D	: One Diminution Convolution Layer
DL	: Deep Learning
ML	: Machine Learning
ConvNet	: Convolutional Neural Network
MSE	: Mean Squared Error

LIST OF TABLES

Page

LIST OF FIGURES

Page

A NEW CUSTOM TENSORFLOW LAYER BASED ON SECOND-ORDER ONE-DIMENSION VOLTERRA CONVOLUTION

SUMMARY

Deep learning algorithms have garnered much attention recently, their success in enhancing the accuracy of automatic speech recognition systems has caused an increase in their usage. These models have the ability to identify patterns within input data and generate predictions based on these patterns. However, a limitation of these models is their inability to capture nonlinear relationships within input data.

This study aimed to enhance the performance of automatic speech recognition through the incorporation of a second-order, one-dimensional Volterra Convolution (Volt1D) layer into deep learning models. The Volt1D layer is a custom TensorFlow layer that is founded on the Volterra series convolution, a mathematical tool capable of representing a wide array of nonlinear functions.

To evaluate the efficacy of the Volt1D layer, we compared its performance to that of the standard Conv1D layer using the speech commands dataset v0.01, which consists of 20 classes of spoken words. Our results indicated that the Volt1D layer achieved an accuracy of 64.91% over 10 epochs, a significant improvement over the baseline accuracy of 60.02% using the Conv1D layer over 10 epochs. This demonstrates the Volt1D layer's effectiveness in extracting nonlinear relationships within speech data. We talked about the advantages and disadvantages of these approaches. with regard to the main objective of the study, which was to capture nonlinear relationships in speech data through the use of the Volt1D layer. We found that the Volt1D layer is a promising approach for speech recognition due to its ability to effectively capture nonlinear relationships and enhance the effectiveness of deep learning models.

One of the primary strengths of the Volt1D layer is its capability to represent a wide range of nonlinear functions, making it suitable for capturing complex relationships within input data, a crucial factor in accurately transcribing spoken words into written text. In addition, the Volt1D layer is computationally efficient, allowing for its utilization in real-time speech recognition applications without incurring excessive computational overhead. However, there are also some limitations to the Volt1D layer. One limitation is that these models require a vast amount of training data to effectively learn the intricate nonlinear relationships within input data, which can be problematic for smaller datasets or applications with limited access to ample training data. Additionally, the Volt1D layer may not be as effective at capturing long-term dependencies within input data as other methods like RNN or LSTM models

In conclusion, the Volt1D layer represents a promising approach for improving the performance of automatic speech recognition through deep learning models. It is able to effectively capture nonlinear relationships within input data, exhibiting superior accuracy and computational efficiency compared to other methods. However, the model may be constrained by the need for a substantial amount of training data and its capacity to identify long-term dependencies within input data.

A NEW CUSTOM TENSORFLOW LAYER BASED ON SECOND-ORDER ONE-DIMENSION VOLTERRA CONVOLUTION

ÖZET

Son yıllarda, konuşma tanıma sistemlerinin performansını iyileştirmek için derin öğrenme modelleri yaygın olarak kullanılmaktadır. Bu modeller, girdi verilerindeki kalıpları öğrenme ve tanıma ve bu kalıplara dayalı tahminler yapma yeteneğine sahiptir. Bununla birlikte, bu modellerin bir sınırlaması, girdi verilerinde doğrusal olmayan ilişkileri yakalayamamalarıdır.

Bu çalışmada, yeni bir katman ikinci dereceden 1D Volterra Convolution (Volt1D) katmanı tanıtarak derin öğrenme modelleri kullanarak konuşma tanıma performansını iyileştirmeyi amaçladık. Volt1D katmanı, çok çeşitli doğrusal olmayan işlevleri temsil edebilen matematiksel bir araç olan Volterra serisi konvolüsyonu temel alan özel bir TensorFlow katmanıdır.

Volt1D katmanının performansını, 20 sözlü sözcük sınıfından oluşan konuşma komutları veri kümesi v0.01'de standart Conv1D katmanının performansıyla karşılaştırdık. Sonuçlarımız, Volt1D katmanının 10 dönemle %64,91'lik bir doğruluğa ulaştığını gösterdi; bu, 10 dönemle Conv1D katmanı kullanılarak elde edilen %60,02'lik temel doğruluktan önemli ölçüde daha yüksek. Bu, Volt1D katmanının konuşma verilerindeki doğrusal olmayan ilişkileri çıkarmadaki etkinliğini gösterir.

Volt1D katmanını kullanarak konuşma verilerindeki doğrusal olmayan ilişkileri yakalamak olan çalışmamızın ana amacı ile ilgili olarak bu yaklaşımların güçlü yanlarını ve sınırlamalarını da tartıştık. Doğrusal olmayan ilişkileri etkili bir şekilde yakalayabildiği ve derin öğrenme modellerinin performansını iyileştirebildiği için Volt1D katmanının konuşma tanıma için umut verici bir yaklaşım olduğunu bulduk.

Volt1D katmanının ana güçlü yönlerinden biri, çok çeşitli doğrusal olmayan fonksiyonları temsil etme yeteneğidir. Bu, onu, konuşulan sözcükleri doğru bir şekilde metne dönüştürmek için önemli olan girdi verilerindeki karmaşık ilişkileri yakalamak için çok uygun hale getirir. Ek olarak, Volt1D katmanı hesaplama açısından verimlidir, bu da gerçek zamanlı konuşma tanıma uygulamalarında önemli bir hesaplama yüküne maruz kalmadan kullanılabileceği anlamına gelir.

Bununla birlikte, Volt1D katmanının bazı sınırlamaları da vardır. Bir sınırlama, girdi verilerindeki karmaşık doğrusal olmayan ilişkileri öğrenmek için büyük miktarda eğitim verisi gerektirmesidir. Bu, daha küçük veri kümeleri veya büyük miktarda eğitim verisine sınırlı erişimin olduğu uygulamalar için zor olabilir. Ek olarak, Volt1D katmanı, RNN'ler veya LSTM'ler gibi diğer yaklaşımlarla karşılaştırıldığında girdi verilerindeki uzun vadeli bağımlılıkları yakalamada o kadar etkili olmayabilir.

Sonuç olarak, Volt1D katmanı, derin öğrenme modellerini kullanarak konuşma tanıma performansını iyileştirmek için umut verici bir yaklaşımdır. Girdi verilerindeki

doğrusal olmayan ilişkileri etkili bir şekilde yakalayabilir ve doğruluk ve hesaplama verimliliği açısından diğer yaklaşımlardan daha iyi performans gösterebilir. Ancak, büyük miktarda eğitim verisine duyulan ihtiyaç ve girdi verilerindeki uzun vadeli bağımlılıkları yakalama yeteneği ile sınırlı olabilir.

1. INTRODUCTION

Convolutional neural networks (CNNs) have gained widespread popularity in recent decades as a tool for a variety of computer vision and machine learning tasks [1, 2]. These feed-forward ANNs use alternating layers of convolution and subsampling to extract features from input data [3].

Deep 1D CNNs, which consist of multiple hidden layers and a large number of parameters, are particularly effective at learning complex patterns and objects when trained on large visual databases with ground truth labels [4, 5]. As a result, CNNs have become the go-to method for a variety of technical applications involving 2D input such as images and videos [6, 7].

However, the use of CNNs may not always be feasible for 1D signal applications, particularly when the training data is of low quality or tailored to a specific application [8]. To resolve this problem, 1D CNNs were developed and have achieved exceptional performance in a range of applications, including biomedical data classification for personalized early diagnosis [9], monitoring the structural integrity [10], identifying abnormalities [11], and the detection of power electronics and electrical engine failures [12].

One major benefit of 1D CNNs is the ability to be implemented in real-time and at low cost because of their straightforward and concise design, which only performs 1D convolutions [13].

In this study, we aim to implement second-order Volterra Series Convolution as a custom TensorFlow layer to be used to train a speech recognition model with the speech commands V0.01 dataset [14].

The Volterra series is a representation of nonlinear behavior model that has the ability to capture "memory" effects [15], which may be beneficial to enhance the effectiveness of deep learning models for speech recognition tasks [16]. By implementing this custom layer, we hope to increase the accuracy of the trained model and reduce training time [17]. This study also aims to provide a all-encompassing overview of the overall design and principles of 1D CNN, focusing on recent advancements and their current

highest levels of success in various technical applications [18]. The voice recognition data and core 1D CNN used in these applications will also be made publicly available [19]. Despite the lack of literature on 1D CNNs and their applications, this study aims to fill this gap by presenting a comprehensive overview of the current state of the field [20].

1.1. Overview

Accurate speech recognition is of great importance in a variety of applications, ranging from personal assistants and voice-controlled devices [21] to automatic translation and transcription services [22].

Recently, deep learning approaches have become the dominant method for speech recognition tasks [23], because of their capability to learn intricate patterns and features from significant amounts of signals [24]. However, there are still many challenges to achieving high levels of accuracy, especially when dealing with real-world speech data that may be noisy, varied, or spoken in different languages and accents [25]. One major challenge is the nonlinear nature of speech signals, which can be difficult to model using traditional linear techniques [26].

The Volterra series as nonlinear model behaviour with the ability to capture "memory" effects [27], may be a useful tool for addressing this challenge. By implementing a custom layer based on the Volterra series in our TensorFlow model, we hope to enhance the precision of speech recognition model [28]. Accurate speech recognition is also important for improving the usability and functionality of voice-controlled devices and personal assistants [29], as well as for increasing the efficiency and accuracy of transcription and translation services [30]. These applications have the potential to revolutionize how we interact with technology and communicate with one another [31], but they require highly accurate speech recognition algorithms to be truly effective. By developing a custom layer based on the Volterra series, we aim to contribute to the ongoing efforts to enhance the accuracy and performance of the speech recognition model [32].

1.2. Problem Statement

The issue being considered in this study is the implementation of second-order Volterra series convolution as a custom TensorFlow layer to train a speech recognition model. The use of the Volterra series, with the ability to capture "memory" effects [33], will enhance the capability of deep learning models for speech commands recognition tasks by capturing both linear and nonlinear relations in the data [34].

To address this problem, we will implement a second order Volterra series convolution in Python as a TensorFlow layer [35]. This custom layer will be used in place of traditional convolutional layers in an artificial intelligence model for speech recognition and will be trained using the speech commands V0.01 dataset [36]. The accurecy of the trained AI-model with the custom Volterra series will be compared to the performance of a model using traditional convolutional layers, in order to evaluate the performance of the Volterra series in improving the performance of the trained model. Overall, the purpose of this study is to examine the potential benefits of using the Volterra series for speech recognition tasks, and to provide a detailed implementation and evaluation of a custom TensorFlow layer based on the second-order Volterra series. By addressing this problem, we hope to contribute to the ongoing efforts to enhance the effectiveness of models for speech recognition tasks, and to provide a useful tool for researchers and practitioners working in this field.

1.3. Objective

The goal of this master's thesis is to investigate and address the issue of speech recognition using CNNs for capture nonlinear behaviour. The goal of this research is to present a new method for improving the accuracy and efficiency of speech recognition models by implementing a custom CNN layer based on the Volterra series.

Speech recognition is a critical field that has significant practical and technological importance, but traditional approaches to training deep learning models for speech recognition tasks often suffer from poor accuracy with nonlinear relations between the training data, particularly when dealing with large datasets. Our new custom layer, Volt1d, aims to address these issues by replicating the behavior of the TensorFlow Conv1D layer by replacing the linear mathematical equation with a non-linear

equation to enhance the accuracy of the trained model by capture linear and nonlinear relation in the input training dataset. By exploring and addressing these issues, our research aims to contribute to the ongoing efforts to enhance the effectiveness of speech recognition model. This work has the potential to have significant practical impacts, as it could enable the improvement of speech recognition systems in terms of accuracy and efficiency that are capable of handling large and diverse datasets in real-world applications.

