



NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

**SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION**

BSc THESIS

**Neural networks with non-linear convolutions for image
classification**

Andreas D. Giannoutsos

Supervisor: Yannis Panagakis, Associate Professor

ATHENS

MARCH 2022



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Νευρωνικά δίκτυα με μη γραμμικές συνελίξεις για
κατηγοριοποίηση εικόνων**

Ανδρέας Δ. Γιαννούτσος

Επιβλέπων: Ιωάννης Παναγάκης, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2022

BSc THESIS

Neural networks with non-linear convolutions for image classification

Andreas D. Giannoutsos

S.N.: 1115201700021

SUPERVISOR: **Yannis Panagakis**, Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Νευρωνικά δίκτυα με μη γραμμικές συνελίξεις για κατηγοριοποίηση εικόνων

Ανδρέας Δ. Γιαννούτσος

A.M.: 1115201700021

ΕΠΙΒΛΕΠΩΝ: Ιωάννης Παναγάκης, Αναπληρωτής Καθηγητής

ABSTRACT

Convolutional Neural Networks have caused a revolution in the field of computer vision in recent years by continually breaking many state-of-the-art records. CNNs are mathematical models that consist of layers of convolutional operators followed by non-linear activation functions. The non-linear activation functions improve the model's expressive ability, by allowing it to adapt to a wide range of data variations. Another method for increasing the non-linearity of models is the employment of numerous convolutional layers and the formulation of a complex structure between them.

Throughout these years, research has focused on improving these non-linear techniques so that the model can generalize with increasing flexibility on the data. However, there has been a minimal study on improving the nature of the convolution process itself. To tackle this issue, in this bachelor's thesis, we seek to replace the linear convolutional operators with non-linear ones, namely Volterra convolutions.

Volterra convolutions are polynomial approximation functions and are, in fact, the most well-known models for analyzing complex dynamic systems found in nature. As a result, they are deemed appropriate for enhancing the expressive capacity of the linear convolution operator, as well as introducing additional search spaces and dimensions for our estimation functions that are more susceptible to data variations.

In this study, we implement and evaluate the non-linear Volterra convolutions by using the CIFAR10 and CIFAR100 datasets. We demonstrate that they outperform their linear counterparts with just minor changes to our model design. Moreover, we cast light on how the information is interpreted and the higher-order relations that arise in the receptive fields. Also, we identify a resemblance between the non-linear terms of this method and the modern self-attention models that have contributed significantly to the field of computer vision recently. Finally, we show relationships between the non-linear convolutions and the deeper layers of our network, revealing a resemblance to polynomial functions.

The implementations of the non-linear convolutions are provided in this link: <https://github.com/AGiannoutsos/Volterra-Convolution>

SUBJECT AREA: Machine Learning, Computer Vision

KEYWORDS: Volterra Convolutions, Non-Linear Convolutions, Convolutional Neural Networks, Quadratic Convolutions

ΠΕΡΙΛΗΨΗ

Τα συνελκτικικά νευρωνικά δίκτυα έχουν προκαλέσει επανάσταση στον τομέα της υπολογιστικής όρασης τα τελευταία χρόνια σπάζοντας συνεχώς πολλά ρεκόρ της τελευταίας λέξης της τεχνολογίας. Τα CNN είναι μαθηματικά μοντέλα που αποτελούνται από στρώματα συνελκτικών τελεστών, ακολουθούμενα από μη γραμμικές συναρτήσεις ενεργοποίησης. Οι μη γραμμικές λειτουργίες ενεργοποίησης βελτιώνουν την εκφραστική ικανότητα του μοντέλου, επιτρέποντάς του να προσαρμόζεται σε ένα ευρύ φάσμα διασποράς δεδομένων. Μια άλλη μέθοδος για την αύξηση της μη γραμμικότητας των μοντέλων είναι η χρήση πολυάριθμων συνελκτικών στρωμάτων και η διαμόρφωση μιας πολύπλοκης δομής μεταξύ τους.

Κατά τη διάρκεια των τελευταίων ετών, η έρευνα έχει επικεντρωθεί στη βελτίωση αυτών των μη γραμμικών τεχνικών, έτσι ώστε το μοντέλο να μπορεί να γενικεύει με αυξανόμενη ευελιξία στα δεδομένα. Ωστόσο, ελάχιστη μελέτη έχει πραγματοποιηθεί για τη βελτίωση της φύσης της ίδιας της διαδικασίας συνέλιξης. Για να αντιμετωπίσουμε αυτό το ζήτημα, σε αυτή τη πτυχιακή διατριβή, επιδιώκουμε να αντικαταστήσουμε τους γραμμικούς συνελκτικούς τελεστές με μη γραμμικούς, τις ονομαζόμενες συνελίξεις Volterra.

Οι συνελίξεις Volterra είναι συναρτήσεις πολυωνυμικής προσέγγισης και καθιστούν ένα από τα πιο γνωστά μοντέλα για την ανάλυση πολύπλοκων δυναμικών συστημάτων που βρίσκονται στη φύση. Ως αποτέλεσμα, κρίνονται κατάλληλες για την ενίσχυση της εκφραστικής ικανότητας του τελεστή γραμμικής συνέλιξης, καθώς και για την εισαγωγή πρόσθετων χώρων αναζήτησης και διαστάσεων για τις συναρτήσεις εκτίμησης που είναι πιο επιρρεπείς σε παραλλαγές δεδομένων.

Σε αυτή τη μελέτη, υλοποιούμε και αξιολογούμε τις μη γραμμικές συνελίξεις Volterra χρησιμοποιώντας τα σύνολα δεδομένων CIFAR10 και CIFAR100. Αποδεικνύουμε ότι υπερτερούν των γραμμικών ομολόγων τους με μικρές μόνο αλλαγές στη σχεδίαση του μοντέλου μας. Επιπλέον, ρίχνουμε φως στον τρόπο ερμηνείας των πληροφοριών και στις σχέσεις υψηλότερου επιπέδου που προκύπτουν στα δεκτικά πεδία. Επίσης, εντοπίζουμε μια ομοιότητα μεταξύ των μη γραμμικών όρων αυτής της μεθόδου και των σύγχρονων μοντέλων αυτοπροσοχής που έχουν συνεισφέρει σημαντικές ανακαλύψεις στον τομέα της υπολογιστικής όρασης πρόσφατα. Τέλος, δείχνουμε σχέσεις μεταξύ των μη γραμμικών συνελίξεων και των βαθύτερων στρωμάτων του δικτύου μας, αποκαλύπτοντας μια ομοιότητα με πολυωνυμικές συναρτήσεις.

Οι υλοποιήσεις των μη γραμμικών συνελίξεων παρέχονται στον παρακάτω σύνδεσμο: <https://github.com/AGiannoutsos/Volterra-Convolution>

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση, Μηχανική Όραση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Συνελίξεις Βολτέρρα, Μη-γραμμικές Συνελίξεις, Συνελκτικά Νευρωνικά Δίκτυα, Τετραγωνικές Συνελίξεις

ACKNOWLEDGMENTS

For my thesis, I would like to thank my supervisor, professor Yannis Panagakis, for his instructions, advice, and the research thinking approach that he introduced to me. I'd also want to thank my parents for their assistance during my studies and for their support in order for me to fulfill my objectives.