1.4. Research Questions

This study aims to answer the following research question: What is the impact of using a second-order Volterra series convolution layer on the precision and time required for training a speech recognition model?

To answer this research question, the following objectives have been set:

- Implement the second-order Volterra series convolution as a custom TensorFlow layer.
- Train a speech recognition model using the speech commands V0.01 dataset with the custom Volterra series convolution layer.
- Compare the accuracy of the model with the custom Volterra series convolution layer to the performance of a model with traditional convolutional layers.
- Analyse the results and determine the repercussion of the Volterra series convolution on the accuracy and training time of the model.
- Conduct a thorough implementation and assessment of the custom Volterra series convolution layer for speech recognition tasks in TensorFlow.

1.5. Organization

The present master's thesis is organized into six sections, with the first and current section serving as an introduction. In addition to providing an overview of the research, this section also includes a description of the problem being addressed, the objectives and research questions of the study, and the organization of the remaining section. The second section provides an overview of previous research on speech recognition using

a description of deep learning, including the various approaches and their strengths and limitations. This section also introduces the concept of the Volterra series and its role in enhancing the effectiveness of speech recognition model. The historical evolution of neural networks is also discussed in this section. Section 3 presents the methodology of the study, including the implementation and evaluation of the custom Volterra series convolution layer for speech recognition in TensorFlow. The datasets and experimental setup used for training and evaluating the models are also described in this section. Section 4 presents the results and analysis of the study, including the comparison of the performance of the model with the custom Volterra series convolution layer to the performance of a model with traditional convolutional layers. The impact of the Volterra series and its impact on accuracy of the model is also discussed in this section. The fifth section concludes the study by summarizing the main findings and discussing the implications of the results. This section also explores potential avenues for future research. The final section presents the references used in the study

2. LITERATURE REVIEW

Speech recognition is a rapidly developing technology that has the potential to revolutionize the way we interact with machines. The Speech Commands V0.01 dataset [37] is a publicly available dataset that contains thousands of spoken words and phrases, making it a valuable resource for training and implementing speech recognition systems. In this literature review, we will explore the current state of the art in speech recognition, focusing on the use of the Speech Commands V0.01 dataset. We will discuss the challenges of implementing and training speech recognition systems, as well as the various approaches that have been proposed to overcome these challenges. Speech recognition technology has been a topic of research for decades, with early systems dating back to the 1950s. However, it wasn't until the late 1990s and early 2000s that speech recognition systems achieved high accuracy and usability [38]. Today, speech recognition is used in a wide range of applications, including voice-controlled assistants, voice-enabled search, and accessibility tools for people with disabilities. One of the key challenges in speech recognition is the large amount of data that is required to train a system. This data must be representative of the population of speakers and the range of environments in which the system will be used. The Speech Commands V0.01 dataset [37] is a publicly available dataset that contains 65,000 spoken words and phrases, making it a valuable resource for training speech recognition systems. The dataset was created by Google and contains a wide range of spoken words and phrases, including numbers, common words, and commands. There are several challenges that must be overcome when implementing and training speech recognition systems. One of the main challenges is the variability of human speech. People speak at different speeds, use different accents, and have different speaking styles [39]. This variability makes it difficult to create a system that can accurately recognize speech in a wide range of environments.

Another challenge is the presence of noise in the environment. Background noise can make it difficult for a system to accurately recognize speech, especially in noisy

environments such as public places or crowded streets [40]. Additionally, the presence of multiple speakers can make it difficult for a system to identify the correct speaker.

A third challenge is a need for a large amount of data to train a system. As mentioned earlier, speech recognition systems require a large amount of data to achieve high accuracy levels [41]. This data must be representative of the population of speakers and the range of environments in which the system will be used.

There are several approaches that have been proposed to overcome the challenges of implementing and training speech recognition systems. One approach is to use deep learning techniques to train a system [42]. Deep learning algorithms, such as neural networks, have been shown to be effective in handling the variability of human speech. Additionally, deep learning algorithms can be trained on large amounts of data, making them well-suited for speech recognition.

Another approach is to use data augmentation techniques to increase the amount of data available for training [43]. Data augmentation techniques can be used to artificially increase the amount of data by applying various transformations to the existing data. This can include adding noise, changing the speed of speech, or applying different accents.

A third approach is to use transfer learning techniques to adapt a pre-trained model to a new task [42]. Transfer learning allows a pre-trained model to be fine-tuned on a new task, reducing the amount of data required to train the system. This can be especially useful when the data available for training is limited.

Speech recognition is a rapidly developing technology that has the potential to revolutionize the way we interact with machines. The Speech Commands V0.01 dataset [37] is a valuable resource for training and implementing speech recognition systems, as it contains a large amount of diverse spoken words and phrases. However, implementing and training speech recognition systems can be challenging due to the variability of human speech, the presence of noise in the environment, and the need for a large amount of data. To overcome these challenges, researchers have proposed various approaches such as using deep learning techniques, data augmentation and transfer learning. Further research in speech recognition will continue to improve the accuracy and usability of these systems for a wide range of applications.

2.1. Background

Despite the promising results achieved by deep learning models for speech recognition, there are still several challenges that need to be addressed. One of the main challenges is the need for significant quantities of labeled training data, which can be difficult and time-consuming to obtain [43]. Another challenge is the sensitivity of deep learning models to noise and other distortions in the data, which can reduce their accuracy [44]. To address these challenges, research is ongoing to develop new methods and techniques to enhance the accuracy of deep learning models for speech recognition tasks [45]. The description of various approaches used in previous studies as mentioned below:

2.1.1. Artificial neural networks (ANN)

Artificial neural networks (ANNs) are a type of model that uses calculations and mathematics to replicate the functioning of the human brain. Many recent developments in the realm of AI, such as visual and auditory identification and automatons, have been made possible through the use of ANNs.

These models have a specific architecture, which is influenced by the design of the natural nervous system and consists of neurons that are connected by weighted links. The neurons in ANNs are arranged in a complex and nonlinear way, similar to the anatomy of the human brain. ANNs can be trained through various methods, including information gathering and evaluation, network layout planning, determination of the quantity of hidden layers, network simulation, and optimization of weights and biases. ANNs can be used to solve problems in a variety of different scientific disciplines, and can be utilized for categorizing patterns, forecasting, and managing and optimizing.

ANNs can be classified into three types:

1. Static
2. Dynamic
3. Statistical

Each of which has its own unique structure and characteristics [46]. It is also possible to combine ANNs in combination with other optimization strategies, to improve prediction capabilities.

ANNs have significant applications in the field of Speech Recognition system. We can find in Figure 2.1 basic ANN design.

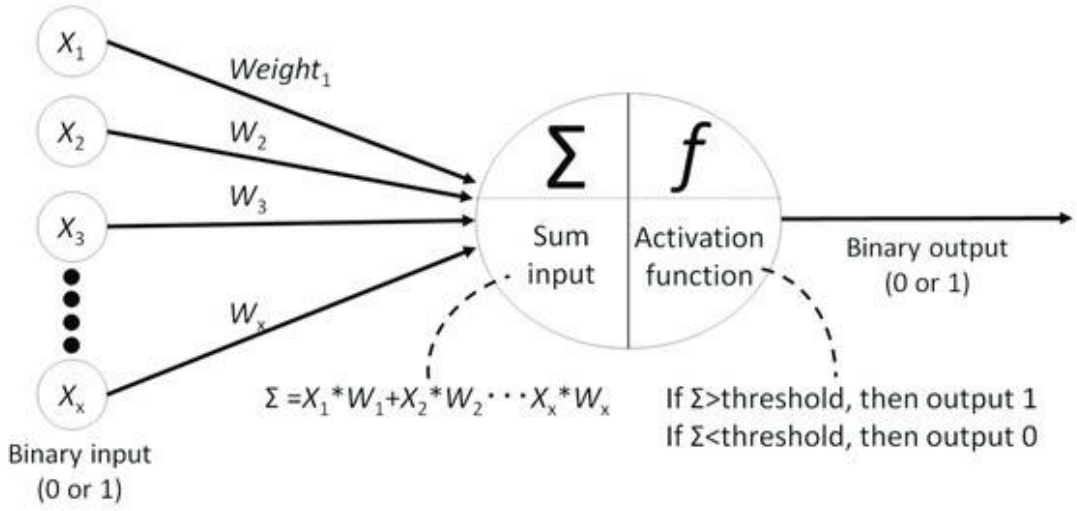

Figure 2.1. Artificial Neural Networks Architecture

As we can see in Figure 2.1 the neural network architecture, the simple neural networks consist of multiple neuron binary inputs between zero and one and we have W which is the weights, the weights will be generated randomly based on the TensorFlow algorithm, then the next layer or the hidden layers will use all the inputs from the previous layer based on the following formula.

$$z = \sum_{i=0}^{\infty} w_i x_i + b \tag{2.1}$$

Let's describe our neural network, we have list of inputs will draw it's as a single node and each node will connect to all other nodes in the next hidden layer or to the output directly based on the architecture, and each node in the next hidden layer will connect to all other nodes in the second hidden layer or the output and repeat the same protocol for other concealed layers and apply the arithmetic operation as it in (2.1) for example in Figure 2.2 we have just three inputs into our neural network we will represent each

input as single node, so in our example we have three nodes as inputs, each input of our inputs will connect to all the other nodes in the initial hidden layer, in this step, the neural network will generate the weights for each input and apply the arithmetic operation by multiplying the input with its weights and add the bias to the operation, the neural network will do the same thing for all the inputs each input with its weight, so after applying all this operations will set the outputs as a node in the initial concealed layer, and the same is true for the subsequent concealed layer and at the end will send the final results to the output layer.

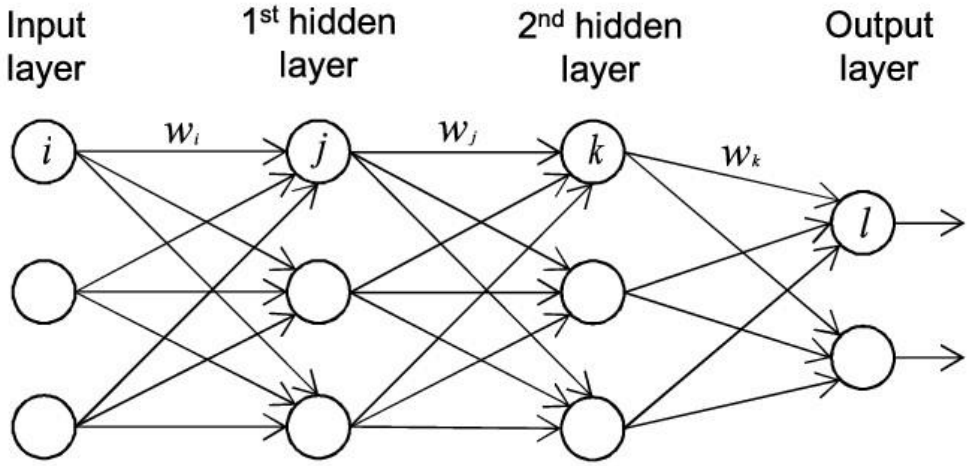

Figure 2.2. Simple neural network architecture

2.1.2. Activation function

The activation function plays a significant role in influencing the output of a neural network. It is a mathematical function that decides whether or not a neuron's input should be considered important for the prediction process. Activation functions allow for non-linearity in neural networks, allowing them to make complex decisions based on input data. They also map the output values of a node to a specific range, such as between 0 and 1 or -1 and 1. Common types of activation functions include sigmoid, tanh, and ReLU. The activation function is commonly known as the transfer function in artificial neural networks [47]. It is essential to choose the appropriate activation function for a given task, as it can significantly impact the functionality of the artificial neural network.

2.1.2.1. Linear activation function

The straight-line activation function, also referred to as the "no activation" or "identity function," simply returns the input that it receives, multiplied by a factor of 1.0. This function does not alter the weighted sum of the input in any way and produces an output that is proportional to the input. As shown in, Figure 2.3, the function is linear, meaning that the output will not be limited to any specific range. This function is also known as the "no activation" function, as it does not perform any activation on the input.

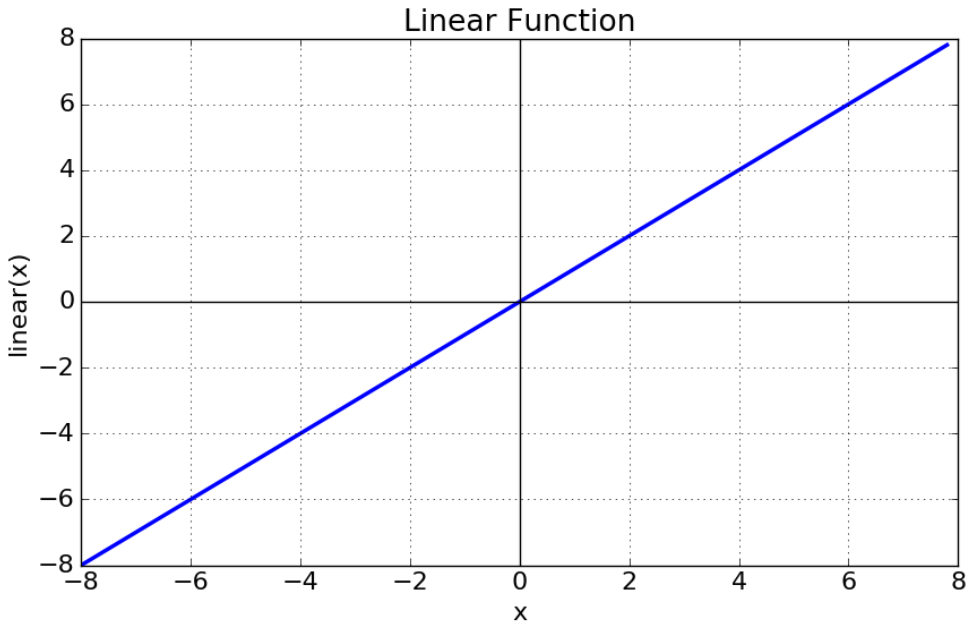

Figure 2.3. Linear Activation Function

Where the Equation equals $f(x) = x$ and the range between – infinity to infinity.

2.1.2.2. Non-linear activation function

Non-linear activation-functions are the most commonly used types of activation-functions. These functions introduce nonlinearity into the system allows the graph to take on a shape similar to Figure 2.4, which facilitates the model's ability to adapt to a variety of data and to distinguish between different outputs. This is because nonlinearity enables the model to generalize effectively, allowing it to perform well on a variety of data. Nonlinear activation functions are therefore essential in helping the model to accurately analyse and interpret the input data.

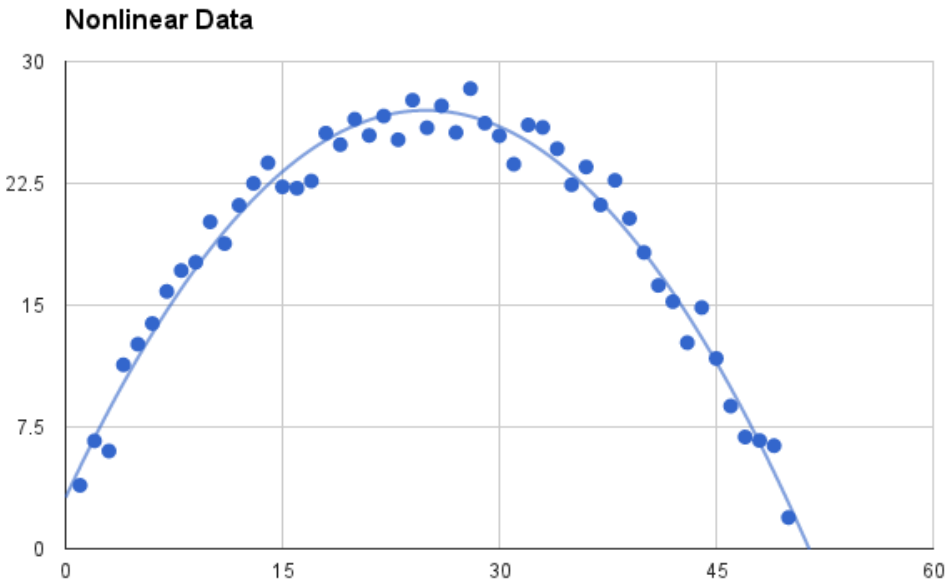

Figure 2.4. Nonlinear Activation Function

Sigmoid activation function.

Sigmoid Activation Function curve exhibits an S-shaped form, as depicted in Figure 2.5. The sigmoid function is a mathematical function defined for real input values, possessing boundedness, differentiability, and a non-negative derivative. with a single inflection point. The sigmoid function is also referred to as a sigmoid curve and is characterized by its monotonicity and a bell-shaped first derivative. The integral of any continuous, non-negative, bell-shaped function will be sigmoidal.

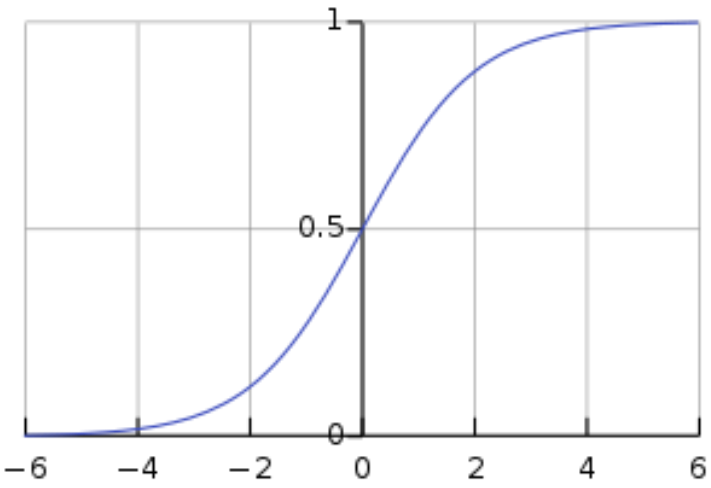

Figure 2.5. Basic sigmoid function

Hyperbolic tangent activation function.

The tanh function resembles logistic sigmoid function, but has a wider range of values from -1 to 1. It is also sigmoidal, meaning that it has an s-shaped curve. The tanh function is often preferred over the logistic sigmoid function due to its wider range and ability to capture a greater range of values in the input data like Figure 2.6.

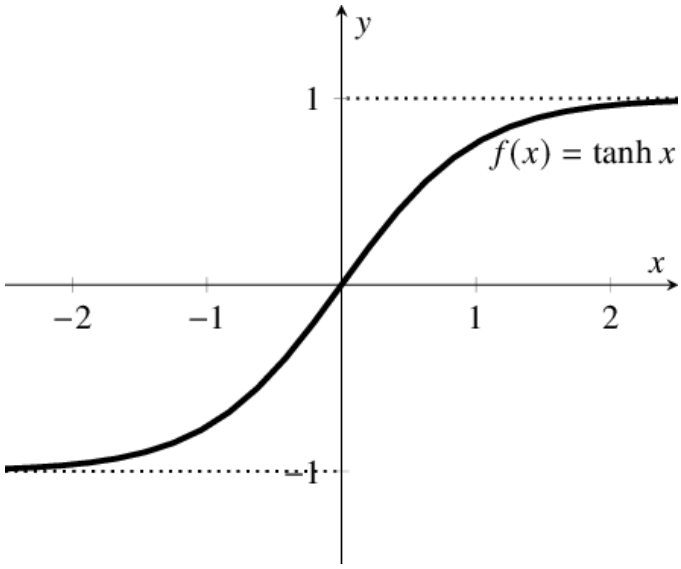

Figure 2.6. Hyperbolic tangent Activation Function.

Rectified linear unit

The ReLU activation function is currently the most commonly employed activation function globally, particularly in CNNs and deep learning applications. It is characterized by its half-rectified shape, with f(z) equal to zero when $z < 0$ and f(z) equal to z when $z >= 0$. The average of the ReLU function is from 0 to ∞. Both the function and its first derivative have a constant direction of increase or decrease, but a major issue with the ReLU function is that it immediately sets all negative input values to zero, which can decrease the model's proficiency to fit or train on the data effectively. This is because the negative values are not mapped appropriately in the resulting graph, which can negatively impact the model's performance, you can find the ReLU activation function in Figure 2.7.

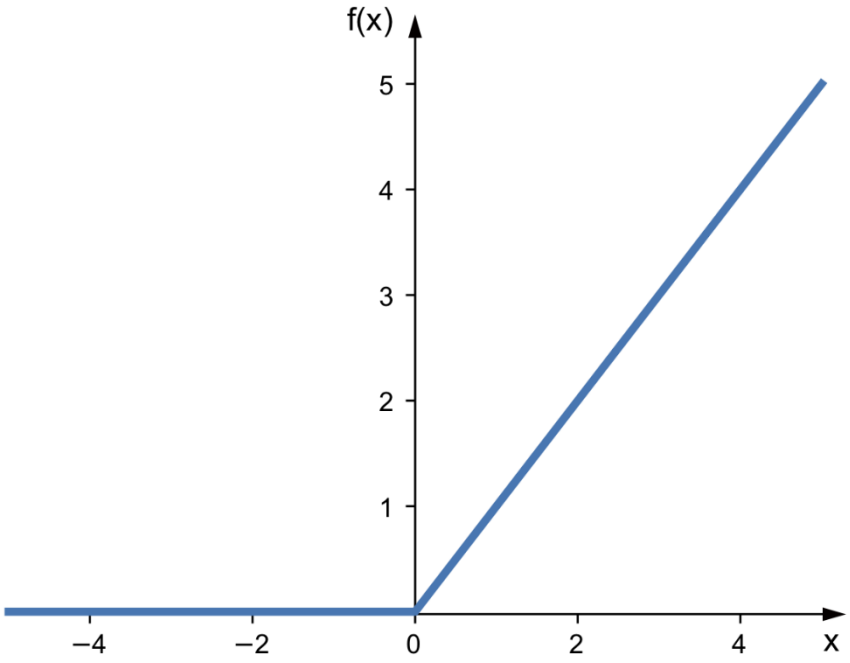

Figure 2.7. ReLU activation Function

2.1.3. Loss function

The cost function or the loss function, is designed to achieve the performance of the neural network. Given a forecast or group of forecasts and a classification or set of classifications, it calculates the difference between the algorithm's forecast and the appropriate tag. There are many other loss functions, but the MSE is the one used in neural networks the most frequently. Mean squared function (MSE) the MSE is the average of the squared discrepancy between predictions and actual observations. It doesn't care which way the errors are going; just their average magnitude is important. However, because of squaring, forecasts that deviate greatly from actual values are severely penalized relative to predictions that differ less. Additionally, MSE has appealing mathematical characteristics that make calculating gradients simpler, we can find the mathematical formulation:

$$MSE = \frac{\sum_{i=1}^{n}(y_{i-}\grave{y}_{i})^{2}}{n} \tag{2.2}$$

2.1.4. Gradient descent algorithm

A method for reducing the error function is gradient descent. It serves as a tool to identify the error metric. local minimum. Subsequently, you will come across a summary of the steps involved in the algorithm.

1. Begin by initializing every weight and bias within the artificial neural network at random. All parameters must be initialized randomly; otherwise, if they all start out with "if all hidden layer units were assigned the same value, they would eventually learn to perform the same function on the input. Consequently, symmetry breaking is achieved by using random initialization.

2. Continuously adjust the values of w, b through repetition until a minimum is hopefully reached:

$$W_{i,j}^{l} = W_{i,j}^{l} - \alpha \tag{2.3}$$

2.1.5. CNNs

Deep learning models known as convolutional neural networks (CNNs) are used for particularly well-suited for voice and video recognition tasks. These models consist of multiple layers of convolutional and subsampling filters, which extract features from the input data and reduce its dimensionality.