CONTENTS

1	INTRODUCTION	13
1.1	Related Work	14
1.1.1	CNNs	14
1.1.2	Self-attention and transformers in vision	15
1.1.3	Non-linear and polynomial convolutions	15
2	METHOD	17
2.1	Linear Convolution	17
2.1.1	Convolutional Neural Networks	17
2.2	Volterra Series	18
2.3	Volterra Convolution	20
2.3.1	Volterra convolution and matrix notation	22
2.3.2	Volterra convolution and Self-attention mechanism	22
2.4	Volterra-Based Model	24
2.4.1	Volterra Convolution Block	24
2.4.2	Model Architecture	26
3	EXPERIMENTS	29
3.1	Experiments Setup	29
3.1.1	Model Hyper-parameters	29
3.1.2	Training Hyper-parameters	30
3.1.3	Training Process	32
3.2	Datasets	32
3.3	Results and Discussions	32
3.3.1	Grid Search	36
3.3.2	Linear and Volterra Activations	39
3.3.3	Linear and Volterra Parameter Distributions	44
4	CONCLUSION	47

LIST OF FIGURES

1	This figure describes in detail all the steps by which we can calculate the 2nd order Volterra convolution. The steps are consisted of reshaping and inner product operations, that update the tensors $A, B, C, D, E, F, G, H, I$	24
2	The ResNet blocks are depicted in this diagram. The basic block is given in the first column by the sequence of convolutional, batch normalization, and ReLU layers. The usage of 1×1 convolutions on residual connections is presented in the second column.	27
3	In these pictures the kernels are represented at different values of dilation. (a) represents the kernel of dilation 1; (b) depicts the kernel of dilation 2; (c) depicts the kernel of dilation 3.	30
4	These plots show the Validation Cross Entropy Loss of our three top performing models on the CIFAR10 dataset per epoch of our training process; (a) represents the total loss from the beginning of the training to the end; (b) depicts a portion of the training process between 100 and 150 epochs in which a decrease in learning rate is observed. This diagram's values are also normalized and averaged using exponential moving averaging.	33
5	These charts depict the Validation Cross Entropy Loss of our three best performing models on the CIFAR100 dataset each training epoch; (a) indicates the overall loss from the start of the training to the finish. A section of the training process between 100 and 150 epochs is illustrated in (b), where a reduction in learning rate is visible. The numbers in this chart are also adjusted and averaged using exponential moving averaging.	34
6	The Validation Cross Entropy Loss of all of our best performing models on both the CIFAR10 and CIFAR100 datasets is shown in these figures; (a) displays the total loss curve from the beginning to the completion of the training; (b) represents a section of the training process between 100 and 150 epochs, during which a reduction in learning rate is evident, along with the loss. The data in this diagram are also scaled and averaged using exponential moving averaging.	35
7	Grid search correlations.	38
8	The probability density of the weights in three distinct convolution layers of the Linear model is shown in these histograms.	44
9	In these histograms, the probability density of the weights in 3 different convolution layers of the Volterra 2nd order model are presente.	45
10	The probability density of the weights in three distinct convolution layers of the Volterra 3rd order model is shown in these histograms.	46

LIST OF TABLES

1	This table covers the model’s design in depth. The model’s phases are listed in the first column. The size of the picture as it outputs at each stage is indicated in the second column. The third column goes into great depth on the parameters of each block. For the Volterra convolution stage, we have two options. One is to use Linear convolutions in the first layer of the model and the other is to replace them with Volterra convolutions on the first layer of the model.	28
2	This table explains the settings of the learning rate, weight decay, optimizer, and momentum per epoch period.	32
3	This table lists the accuracy and mean accuracy on CIFAR10 for each model evaluated between Linear, Volterra 2nd, and Volterra 3rd order, together with the number of parameters for each model in millions. Volterra 2nd order had the greatest single and mean accuracy, indicating that Volterra convolutions enhance the model's performance. . . .	33
4	This table shows the accuracy and mean accuracy on CIFAR100 for each model evaluated between Linear, Volterra 2nd, and Volterra 3rd order together with the number of parameters for each model in millions. The Linear model obtained the highest single and mean accuracy.	34
5	This table shows the grid search parameter values used to identify improved channel, kernel size, and dilation parameters in Volterra convolutions.	36
6	This table describes the parameter values for grid search in order to find better parameters of scaling and masking in Volterra convolutions. . . .	36
7	The optimal parameter settings for the Volterra convolution are described in this table, for the 12 best performing models.	37
8	This table describes the optimal scaling and masking settings, resulting in the best 4 models of our experimentation.	37
9	This table depicts the early initial activations of our Volterra 2nd order network for distinct CIFAR10 classes at the commencement of our training procedure. The first column displays the original input image; the second column demonstrates the output of the linear convolution layer; the third column exhibits the activations of the 2nd order Volterra layer; and the fourth, fifth, and sixth columns display the activations of the group1, group2, and group3 convolutional blocks, respectively. . . .	40
10	This table illustrates the later intermediary activations of our Volterra 2nd order net for different CIFAR10 classes at the completion of our training phase. The first column displays the original input image; the second column shows the output of the linear convolution layer; the third column showcases the activations of the 2nd order Volterra layer; and the fourth, fifth, and sixth columns demonstrate the activations of the group1, group2, and group3 convolutional blocks, accordingly. . . .	41