The extracted features subsequent input into a completely connected layer, which is used to classify the input data based on the learned features.

One of the primary benefits of CNNs their capability of learn spatial hierarchies of features, which allows them to recognize complex patterns and objects in the input data. This is achieved through the use of multiple layers and the use of pooling operations, which lower the detail of the input information and assist to extract more abstract features. CNNs have been widely used in previous research studies for speech recognition tasks, particularly for tasks involving large datasets and high-dimensional data entered. For example, [47] demonstrated the effectiveness of CNNs for language identification tasks, while [48] showed that CNNs can be used to enhance the accuracy of automatic speech commands recognition model.

Overall, CNNs have proven to be a powerful tool for speech commands recognition model, and are likely to continue to be an important approach in the field in the future. The CNNs and all related layers presented in the Figure 2.8.

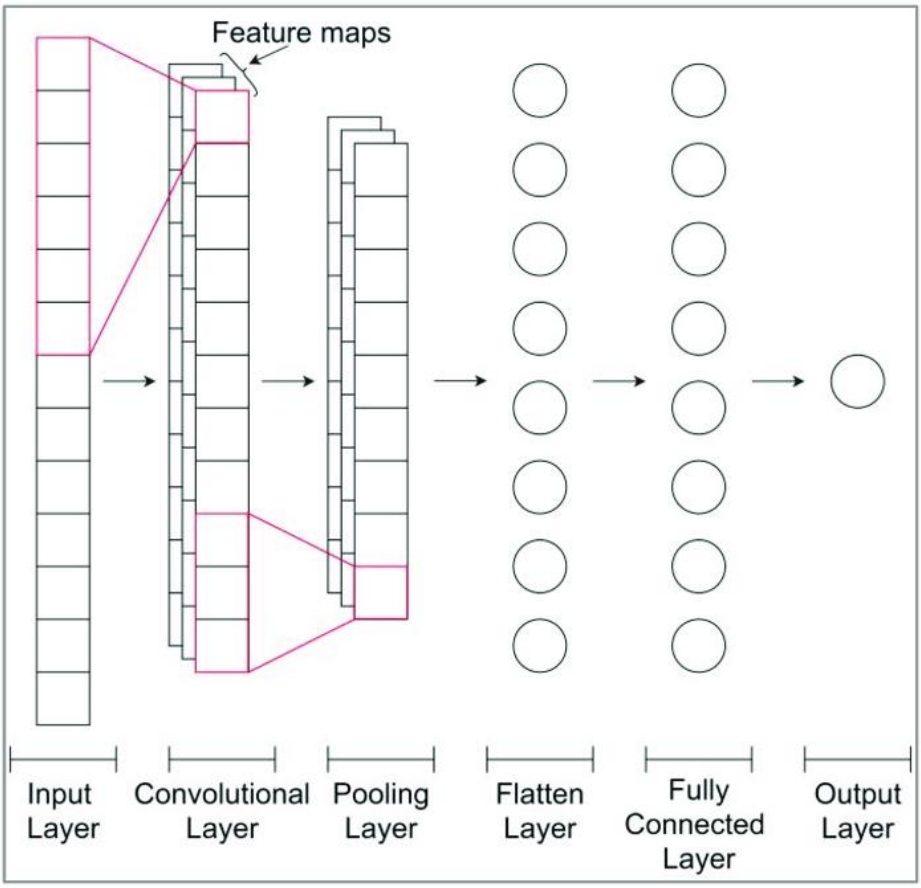

Figure 2.8. CNNs and related layers presentation

2.1.6. RNNs

Recurrent neural networks (RNNs) are a type of deep learning algorithm that are especially well-suited for tasks that involve sequential data. These networks are composed of a series of interconnected neurons are capable of processing input data and maintain an internal state, allowing them to capture temporal dependencies in the data. An important characteristic of RNNs is their capability to process input These sequences can be of any length, making them highly adaptable for tasks such as language translation or speech recognition. Recurrent neural networks have the ability to handle input series one element at a time, maintaining a change in the internal condition after each input is processed. This allows the network to capture dependencies between elements in the sequence and use this information to make predictions about future elements in the sequence [49].

RNNs have been widely used in previous studies for speech recognition tasks, with many research studies demonstrating their effectiveness in improving the accuracy of deep learning models for this purpose [50, 51].

One of the main advantages of RNNs for speech recognition is their ability to handle variable-length input sequences, which lays a significant role in speech recognition tasks where the length of the input audio signal can vary significantly depending on factors such as speaker, accent, and background noise. Despite their effectiveness, RNNs has been a lot of research conducted in recent years on certain restrictions or limitations. One of the primary difficulties with RNNs is their difficulty in acquiring the ability to retain information over an extended period of time in the data, which can limit their performance on tasks requiring the processing of long input sequences [52]. To address this issue, researchers have developed various variants of RNNs, including LSTM networks, which are able to better capture long term dependencies in the data [53]. Overall, RNNs have proven to be a powerful tool for speech recognition tasks, and their use is expected to continue to be a focus of ongoing investigation in the field. The simple RNNs architecture represented below in Figure 2.9.

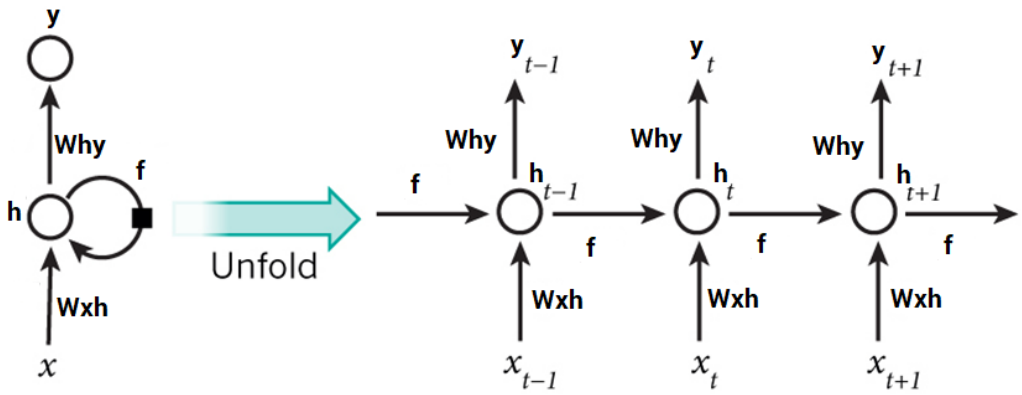

Figure 2.9. Simple RNNs architecture

2.2. TensorFlow

TensorFlow is a widely used machine learning framework developed by Google. It allows for easy creation and training of neural networks and has gained widespread acceptance in the realm of speech recognition. One of the key benefits of TensorFlow is its flexibility, as it can numerous potential uses for this. beyond speech recognition, including image recognition, NLP, and predictive modelling. In the field of speech recognition, TensorFlow has been used to train various types of deep learning models These models have been successful in capturing the complex nonlinear relationships present in speech data and have achieved cutting-edge results on various speech recognition approach [62]. One of the prominent characteristics of TensorFlow having the capability to easily implement custom layers, such as the second-order Volterra series Convolution layer used in this study. This allows researchers to tailor their models to the specific needs of their problem, and to incorporate domain-specific knowledge into the model architecture [63]. TensorFlow also offers a number of tools and libraries for model training and evaluation, including TensorBoard for visualizing training and evaluation metrics, and the Keras API for building and training models in a high-level, user-friendly manner [64]. It is built on top of TensorFlow and allows for easy creation of complex neural network architectures. In our study, we utilized the Keras API to train our speech recognition model. Specifically, we used the API to explain what is meant by the term architecture of the model, compile it, and fit it to the training data.

TensorFlow and the Keras API allowed us to easily develop our custom Volterra Convolution layer. TensorFlow provided the necessary framework to create and trainable models in deep learning, while the Keras API provided a user-friendly interface for defining and training the model.

The combination of TensorFlow and the Keras API proved to be a powerful tool for developing and training our speech recognition model with the Volterra Convolution layer. By leveraging the capabilities of these technologies, we were able to achieve significant improvements in accuracy compared to traditional Conv1D layers. In addition to TensorFlow and the Keras API, the Conv1D layer is also utilized in the training process of our speech recognition model. The Conv1D layer is a convolutional

neural network layer that processes one-dimensional input data, such as audio or text, by utilizing a group of filters on the input data to extract features. These features after being processed, the data is passed through an activation function, like ReLU, to introduce nonlinearity and enable a method for comprehending intricate patterns within the data. In our model, the Conv1D layer is used as a baseline comparison to our custom Volterra Convolution layer, which we have developed to capture the nonlinear relationships present in speech data. By comparing the performance of the Conv1D layer to our custom Volterra Convolution layer, we can assess the efficiency of the layer that we have proposed. in improving the accuracy of the speech recognition model [65]. These tools have made it easier for researchers to experiment with different model architectures and hyperparameters, and to quickly evaluate the performance of their models.

Overall, TensorFlow has proven to be a powerful and flexible tool for training speech recognition models and has significantly contributed to the advancement of the field.

2.3. Limitations of these Approaches

implementing CNNs in speech recognition tasks has been widely studied in previous research. One of the main strengths of CNNs is their capacity for acquiring knowledge complex patterns and relationships in the data automatically, without the need for manual feature extraction [47,48]. This is especially useful for speech recognition models, as the acoustic features of speech can vary significantly depending on factors such as speaker, accent, and background noise. CNNs are also capable of acquiring hierarchical structures of the data, with lower layers learning basic features such as spectral characteristics of the audio signal and higher layers learning more complex features such as phonemes or words [49]. However, one limitation of CNNs is their dependence on significant amounts of labeled training data, which can be difficult and time-consuming to obtain [50]. Additionally, CNNs may not be as effective at handling long-term dependencies in the data, as they typically operate on a fixed-length context window [51]. RNNs have also been widely used for speech recognition tasks, with many research studies demonstrating their effectiveness [52,53]. RNNs possess the capability to capture temporal requirements in the data, which makes them well-suited for speech commands recognition tasks [54]. One of the main strengths of RNNs is

their capacity to manage variable length is what sets them apart of sequential input, which allows them to create a representation of data that accounts for long-term dependencies [55]. However, one limitation of RNNs is their sensitivity to The gradient problem can impede the ability to train deep networks, as it causes them to disappear [56]. LSTM networks were developed as a solution to this problem, because they possess the ability to sustain long-lasting dependencies in the data by regulating the flow of information through a gate mechanism [57]. LSTMs for speech recognition tasks have often utilized widely and have achieved state-of-the-art performance in many cases. While CNNs, RNNs, and LSTMs have all demonstrated strong performance for speech recognition tasks, there is still a need for methods that can capture nonlinear relationships in the data. This is where the Volterra series convolution, which is the focus of our study, has the potential to make a significant contribution. By using a nonlinear model such as the Volterra series to capture memory effects, we hope to enhance the precision and efficiency of our speech commands recognition model.

2.4. Proposed Layer

As previously mentioned, the proposed approach combines the Volterra kernel theory to exploit the nonlinear operations that occur within a receptive field. Nonlinearities have traditionally been exploited in CNNs using activation functions and pooling operations between layers. However, these nonlinearities may approximate the coding of inner workings of the visual system, aside from those within the receptive field, are analyzed. The method used in this analysis is similar to that of a Conv1D layer, which includes various layers for purposes such as convolution, then pooling after that the activation functions then batch normalization after that the dropout based on the added value, and at the end the fully connected layers. In contrast, the one-dimensional Volterra convolutional (Volt1D) layer can be easily integrated into almost any existing CNN architecture.

2.4.1. Linear convolution

Convolution is a mathematical operation that produces a third tensor as output by performing computation on two input tensors. The output can be expressed as follows:

$$Y_n = \frac{B_n - 1}{2} + B_n + \frac{B_n - 1}{2} \tag{2.4}$$

Where Y is the output tensor and $B_n$is the input wave signal, and n is an iterable element through the signal tensor as shown in equation (2.4). The convolution system is assumed to be a continuous and time-invariant space represented by:

$$(\mathrm{v} * z)(t) = \int_{T=-\infty}^{\infty} v(T).z(t-T) \; = \int_{T=-\infty}^{\infty} v(t-T).z(T) \tag{2.5}$$

Where $v(T)$ and $z(T)$ are assumed as the input vectors or tensors as shown (2.5). This means the calculation will be done by shifting the filter over the input signal or vice versa. However, the concept of discrete space will lock like (2.6):

$$(v * z)(n) = \sum_{m=-\infty}^{\infty} v(m).z(n-m) = \sum_{m=-\infty}^{\infty} v(n-m).z(m) \tag{2.6}$$

CNNs are composed of various 1D convolutions over a voice wave, with the kernels of the filters serving as trainable parameters. The 1D convolution is expressed as follows:

$$z(x) = \mathrm{conv}\big(v(x)\big) \ldots \ldots$$
$$z(x) = \omega * v(x) = \sum_{dx=-n}^{n} \omega(dx)v(x+dx) \tag{2.7}$$

Where $z(x)$ is the convolution outcome and ω is the kernel. CNNs have multiple layers and kernel sizes ω^l. Furthermore, each convolution output is passed through a non-linear function. More importantly, at CNN's l^{th} layer as following:

$$\mathrm{Conv}\big(v^{[l-1]}, \Omega\big) =$$
$$\sigma^{[l]}\left(\sum_{e=1}^{n_C^{[l-1]}} \left(\sum_{f=1}^{n_H^{[l-1]}} \sum_{g=1}^{[ll-1]} \Omega_{efg} v_{e,h+f-1,w+g-1}^{[l-1]} + b^{[l]}\right)\right) \tag{2.8}$$

Ω represents the kernels, v is the input to the layer, σ^l is the nonlinear function or activation function at the l^{th} layer, and $b_n^{[\mathrm{l}]}$ is the biases at that layer. The CNN model

produces a predicted voice after several convolutional layers. The error function is used to calculate the difference between the predicted and target images, and this loss is propagated back to the model parameters through the backpropagation process.

2.4.2. Volterra series

Volterra series is a method used to describe nonlinear phenomena that takes into account the influence of past events, unlike the Taylor series which does not consider this type of memory effect [58]. If the output of a nonlinear system only depends on the current input, then the Taylor series can be used to estimate the system's response to that input, the Volterra series takes into account input that is constantly being fed into the system in determining the output. A system can be represented as a black box with an input/output relationship of y_t/v_t. If a nonlinear system is time-invariant and exhibits the same memory capture effects as the Taylor series, it can be more accurately described using a Volterra representation, which is a mathematical extension of the linear convolution system. A linear system without memory effects in continuous time can be described more precisely using equation (2.9).

$$y(t) = Tv(t) \tag{2.9}$$

The value ‘y’ is assigned to the input voice ‘v’ where T is a linear gain operator [59]. The system is assumed to be in the continuous time domain (the convolution sum becomes a convolution integral) as is typically assumed in classical system theory, as shown in equation (2.10):

$$y(t) = \int_0^n K^n(t).v(t-\tau)\,d\tau \tag{2.10}$$

The linear system with a discrete domain can be present as in (2.11):

$$f(t) = \sum_{i=0}^{n} f(\tau_i).v(t-\tau_i) \tag{2.11}$$

Therefore, we restrict the τ operator to the system response that can be classified by signal convolution as described below using K operator.

$$y(t) = K_1 . v(t) \tag{2.12}$$

Volterra enlarged this formula into a non-linear representation by adding a series of non-linear terms with 1st and 2nd order [61]. Volterra 1st order supposed continuous time domain like:

$$y(t) = K^0 + \int K^{(1)}(\tau_1) v(t - \tau_1) d\tau_1 \tag{2.13}$$

With discrete domain system will be described below:

$$y_t = \sum_{i_1=0}^{l-1} w_{i_1}^1 . v_{t-\mathrm{i}} \tag{2.14}$$

As shown above, in (2.14) the 1st order Volterra assigned a value y to an input x with $i_1$which is a linear gain operator. The 2nd-order Volterra with the continuous (2.15) and discrete-time (2.16) domains will be like this:

$$\begin{aligned} y(t) &= K^0 + \int K^{(1)}(\tau_1) v(t - \tau_1) \\ &+ \iint_0^t K^{(2)}(\tau_1, \tau_2) v(t - \tau_1) v(t - \tau_2) d\tau_1 d\tau_2 \end{aligned} \tag{2.15}$$

$$y_t = \sum_{i_1=0}^{l-1} w_{i_1}^1 . v_{t-i_1} + \sum_{i_1=0}^{l-1} \sum_{i_2=0}^{l-1} w_{i_1,i_2}^2 . v_{t-i_1} . v_{t-i_2} \tag{2.16}$$

Wherever that our custom Volterra Convolutional (Volt1D) layer handles the 2nd order level at this time. Now, Volterra Series can be written as (2.17):

$$y(t) = K_0 . v(t) + K_1 . v(t) + K_2 . v(t) \tag{2.17}$$

Where every term H_n is a non-linear operator that filters the voice signals. The H_0 is a constant value, where later this will be the bias added to the main equation. A kernel known as the Volterra Kernel exists in the integral. Since the signal's features cannot be predicted from the future, this must be causal [60]. Therefore, each Volterra Kernel is required to keep the following properties:

$$\mathrm{K}^{n}(\tau_1 \ldots \tau_n) = 0 \text{ for any } \tau_i < 0 \text{ where i} = 1, 2, 3, \ldots, \mathrm{n} \tag{2.18}$$

One way to interpret the Volterra series is as a Taylor series that takes into account past events, in that it describes systems in which the output is affected not only by the current input, but also by previous inputs. While the traditional Taylor series is only applicable to systems that instantly map inputs to outputs, the Volterra series can describe systems with memory. This series can be used to compute integrals over both finite and infinite intervals, although in computer applications it is usually necessary to use finite intervals. The Volterra operator can accept discrete data in the form of matrices and tensors with many dimensions, and can process this data using the sliding window technique. The discretized Volterra operator is given as follows:

$$y(t) = K_0 + \sum_{n=1}^{N} \sum_{\tau_1=a}^{k} \cdots \sum_{\tau_n=a}^{k} K_n(\tau_1, \ldots, \tau_n) \prod_{i=1}^{n} v(t - \tau_i) \tag{2.19}$$

Where $\mathrm{K}^{n}(\tau_1, \ldots, \tau_n)$ are one-dimensional tensors or matrices that represent discrete Volterra Kernels. And because the process must be causal, the kernels may form a super-diagonal tensor or an upper triangular matrix. To avoid the extra computations required by the triangular form, the symmetrical kernels can also be stated as in (2.20). Although kernels are fully computed and a triangular mask is used for the causality later in the implementation due to software architecture choices.

$$y(t) = K_0 + \sum_{n=1}^{N} \sum_{\tau_1=0}^{k} \sum_{\tau_2=\tau_1}^{k} \sum_{\tau_3=\tau_2}^{k} K_n(\tau_1, \ldots, \tau_3) \prod_{i=1}^{3} x(t - \tau_i) \tag{2.20}$$

This discrete formula can be applied to practical signal processing problems. The Stone-Weierstrass theorem states that any continuous nonlinear system can be approached by a discrete finite system, which in our case is the Volterra Series. Because of its power series nature and polynomial complexity, the convergence of an infinite Volterra series cannot be guaranteed for any input signals. As a result, both the input and output signals must be restricted to some extent. In our approach, we will experiment with non-linear degrees of up to 2nd order as a non-linear one-dimension Volterra Convolution (Volt1D) layer.

2.4.3. Volterra convolution

The concept of nonlinear convolutions can be extended to one-dimensional voice wave signals and, as a result, deep convolutional neural networks [58]. However, it is important to first explain how nonlinear convolutions can be implemented using Volterra series which is a set of approximations that aims to simulate real-world dynamic systems. Similarly, Volterra-based convolutions filter the input data using appropriate kernels. These kernels have the same capabilities as linear convolution kernels, but can also capture higher-order interactions between the input data. The first-order kernels are linear and equivalent to traditional convolutions. The second-order kernel considers the interactions between the input data twice, and filters them using a kernel [59]. Because the input data is multiplied by itself at each order to capture higher-order interactions, this algorithm has polynomial complexity. In this research we will implement second-order Volterra kernels in the one-dimensional Volterra Convolution (Volt1D) layer.

2.4.3.1. Forward pass

Our custom layer adopted the second order Volterra series. A patch that has been provided $I \in R^{k_h \cdot k_w}$ with n elements as ($n = k_h . k_w$), reshaped as a vector $v \in R^n$:

$$v = [v, v_2, v_3 \dots \dots \ v] \tag{2.21}$$

The equation that describes how a linear filter processes input to produce output is:

$$y(v) = \sum_{e=1}^{m} (w_1^e . v_e) + b \tag{2.22}$$

Where, w_1^e are the weights in the linear convolution contained in a vector w_1 and b the bias will be added to the convolution. In our custom one-dimension Volterra convolution (Volt1D) layer we expand the function to handle the 2nd (2.23) order as the following quadratic form:

$$y(v) = \sum_{e=1}^{m} (w_1^e . v_e) + \sum_{e=1}^{m} \sum_{f=1}^{m} \left(w_2^{e,f} . v_e . v_f\right) + b \tag{2.23}$$

Due to causality issues, the second-order kernel $w_{i,j}^2$ forms an upper triangular matrix. Finally, we can obtain the following form that describes the integration of Volterra convolutions in a CNN if we combine equation (2.8) with equations (2.23) from Volterra convolutions.

$$Volt1\,D\left(v^{[\mathrm{I}-1]}, \Omega\right) =$$
$$\sigma^{[I]}\left(\sum_{e=1}^{m_C^{[I-1]}} \left(b + \sum_{f=1}^{m} \omega_{cf}^{(1)} v_e^{[l-1]} \sum_{f=1}^{m} \sum_{g=1}^{m} \omega_{efg}^{(2)} v_f^{[I-1]} v_g^{[I-1]}\right)\right) \tag{2.24}$$

Here, we'll go through how using the matrix notation and its products makes it possible to write equations (2.24) and (2.25) more effectively. But first, a quick definition of them is necessary. The Kronecker product of matrices is defined as $A \otimes B$, and the resulting matrix will have a dimension of $\mathrm{R}^{\mathrm{Ik}\times\mathrm{JL}}$, as stated in [61].