- 11 This table illustrates the **early** intermediate activations of our Volterra 3rd order network at the beginning of our training process for various CIFAR10 pictures. The original input image is shown in the first column; the output of the linear convolution layer can be seen in the second column; the activations of the 2nd order Volterra layer are included in the third column; the activations of the 3rd order Volterra layer are shown in the fourth column; and the activations of the group1, group2, and group3 convolutional blocks are depicted in the fourth, fifth, and sixth columns correspondingly. 42
- 12 This table illustrates the **late** intermediate activations of our Volterra 3rd order network at the end of our training process for various CIFAR10 classes. The original input image is shown in the first column; the output of the linear convolution layer can be seen in the second column; the activations of the 2nd order Volterra layer are included in the third column; the activations of the 3rd order Volterra layer are shown in the fourth column; and the activations of the group1, group2, and group3 convolutional blocks are displayed in the fourth, fifth, and sixth columns accordingly. 43

PREFACE

This thesis was completed as part of the Bachelor's degree program at the National and Kapodistrian University of Athens' Department of Informatics and Telecommunications from October 2021 to March 2022.

1 INTRODUCTION

Convolutional neural networks, CNNs, have been proved to produce cutting-edge results on a variety of computer vision applications, including image classification. Their design was heavily influenced by biology and the models of primate visual systems, such as the one given by Hubel and Wiesel [19]. Convolution is an essential component because it allows the model to learn invariant representations.

Convolution is a linear process. As a result, since linear convolutions do not have enough expressive capacity to explain the world, there have been developed approaches that modify its non-linear property. The introduction of activation functions such as the ReLU function [29] is an excellent example. Also, the depth of the models with several deep layers stacked on each other offers the model more expressive capability. Although much research has been conducted to improve the architecture of the CNN model, little has been done to incorporate non-linearity [50] in convolutions themselves.

For these reasons in this work we introduce non-linearity in our models by using the Volterra Series [43] which form the Volterra convolutions. Volterra Series are well-known systems that have been used to simulate the complex processes of nature from frequency to time and space domains. The non-linearity is achieved by adding to the convolutional operator, beyond the linear elements, the multiplication of all the input element interactions from a small patch of the image. Non-linear convolutions can provide a richer representation of the data by utilizing the higher order relations between the input data, thus improving the selectivity in the translation invariance property of the convolution.

Volterra convolutions and their quadratic and cubic forms, multiply the input data by many times in order to capture meaningful information that belies within the interactions of the data. Having numbers that better correlate with each other tends to increase the similarity factor, thus the signal in that part of the model is magnified and contributes more to the final resolution of the model. From the above features we can also assume that exhibit similar characteristics with self-attention models, as they follow an input adaptive weighting and the form correlation between pairs in the input data.

Furthermore, the stacking of many convolutional layers in order to increase the non-linear aspects of a models, is similar to the mathematical formulation of a deep multi-linear polynomial network [7]. With the use of non-linear convolutions, which resemble the synthesis of polynomials, we can verify the theory of the unification of the neural networks under a mathematical framework of multi-linear polynomial networks.