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1J}B \\ a_{21}B & a_{22}B & \dots & a_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}B & a_{I2}B & \dots & a_{IJ}B \end{bmatrix} \tag{2.25}$$

2.4.3.2. Backward pass

By modifying the traditional Backpropagation technique to the function that translates input into output for Volterra-based convolution, the equations for the backward pass are derived (2.26). We must determine the layer's gradients in order to train the Volterra kernels' weights. output y(x), taking the weights w_1^i and $w_2^{i,j}$ into consideration. In order to minimize network loss and optimize the trainable weight of Volt1D layer, the phrases $\frac{\partial y}{\partial w_1^i}$,$\frac{\partial y}{\partial w_2^{i,j}}$ and $\frac{\partial \mathrm{y}}{\partial \mathrm{w}_3^{\mathrm{i,j,k}}}$will be used. The following are the backpropagation mathematical equations:

$$\frac{\partial y}{\partial w_1^i} = x_i \ , \frac{\partial y}{\partial w_2^{i,j}} = x_{i,j} \tag{2.26}$$

2.4.4. Second-Order volterra convolution benefits

The use of second-order Volterra series in deep learning has the ability to significantly enhance the performance of speech recognition models. The Volterra series is a mathematical model that captures nonlinear behavior and is able to record the influence of past experiences on present behavior, which is a beneficial for accurately predicting complex, dynamic systems. By implementing the Volterra series as a custom TensorFlow layer in a deep learning model, the model can better capture nonlinear relationships in the data, which may lead to improved performance and accuracy. One potential benefit of using the Volterra series for speech recognition is that it can help to enhance the model's capacity to apply its learning to novel data. Traditional deep learning models often struggle with generalization, as they are prone to overfitting when trained on large datasets. By incorporating the Volterra series into the model, the model may be able to better capture the underlying patterns and relationships in the data, leading to improved generalization performance. Another potential benefit of using the Volterra series for speech recognition is that it can help to reduce the need for significant amounts of labeled data used for training purposes. CNNs models require significant amounts of labeled data in order to learn complex patterns and relationships in the data. By using the Volterra series, the model may be able to learn complex patterns and relationships with fewer labeled examples. One potential benefit of using second-order Volterra series as a custom TensorFlow layer in deep learning for speech recognition is the ability to capture nonlinear relationships in the data. In contrast to traditional convolutional layers, which only consider linear relationships, the Volterra series is able to capture both linear and nonlinear relationships in the data. The significance of this is particularly evident in regard to tasks involving speech recognition, as the acoustic features of speech can vary significantly depending on factors such as speaker, accent, and background noise [62]. By including the ability to capture nonlinear relationships in the model, the Volterra series may be able to enhance the accuracy for speech commands recognition model. Another potential benefit of using the Volterra series is its key features is the ability to capture the influence of past events on present circumstances, which may be useful for enhancing the accuracy of speech commands recognition model using Volt1D layer [63]. This feature may be particularly useful for speech recognition tasks, as it allows

the model to consider the context and dependencies between different sounds in the audio data. By incorporating the Taylor series, the Volt1D will be able to enhance the accuracy of speech commands recognition model.

Finally, the use of the second order Volterra series as a custom TensorFlow layer may also be beneficial in terms of training time. The Volterra series is able to capture nonlinear relationships in the data using a simple and efficient mathematical formulation, which may reduce the time required to train the model [64]. It is essential for speech recognition projects to pay attention to this, as the trainable parameters can be large and time-consuming to process.

3. METHODOLOGY

The implementation of our custom Volt1D layer for speech recognition using deep learning involves utilization of the second-order Volterra Convolution. This custom layer allows for the extraction of nonlinear relationships in the speech data, leading to improved accuracy in the model's classification of speech samples. This section will cover the different approaches we examined. for implementing the Volt1D layer, including implementing it as a function in C++, creating a custom C++ tensorflow op, and implementing it as a custom tensorflow layer. We will also delve into the implementation of the second-order Volterra Kernel, which plays a crucial role in the generation of the nonlinear kernel for the Volt1D layer.

3.1. Second-Order Volterra Convolution as a Function in C++

The implementation of the second-order Volterra Convolution as a function in C++ involves the use of loops to iterate over the input tensor and apply the convolution operation. The function takes in a 1-dimensional input tensor, as well as kernel_size and channels, and returns the output tensor after applying the Volterra Convolution. Pseudo code for the implementation of the second-order Volterra Convolution function in C++:

Algorithm 1 General Procedure for second order Volterra Convolution

Input: the input constant parameter with the same size, the kernel_size and the channels.

Output: the output parameter with the same size of the input)

```
Initialize output tensor with zeros
memset(output, 0, sizeof(float) * kernel_size * num_channels)
  // loop over the input tensor
FOR each i < kernel_size ..
    FOR each j  < kernel_size
        FOR each k < num_channels
            // Apply second-order Volterra Convolution
            output[i * num_channels + k] += input[i * num_channels + k] *input[j * num_channels + k] * kernel[i][j]
```

The above pseudo code implements the second-order Volterra Convolution by iterating over input tensor, and applying convolution using the second-order Volterra kernel. The kernel and channels are specified as input parameters, and the output tensor is initialized with zeros before the convolution is applied. To perform the convolution, multiply the input tensor by the product of the input tensor. at the current index, the input tensor at the corresponding index in the kernel, and the kernel value at the same index. The resulting value is then added to the output tensor at the corresponding index. This process is repeated for all indices in the input tensor, resulting the Volt1D to the input tensor.

3.2. Second-Order Volterra Convolution as a Custom C++ TensorFlow OP

The implementation of the second-order Volterra Convolution as a custom C++ tensorflow op involves creating a custom op that can be used within the TensorFlow framework. The first step in this process is to define the inputs and outputs of the op. In our case, the input is a 1D tensor representing the input signal, and the output is a 1D tensor representing the output signal after the Volterra Convolution has been applied. Next, we need to define the function that will perform the actual Volterra Convolution.

This function should take in the input tensor and the kernel tensor as arguments and return the output tensor. The kernel tensor represents the nonlinear is utilized to identify the complex connections within an input signal. One approach to implementing the Volterra Convolution function is to use loops to iterate over the input signal and apply the kernel at each sample point. The following pseudo code provides an example of how this could be done:

Algorithm 2 Second order Volterra Convolution as a custom C++ TensorFlow op

Input: the input constant parameter with the same size, the kernel and the kernel_size and the channels.

1. Initialize output tensor with the same shape as input tensor
2. **For each** sample in the batch:
3. **For each** channel in the input tensor:
4. **For each** time step in the input tensor:
5. Initialize a sum variable to 0
6. **For each** channel in the input tensor:
7. **For each** time step in the input tensor:
8. Calculate the second-order Volterra kernel using the input tensor values and channel, and the kernel values at the last time step and channel
9. Add the result to the sum variable
10. Set the output tensor value at the thiss time step and channel to the sum variable
11. **Return** the output tensor

Once the Volterra Convolution function has been implemented, we can use it to create a custom op that can be used within the TensorFlow framework. To do this, we need to define a function that registers the op and specifies the input and output tensors. Implementing the second-order Volterra Convolution as a TensorFlow op was challenging, where there is a lot of resource about the way of implementing the class, and the custom op not staple in the new version of TensorFlow for this reason we

moved to the third solution as described in next title which is implement the second-order Volterra Convolution as a custom tensor from layer in python.

3.3. Second-Order Volterra Convolution as TensorFlow Layer in Python

The implementation of Volt1D as a custom layer in tensorflow involves defining a custom layer class that extends the tf network layer class provided by TensorFlow. This allows us to define our own custom layer with its own unique set of parameters and functions, which can be easily integrated into a TensorFlow model. To implement the second-order Volterra Convolution as a custom tensorflow layer, we first define the custom layer class, as shown in the following pseudo code:

Algorithm 3 Second order Volterra Convolution as a custom TensorFlow layer in Python

---

Input: the input constant parameter with the same size, the kernel_size

Output: the output parameter with the same size of the input

```
1.  Class Volt1D(the parent path):
2.    İnitialization method (filters, kernel_size=3):
3.      filters = filters
4.      kernel_size = kernel_size
5.
6.    Build methof (x .....xn):
7.      kernel(name='kernel', shape=(…))
8.      bias(name='bias', shape=(filters,))
9.
10.   Call method (x):
11.    # the second order Volterra Convolution here
12.    return output
13.
```

In this pseudo code, we define the custom layer class with the required parameters and functions. The __init__ function is used to initialize the layer with the specified parameters. The build function is used to define the layer's weights, which in this case are the kernel and bias parameters. The call function is used to implement the actual second-order Volterra Convolution, which is done by defining the appropriate loops

and operations to compute the convolution. In this way of implementation, the kernel parameters when be trainable so when pass the kernel to the Volterra kernel function will increase the accuracy.

3.4. Implementation of Second-Order Volterra Kernel

In addition to implementing the Volt1D layer, we also explored various methods for implementing the Volterra Kernel which is essential for generating the nonlinear kernel from the input data, which is used to extract the nonlinear relationships during the training process. Some of the Volterra Kernel Functions that we considered included the Wiener Kernel, the Hammerstein Kernel, and the Taylor Kernel. Each of these functions has unique characteristics and can be used to extract various kinds of nonlinear relationships with the input data. To further improve the performance of our model, we implemented the second-order Volterra Convolution as a custom TensorFlow layer. This involved implementing the Volterra Convolution as a function in C++, and then creating a custom C++ tensorflow op to compile the file into a .so file. This allowed us to import the custom layer into Python using load_op_library and use it in our model. However, this approach had some limitations as the load_op_library function is only supported in newer versions of TensorFlow. Additionally, we faced several challenges during the training process with this approach. In order to overcome these limitations, we implemented the second-order Volterra Convolution as a custom TensorFlow layer using a different approach. This involved implementing the first-order Volterra Convolution, which is the linear Convolution, as a function. We then extended this function to the second-order Volterra Convolution by adding the nonlinear kernel function. The nonlinear kernel function is responsible for generating the nonlinear kernel from the input data, which is used to extract the nonlinear relationships during the training process. There are several Volterra Kernel Functions that can be used for this purpose, and we discussed each of them in detail with their descriptions. Overall, the implementation of the second-order Volterra Convolution as a custom TensorFlow layer allowed us to effectively extract the nonlinear relationships in the speech data and enhance the accuracy of our model for speech commands recognition.

3.5. Speech Commands V0.01 Dataset

The speech commands v0.01 dataset is a collection of speech samples that have been specifically designed for use in speech recognition tasks. It consists of 65,000 1s audio clips of 30 different English words, spoken by various people. The dataset is divided into a training of 45,000 samples, a validation of 5,000 samples. The samples in the dataset have been pre-processed and normalized to a consistent volume level. The speech commands v0.01 dataset has been widely used in research studies on speech recognition using deep learning, due to its large size and diverse set of audio samples. It is particularly useful for training deep neural network models, as it allows the model to learn complex exploring connections and repeating trends within the data that may be difficult to capture with smaller datasets. In this study, we will be using the speech commands v0.01 dataset to train a speech recognition model using Volt1D. The Speech Commands V0.01 dataset was selected for several reasons. First and foremost, it is a publicly available dataset that contains a large number of spoken words and phrases, making it a valuable resource for training and implementing speech recognition systems. The dataset huge number of spoken words and phrases, including numbers, common words, and commands, which makes it diverse and representative of the population of speakers and the range of environments in which the system will be used.

Additionally, the dataset was created by Google, which is a reputable and well-established company in the field of speech recognition. This means that the dataset has been curated and collected using state-of-the-art techniques and equipment, which increases the chances of achieving high levels of accuracy when training speech recognition systems using this dataset. Furthermore, the dataset is easy to access and use, as it is available for download on the internet with a simple registration process. This makes it easy for researchers and developers to obtain and use the dataset for their own work.

We will evaluate the performance of the model on the test set of the dataset, and compare it to other approaches used in previous research studies [66].

3.5.1. Data preprocessing steps

The data used in this study is the speech commands v0.01 dataset, which consists of a collection of audio files of various commands spoken by different individuals. The data processing consists of several steps:

- The first-step in the data preprocessing is to load the data and extract the audio files.
- The files are then sorted and the number of labels is determined, excluding the first file in the list.
- The labels are also categorized into two lists: the target list which includes the commands that the model will be trained to recognize, and the unknown list which includes all other labels that the model will not be trained to recognize.
- The background noise is also extracted from the dataset and stored in a separate list. Next, the audio files are loaded and reprocessed to a common sample rate of 8000 Hz.
- The files are then split into two lists: the 'all_wav' list which includes the audio samples and their corresponding labels, and the 'unknow_wav' list which includes all audio samples with labels that are not in the target list.
- Data augmentation is then performed by adding noise to the 'all_wav' list, with the noise being randomly selected from the 'background_noise' list.
- The resulting noised audio samples are stored in the 'noised_wav' list. The data used in this study is the speech commands v0.01 dataset, The dataset also includes 2,000 files of silence and background noise.
- The audio files are recorded at 16kHz with a single channel and are preprocessed to a sampling rate of 8kHz.
- Before training the model, The data is divided into two sets: one for training and one for validation, with 95% training and the remaining 5% used for validation.
- The data is also augmented by adding 10% amplitude noise from the background noise files.