Finally, in our efforts to increase image recognition performance, we obtain three unambiguous results: we present an alternative convolutional operator as well as a self-attention approach and we verify the occurrence of polynomials in deep CNNs. For that reason our work can be interpreted as a contribution to the advancement of

the convolutional operator, while being also a variant of a self-attention method and a verification of polynomial neural networks.

1.1 Related Work

1.1.1 CNNs

Convolutional Neural Networks are not a novel concept. In order to extract features from an image Gabor filters were implemented at first [28]. Since these methods lacked consistency, Fukushima established one of the initial versions of CNNs in 1988 [13]. Later in that decade, the back propagation mechanism for automatic optimization of model parameters was applied to the CNN models in [27]. It is in [26] where gradient descent based back propagation was utilized which was meant to offer countless possibilities in the way parameters are adapted to the data variations. These models, while expressive enough, were not adequately employed due to a lack of computational resources. Deep and large models, as well as deep learning, were to usher in a revolution in computer vision at the beginnings of the previous decade.

The introduction of deep CNNs with the AlexNet model [24], which set a new record in the ImageNet benchmark [35] with a vast improvement started a new era in computer vision. Following that advancement the field has developed with an unprecedented speed. Many networks later focused on various parts of the model's structure. VGGNet [36] concentrated on the depth and frequency of convolutions, whereas Inception Network [40] focused on the representational capability of deep convolutions with multiple interchangeable filters and channels. ResNet [16] pioneered the concept of skip connections, which convey data and improve the training process. Efforts have also been made in terms of efficiency, with MobileNet [17] investigating separable convolutions and their capacity to reduce model parameters, and EffientNet [41] introducing a new class of sophisticatedly scaled convolutional models. Convolutions have been utilized for many additional computer vision problems outside image classification, such as object detection [33] and semantic segmentation [14], [34] Convolution's great translation equivariance and invariance properties made this operator appealing for usage in a wide range of image-related applications. Furthermore, the weigh sharing feature meant that the models had a reduced number of parameters while still being generalizable. Convolutions were the primary building block of computer vision machine learning models across all of these breakthroughs. Convolutions, on the other hand, are incapable of capturing long-range interactions due to their restricted receptive fields. As a result, several approaches from sequence models, such as the self-attention mechanism, have been introduced to mitigate this issue. Additionally, a high association between these models and non-linear convolutions can be found.

1.1.2 Self-attention and transformers in vision

As we expect a resemblance to non-linear convolutions and self-attentions it is worth noting similar methods that have been applied to computer vision problems. A plethora of such models exist like [18] where the local relation layer dynamically calculates weight aggregation based on the relation of the local pixel pairs with linear layers and imputed structural information. In addition to that [32] makes use solely on self-attention modules on convolutional kernels instead of the plain convolutions, while achieving an decrease in computational cost. Following that approach, convolutional operators can be completely replaced by attention layers.

There are also approaches that self-attention layers serve as augmentations on the convolutional blocks in which they try to benefit from both of their properties like the convolutions' translation equivariance and the self-attentions' input-adaptive weighting and global receptive field. Such a method is incorporated by [9] in which they add the convolutional weights to the self-attention weights, thus gaining an advantage of the both methods at the same time. Another great example could be [2] where authors support that the concatenation of features attained from attention heads and convolutional filters yields the best performance. Moreover [37] uses attention layers as a bottleneck layer between convolutions.

Apart from the self-attention heads, in the recent years, the introduction of transformers [42] has caused a revolution in NLP world with language models like BERT [10]. Despite the gap in task of interest between computer vision and NLP, the two processes converged once Vision Transformers [11] were introduced. ViT treats images as a sequence of strings, therefore no image-specific logical bias is introduced. One advantage of this is that ViT is an excellent model for scaling in huge datasets. However, because to a lack of worldwide understanding of image detection difficulties, ViT face implementation hurdles as key building blocks of computer vision. The Hierarchical Transformers [12] have been one approach to resolving this issue, albeit they are seen as a temporary solution. As a result, CNNs are an essential component of computer vision, while non-linear convolutions may incorporate the characteristics of both convolutions and self-attentions.

1.1.3 Non-linear and polynomial convolutions

Aside from study on the design of convolutional networks, there have also been attempts to clarify the domains of non-linear convolutions and their introduction as polynomials inside the neural network. To begin with, [50] have already introduced the concept of non-linear Volterra convolutions. Our research on non-linear convolutions is based mostly on [50], as they had already implemented an version of Volterra convolutions and tested it on the CIFAR10 and CIFAR100 datasets. Although, this work lacks the experimentation of the 3rd order Volterra convolutions and an insight about the parameters of the model that contribute the most to the non-linear convolutions.

In addition to that research, non-linear convolutions have been also treated as a form of polynomials. In fact, the addition of a linear, a square and a cubic term multiplied by coefficients - in our case filters - is the concept of a 3rd order polynomial expression.

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n, n \geq 0$$

By seizing that representational opportunity, [44] proposed the use of polynomial equation in the convolutional operator, thus the use of non-linear convolutions. They have proved this theory on GNNs with 2nd order polynomials on the GCNN model, a variant of convolutional models on non-euclidean space.

Parallel to neural network and CNN research, efforts have been made to integrate neural network mathematical operations into a sequence of polynomial functions. Deep CNNs in [7] are represented by high-order multi-linear polynomials, while high-order polynomials in [6] have been introduced in GANs.

It was important that we commented on the developments in the research Polynomial networks because since as our research is largely unrelated to it, in the findings of the experiments there are presented data that verify and reinforce the idea that neural networks can be replaced by polynomial approximation functions.

2 METHOD

In this section, we will go through the fundamentals of Volterra-based convolutions. We will begin with an overview of linear convolutions and then go on to non-linear Volterra-based convolutions. In addition to the explanation of the non-linear convolutions, we will offer a short insight about the similarity with the self-attention models. After we've established a mathematical foundation, we'll move on to an explanation of how they're implemented in PyTorch. Finally, we will present not just the model in which we will use Volterra-based convolutions, but also its architecture.

2.1 Linear Convolution

Convolution is a mathematical operation that is executed on two functions to produce a third function that reveals how the shape of one is modified by the other. Convolution refers to both the outcome function and the method used to compute it. It is defined as the integral of the product of two reversed and shifted functions. Let f and g be these two functions and $*$ be the operator. Then the convolution will be calculated as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)d\tau \quad (1)$$

It can be viewed as a method of multiplying two arrays of numbers, often of different sizes but the same dimensions, to generate a third array of numbers of the same dimensionality. This can be expanded in images as convolutions may be used to build operators whose output pixel values are simple linear combinations of particular input pixel values. In other words it is the process of adding each feature of the image to its nearby neighbors.

2.1.1 Convolutional Neural Networks

Deep Learning has powerful applications in many practical fields of science and technology. Deep Convolutional Neural Networks (deep CNNs) are an important family of artificial deep neural networks. MLPs have been regularized to create CNNs. Fully connected networks, or MLPs, are networks in which each neuron in one layer is linked to all neurons in the following layer. Convolutional Neural Networks are a form of neural network that uses convolution rather than standard matrix multiplication in at least one layer. Theoretical [49] as well as empirical evidence [20] suggests that convolutions help deep CNNs to efficiently learn locally shift-invariant features, allowing them to demonstrate their capabilities in texts, images, and several other types of data.

CNNs consist of multiple 2D convolutions over the images where the kernels of the filters are the trainable parameters. The expression of the 2D convolution is:

$$g(x, y) = conv(f(x, y)) \Rightarrow$$

$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b \omega(dx, dy) f(x + dx, y + dy)$$

where $g(x, y)$ is the product of the convolution and ω is the kernel.