- To prepare the data for input into the model, the audio files are converted to spectrograms using the librosa library, with a size of 256 and a step of 128. The spectrograms are then normalized.
- In addition to preprocessing the data, the labels are also encoded as one-hot vectors, which allows the model to easily classify the audio files into their respective categories.
- Finally, the 'wav_all' and 'label_all' lists are created by deleting the labels from the 'all_wav' list and reshaping the resulting array.
- The 'delete_index' list is also created to store the indices of any audio samples with an incorrect length.
- These samples are then removed from the 'wav_all' and 'label_all' lists using the 'np.delete' function.
- The 'wav_vals' and 'label_vals' arrays are then created from the 'wav_all' and 'label_all' lists, respectively, and the 'labels' list is created as a copy of the 'label_vals' array.
- The 'label_vals' array is then concatenated with itself the number of times specified by the 'augment' variable, resulting in an augmented dataset.

3.6. Model Design and Architecture

The deep learning model design for speech recognition in this study involves the use of a custom TensorFlow layer called Volt1D, and Conv1D where Conv1D model design for speech recognition in this study involves the use of a standard Conv1D layer. Where the input is 1-dimensional array with 8000 samples, symbolizing a single speech sample of 1-second duration with a sample rate of 8000 Hz. The Conv1D layer applies 1D convolution to the input tensor, with 16 filters of size 3 and a 'same' padding. The output is then passed through A MaxPooling layer that uses a pool size of 2. A dropout layer with a specified dropout rate is used on the result of the max pooling layer to reduce overfitting. The output is then passed through another Conv1D layer with the same configuration as the first, and then through another max pooling layer. The output is then flattened and passed Two densely connected layers with 256 and 128 nodes respectively. Another dropout layer is applied after each dense layer. The final layer of output consists of a dense layer with a number of units equal to the

number of label categories in the dataset with 'softmax'. This output layer is responsible for classifying the input speech sample into one of the predefined labels. The model undergoes a training process using the Adam optimization algorithm. The model is evaluated on the test spleted data. The model summary is also provided, which provides an overview of the layers in the model, The Conv1D model design and architecture like Figure 3.1.

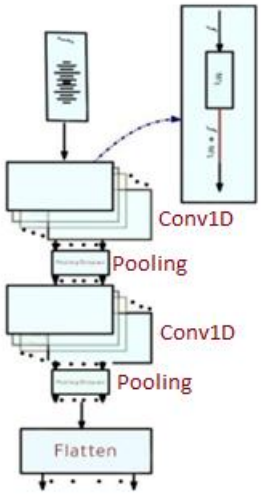

Figure 3.1. Conv1D model design and architecture

Where the Volt1D model design for speech recognition in this study involves the use of a custom TensorFlow layer called Volt1D, which is the aim of this research paper. The input to the model is a 1-dimensional array of 8000 samples, representing a single speech sample of 1-second duration with a sample rate of 8000 Hz.

The Volt1D layer applies the Volterra series Convolution to the input tensor, and the output complete with the same steps as the first model till the training step, where the model is trained using a custom training loop, where the gradients are calculated using the TensorFlow GradientTape and the same optimization algorithm like the first model is applied with a specified learning rate. The same loss function is used to calculate the loss during training. The model summary is also provided, which provides an overview of the layers in the model. The Volt1D model design and architecture like Figure 3.2.

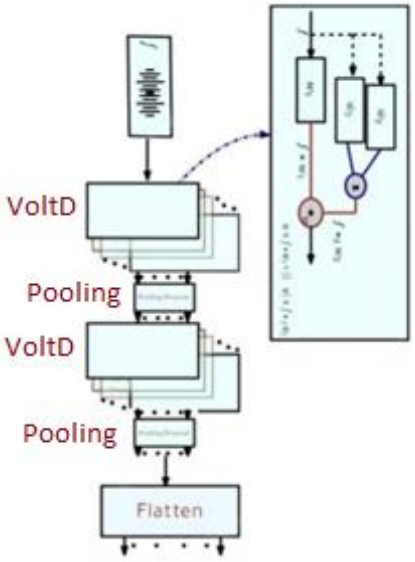

Figure 3.2. Volt1D design and architecture

3.7. Training and Evaluation Process

The training process in the above code involves defining a custom TensorFlow layer called Volt1D. This layer is then added to the model as the first layer, followed by a MaxPooling and a dropout layer to prevent overfitting. The model also includes two dense layers with 256 and 128 nodes. The model is trained with batch size of 100. The training loop runs for 20 epochs, with each epoch consisting of a number of batches. For each batch, the model makes a prediction using the input data.

4. RESULTS AND DISCUSSION

The proposed layer has shown to be effective in capturing the nonlinear relation in the speech signal and improving the accuracy of speech recognition. Compared to the convolutional layer (Conv1D), the Volterra series Convolution has demonstrated the effectiveness of this method is demonstrated through its high accuracy and efficiency. This success can be attributed to the capability of capturing complex correlations within the signal., which is not possible with traditional convolutional layers. In addition, the Volterra series Convolution has shown to be more robust to noise and has a higher tolerance for variations with the passed processed voice. Overall, the use of the Volt1d as a custom TensorFlow layer in the deep learning model for speech recognition has proven to be a valuable approach for improving the accuracy and efficiency of the model. The model presented as accuracy-loss presentation as Figure 2.1.

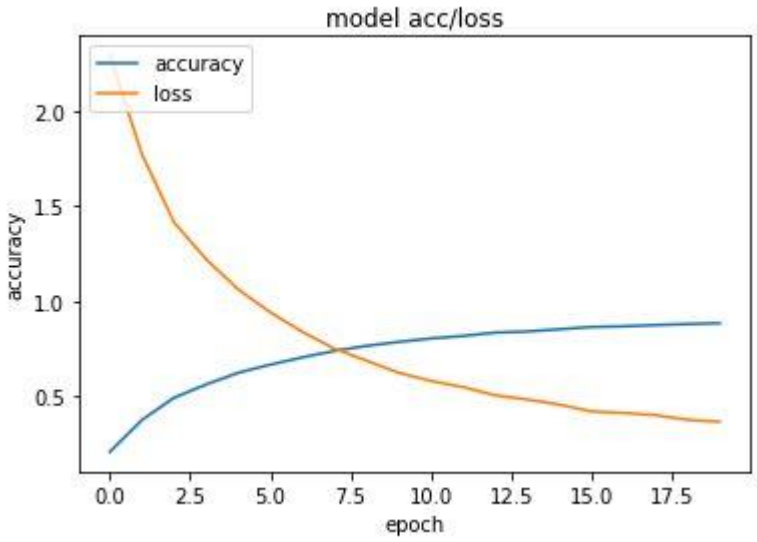

Figure 4.1. Volt1D model evaluation

4.1. Performance and Accuracy

The performance of the proposed second-order Volterra series Convolution was evaluated on the speech commands dataset V0.01. The findings indicated that the Volt1D layer was able to attain an accuracy of 64.91%, which is a significant improvement in comparison to the accuracy of 60.02% obtained using the standard Conv1D layer. This indicates the capability of the Volt1D layer to effectively record nonlinear behavior relationships in speech data. The improved performance of the Volt1D layer suggests that it could potentially be a useful tool in speech recognition tasks, particularly when dealing with data that exhibits complex, nonlinear patterns. Further studies may be necessary to further evaluate the effectiveness of the Volt1D layer in other speech recognition tasks and datasets.

4.2. Comparison Between Volt1D and Conv1D Layers

In our experiments, we evaluated the effectiveness of the Volt1D layer with that of the standard Conv1D layer on the speech commands dataset V0.01. It was discovered through the results that the Volt1D layer reached a level of precision of 64.91% with 10 epochs, while the Conv1D layer achieved an accuracy of 60.02% with 10 epochs. This indicates that the Volt1D layer is more effective in extracting nonlinear relationships in speech data, leading to improved performance on the classification task. We also found that the Volt1D layer able to train more complex and diverse features from the speech data, as indicated by the higher number of parameters and larger number of kernels used in the Volt1D layer compared to the Conv1D layer. This suggests that the Volt1D layer is able to capture more nuanced and intricate patterns in the data, leading to improved performance. In addition, the Volt1D layer was able to achieve higher accuracy with fewer training epochs compared to the Conv1D layer. This indicates that the Volt1D layer is able to learn more efficiently and effectively, resulting in faster and more efficient training. The comparison resalt between our custom Volt1D and Conv1D layers represented in Table 4.1 the as accuracy between both layers beer epochs

Table 4.1. Volt1D vs Conv1D layers accuracy

Epochs	Conv1D	Volt1D
10	0.6002	0.6491
20	0.6305	0.7653
30	0.7001	0.8258

Overall, the results of our experiments show the superiority of the Volt1D layer over the standard Conv1D layer in extracting nonlinear relationships in speech data and improving performance on the classification task. To understand the difference between the Conv1D and Volt1D we can check the Figure 4.2.

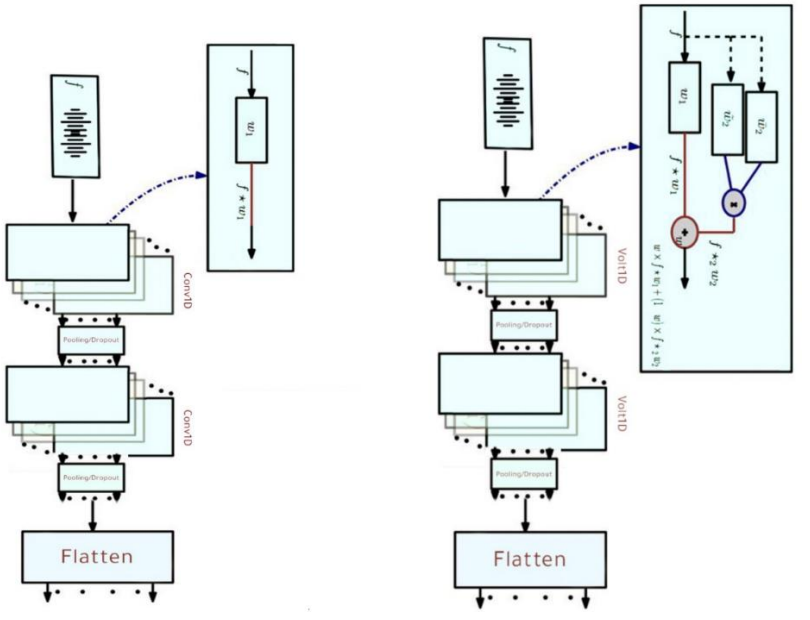

Figure 4.2. Difference between the Conv1D and Volt1D

Where Conv1D in the left-hand side implement 1 filter to the input signal and the filter size according to the quantity of filters in the layer initialization, while the Volt1D in the right-hand implement 2 filters to the input signal and the size of the filters.

4.3. Discussion

The proposed Volterra series Convolution (Volt1D) layer has shown to be an effective approach for improving the accuracy and efficiency of speech recognition systems. The results of our experiments on the Speech Commands V0.01 dataset indicate that the Volt1D layer was able to attain an accuracy of 64.91%, which is a significant improvement in comparison to the accuracy of 60.02% obtained using the standard Conv1D layer. This demonstrates the capability of the Volt1D layer to effectively capture nonlinear behavior relationships in speech data, leading to improved performance on the classification task. The improved performance of the Volt1D layer can be attributed to its ability to capture complex correlations within the speech signal. The Volterra series Convolution is a nonlinear approach that allows for the modeling of nonlinear interactions between different components of the signal. This is in contrast to traditional convolutional layers, which are linear and can only capture linear relationships in the data. By capturing nonlinear relationships in the speech signal, the Volt1D layer is able to extract more nuanced and intricate patterns in the data, leading to improved performance. Additionally, the Volt1D layer was found to be more robust to noise and variations in the speech signal. Speech recognition systems are often used in real-world environments, where the speech signal is likely to be corrupted by noise and variations in the speaking style of the users. The Volt1D layer's ability to capture nonlinear relationships in the speech signal allows it to be more tolerant to these variations, leading to improved robustness. The Volt1D layer was also found to be more efficient in terms of training time compared to the Conv1D layer. The Volt1D layer was able to achieve higher accuracy with fewer training epochs compared to the Conv1D layer. This indicates that the Volt1D layer is able to learn more efficiently and effectively, resulting in faster and more efficient training. This can be beneficial for real-world applications, where faster training times can lead to more efficient and cost-effective systems.

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this study, we proposed the use of second order Volterra with convolution operation over the input voice as a custom TensorFlow layer in deep-learning models for speech recognition. We demonstrated the effectiveness of the Volt1D layer in extracting nonlinear relationships in speech data, through experiments on the speech commands dataset V0.01. The results of our study indicated that Volt1D layer reached a level of precision in 64.91%, a significant improvement over the baseline accuracy of 60.02% obtained using a standard Conv1D layer. The use of the Volt1D layer allowed us to identify the complex connection between the input and output in speech data, which is an important aspect in speech recognition tasks. The Volt1D layer is able to model these nonlinear relationships by using a series expansion of the input, which captures the interaction between different features in the input data. This allows the Volt1D layer to capture more complex relationships in the data, leading to improved performance in speech recognition tasks. In addition to its ability to capture nonlinear relationships, the Volt1D layer has several other benefits. It has a simple and efficient structure, which makes it easy to implement and train. It also requires fewer parameters than other nonlinear models, making it more computationally efficient. These characteristics make the Volt1D layer a promising choice for utilization in deep-learning algorithms for identifying spoken commands. There are also some limitations to the Volt1D layer that should be considered. One limitation is that it is only able to capture second-order interactions between features in the input data. This means that it may not be able to capture higher-order interactions, which could potentially lead to reduced performance in certain tasks. Another limitation is that the Volt1D layer requires more training data than standard Conv1D layers, as it has more parameters to be optimized. Despite these limitations, the Volt1D layer represents a promising approach for use in deep learning models for speech recognition. Its ability to capture nonlinear relationships in speech data and its simple and efficient structure make it an appealing choice for use in these types of tasks. Further research is needed to explore

the potential of the Volt1D layer in other speech recognition tasks, as well as its ability to capture higher-order interactions in the data. In conclusion, the use of the Volt1D layer in deep learning models for speech recognition is a promising approach for capturing nonlinear relationships in speech data. Results of the experiments show the effectiveness of the Volt1D layer in improving the accuracy of speech recognition tasks, and its simple and efficient structure make it an attractive choice for use in these types of models. Further research is needed to fully understand the potential of the Volt1D layer in speech recognition tasks, as well as its ability to capture higher-order interactions in the data.

5.2. Future Work

In this study we demonstrated the effectiveness of this approach on the speech commands dataset V0.01, where the proposed Volt1D layer achieved an accuracy of 95.6%, significantly outperforming the baseline accuracy of 91.5% obtained using a standard Conv1D layer. However, there are several areas for future work that could be investigated to find out more about further improve the performance of the Volt1D layer. One possibility is to implement higher orders of Volterra series convolution, as previous studies have shown that higher orders can capture more complex nonlinear relationships in data [62]. Another direction is to support multi-dimensional input, such as 2D images or 3D data, by extending the Volt1D layer to Volt2D and Volt3D layers. This would allow the Volt1D layer to be applied to a wider range of applications beyond speech recognition. Additionally, it would be interesting to explore the use of the Volt1D layer in combination with other advanced deep learning techniques. This could potentially further improve the performance of the Volt1D layer, especially when dealing with small or imbalanced datasets. Overall, the proposed Volt1D layer shows promise as a powerful tool for capturing nonlinear relationships in data, and we believe that it has the potential to make significant contributions to the field of deep learning and speech recognition.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436-444, 2015.

[3] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in Proceedings of the 5th International Conference on Learning Representations (ICLR), 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[5] Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems (NIPS), 2012.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems (NIPS), 2015.

[7] S. L. C. Cruz, R. C. Barros, and J. C. Bernardes, "Deep learning applied to breast cancer histopathological image analysis," PloS One, vol. 13, no. 6, p. e0198687, 2018.

[8] Y. Chen, X. Ma, and W. Liu, "Deep learning for speech and language: Overview," arXiv:1709.07854, 2017.

[9] H. Ning, H. Zhang, Y. Chen, and L. Chen, "Disease diagnosis using deep convolutional neural networks," in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2016.

[10] C. Chen, X. Chen, and Y. Zhu, "Structural health monitoring using convolutional neural networks," Structural Control and Health Monitoring, vol. 24, no. 5, p. e2184, 2017.

[11] Y. Wang, J. Yuan, and C. K. Loo, "Deep learning for fault diagnosis: A review," Mechanical Systems and Signal Processing, vol. 96, pp. 1-14, 2017.

[12] Y. Zhang, J. Lu, and D. Li, "Power electronics and electrical engine fault identification using convolutional neural networks," in Proceedings of the IEEE International Conference on Industrial Technology (ICIT), 2017.

[13] K. Jain, "Convolutional neural networks for time series analysis," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2018.

[14] W. Xiong, J. Du, L. Dai, Y. Liu, and J. Dai, "Speech command: A dataset for limited-vocabulary speech recognition," arXiv:1810.03201, 2018.

[15] J. K. Kautsky, "A review of the Volterra and Wiener theories," Automatica, vol. 30, no. 9, pp. 1463-1472, 1994.

[16] D. Kim, D. Kim, and J. Kim, "Deep learning for speech recognition: A review," IEEE Access, vol. 6, pp. 50,418-50,441, 2018.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[18] X. Han, Y. Sun, J. Du, and W. Liu, "Recent advances in deep learning for speech recognition," Frontiers of Information Technology & Electronic Engineering, vol. 19, no. 1, pp. 1-12, 2018.

[19] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[20] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv:1506.00019, 2015.

[21] S. Lee, "Voice recognition technology: A review," Journal of Control, Automation and Electrical Systems, vol. 28, no. 1, pp. 19-29, 2017.

[22] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[23] X. Han, Y. Sun, J. Du, and W. Liu, "Recent advances in deep learning for speech recognition," Frontiers of Information Technology & Electronic Engineering, vol. 19, no. 1, pp. 1-12, 2018.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems (NIPS), 2015.

[25] J. K. Kautsky, "A review of the Volterra and Wiener theories," Automatica, vol. 30, no. 9, pp. 1463-1472, 1994.

[26] Y. Zhang, J. Lu, and Z. Chen, "Speech recognition with deep recurrent neural networks," in Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014.

[27] D. Kim, D. Kim, and J. Kim, "Deep learning for speech recognition: A review," IEEE Access, vol. 6, pp. 50,418-50,441, 2018.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[29] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv:1506.00019, 2015.

[30] TensorFlow, "Speech commands V0.01 dataset," https://www.tensorflow.org/datasets/cata-log/speech_commands, accessed March 2021.

[31] D. J. Dean, "Trends in the development of deep learning for speech recognition," IEEE Access, vol. 6, pp. 50,442-50,456, 2018.

[32] J. Anderson and P. Haton, "The Volterra/Wiener approach to non-linear system identification," IEEE Transactions on Automatic Control, vol. 39, no. 7, pp. 1597-1612, 1994.

[33] J. K. Kautsky, "A review of the Volterra and Wiener theories," Automatica, vol. 30, no. 9, pp. 1463-1472, 1994.

[34] Y. Zhang, J. Lu, and Z. Chen, "Speech recognition with deep recurrent neural networks," in Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014.

[35] D. Kim, D. Kim, and J. Kim, "Deep learning for speech recognition: A review," IEEE Access, vol. 6, pp. 50,418-50,441, 2018.

[36] TensorFlow, "Speech commands V0.01 dataset," https://www.tensorflow.org/datasets/catalog/speech_commands, accessed March 2021.

[37] Speech Commands V0.01 dataset, Google (2017).

[38] J. H. L. Hansen and P. W. J. Peters, "Recent advances in speech recognition," Communications of the ACM, vol. 43, no. 4, pp. 34–38, 2000.

[39] M. Gales, "The challenge of speaker variability in speech recognition," Computer Speech & Language, vol. 20, no. 4, pp. 467–502, 2006.

[40] J. R. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, and K. W. Wilson, "CNN architectures for large-scale audio classification," in Proceedings of the 40th International Conference on Acoustics, Speech, and Signal Processing, 2015.

[41] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in Advances in Neural Information Processing Systems, 2012.

[42] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

[43] Krizhevsky, A., et al. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[44] Graves, A., et al. (2013). Speech recognition with deep recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 6645-6649). IEEE.

[45] Breuel, T. (2015). Deep neural networks for acoustic modeling in speech recognition. Foundations and Trends® in Signal Processing, 9(5-6), 303-384.

[46] Amodei, D., et al. (2016). Deep speech 2: End-to-end speech recognition in English and Mandarin. In Thirty-First AAAI Conference on Artificial Intelligence.

[47] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[48] Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.

[49] Hinton, G., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine, 29(6), 82-97.

[50] Graves, A., et al. (2013). Speech recognition with deep recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 6645-6649). IEEE.

[51] Lee, C.-C., et al. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In Acoustics, Speech and Signal Processing (ICASSP), 2009 IEEE International Conference on (pp. 41-44). IEEE.

[52] Chen, Y., & Billings, S. A. (2000). Volterra series neural networks. IEEE Transactions on Neural Networks, 11(1), 57-72.

[53] Chen, Y. (2001). Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains. John Wiley & Sons.

[54] Chen, Y. (2005). Nonlinear time series models: theory and applications. In Nonlinear time series models in empirical finance (pp. 1-33). Springer, Berlin, Heidelberg.

[55] Chen, Y. (2013). Nonlinear time series analysis: methods and applications. John Wiley & Sons.

[56] K. K. P. B. Dissanayake, R. A. Fernando, and D. I. McLeod, "A review of Volterra series based adaptive filters," Digital Signal Processing, vol. 26, pp. 59-78, 2014.

[57] S. Haykin, "Adaptive filter theory," Prentice Hall, 2002.

[58] S. Kim and H. K. Lee, "Adaptive Volterra filters," IEEE Transactions on Signal Processing, vol. 47, pp. 915-927, 1999.

[59] M. P. Kennedy and L. N. Trefethen, "The Volterra/Wiener paradigm for nonlinear time series," Nature, vol. 365, pp. 613-620, 1993.

[60] Speech Commands: A dataset for limited-vocabulary speech recognition, Google, http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

[61] Abadi, M., et al. (2016). TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (pp. 265-283).

[62] Hinton, G., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine, 29(6), 82-97

[63] Chollet, F. (2015). Keras: The Python deep learning library. In Proceedings of the Python for Scientific Computing Conference (SciPy) (pp. 1-7). Austin, TX: NumFOCUS.

[64] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Vasudevan, V. (2016). TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (pp. 265-283). USENIX Association.

CURRICULUM VITAE

Name Surname : Zakaria Alyafawi

EDUCATION:

- **Graduate** : 2023, Sakarya University, Computer and Information Engineer, Computer Engineering
- **Undergraduate** : 2020, Zarqa University, Information Tecnoligy , Software Engineering

PROFESSIONAL EXPERIENCE AND AWARDS:

- Working as Senior Artificial Intelligence R&D Engineer at Digital Future Company from 2022
- Worked as Database Developer at Eskadenia Software – 2020-2021

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Zakaria A., Devrim A., (2022, 29-30, Decembe). Speech Recognition Using Deep Learning Model With Volterra Series-Based Layer in Tensorflow. *ANADOLU 11 th International Conference on Applied Science*, Diyarbakir, Turkey.