The CNNs have multiple layers and sizes of kernels ω^L . In addition every convolution output is passed through a non-linear function. More preciously at the l^{th} layer of the CNN we have the following:

$$conv(x^{[l-1]}, \Omega) = \sigma^{[l]} \left(\sum_{i=1}^{n_C^{[l-1]}} \left(\sum_{j=1}^{n_H^{[l-1]}} \sum_{k=1}^{n_W^{[l-1]}} \Omega_{ijk} x_{i,h+j-1,w+k-1}^{[l-1]} + b^{[l]} \right) \right) \quad (2)$$

Ω are the kernels, x is the input of the layer, σ^l is the non-linear function or activation function at the l^{th} layer and $b_n^{[l]}$ are the biases at that layer. After several convolutional layers, the CNN model produces a predicted picture. The mean error between the predicted and target images is then calculated using the loss function, and the loss is propagated to the parameters using the back-propagation process.

These models are effective because they examine linear correlations between the picture and its features. Correlations in nature, on the other hand, are extremely complicated, as they nearly never form a linear function between physical quantities. That is why we strive to simulate physical processes using linear equations. Nonetheless, there are methods in convolutions that allow us to harness nature's non-linear components and extract their properties. These techniques can be found in the Volterra Series.

2.2 Volterra Series

Volterra series are a type of nonlinear polynomial representation. They are possibly the most well-known and commonly utilized non-linear system representations in signal processing. A Volterra representation is a mathematical extension of the conventional linear system representation.

A system may be described more precisely as a rule that assigns a value y to an input x . This rule can also be expressed as in 3 using the T operator.

$$y(t) = Tx(t) \quad (3)$$

In classical system theory the systems assumed continuous and time-invariant. So and we limit the T operator to the system response that can be expressed by signal convolution as can be described in 4 by using the H operator.

$$y(t) = H_1x(t) = \int_{\mathbb{R}} h^{(1)}(\tau)x(t - \tau)d\tau \quad (4)$$

Volterra [43] expanded this formula into non linear representations by adding a series of non linear terms like:

$$\begin{aligned} y(t) = & h^0 + \int_{\mathbb{R}} h^{(1)}(\tau_1)x(t - \tau_1)d\tau_1 \\ & + \int_{\mathbb{R}^{\neq}} h^{(2)}(\tau_1, \tau_2)x(t - \tau_1)x(t - \tau_2)d\tau_1d\tau_2 \\ & + \int_{\mathbb{R}^{\neq}} h^{(3)}(\tau_1, \tau_2, \tau_3)x(t - \tau_1)x(t - \tau_2)x(t - \tau_3)d\tau_1d\tau_2d\tau_3 \\ & + \dots \end{aligned}$$

The Volterra Series then can be written as:

$$y(t) = H_0x(t) + H_1x(t) + H_2x(t) + \dots + H_nx(t) \quad (5)$$

Where every term H_n is a non linear operator that filters the signal.

$$H_nx(t) = \int_{\mathbb{R}^n} h^n(\tau_1, \dots, \tau_n)x(t - \tau_1)(t - \tau_n)d\tau_1 \dots d\tau_n \quad (6)$$

The term H_0 is a constant value and later this will be our bias for the representation. In equation 6 every $h^{(n)}$ in the integral is a kernel which is called Volterra Kernel. This must be causal because the features of the signal can not be calculated from the future. For that reason every Volterra Kernel must maintain the following properties:

$$h^n(\tau_1 \dots \tau_n) = 0 \text{ for any } \tau_i < 0 \text{ where } i = 1, 2, 3, \dots, n \quad (7)$$

The Volterra series may be viewed of as a Taylor series with memory: while the conventional Taylor series only reflect systems that transfer the inputs to outputs instantly, the Volterra series describes systems in which the result is additionally influenced by input information. Regardless of the type of problem, integrals can be computed in finite and non-finite intervals. Nevertheless, for computer applications we have to use finite completion intervals. The discrete data can enter the model in the form of matrices and tensors of many dimensions. In this way Volterra Kernels can parse the data with the sliding window technique. The discretized Volterra operator [1] is defined as:

$$y(t) = h_0 + \sum_{n=1}^N \sum_{\tau_1=a}^k \cdots \sum_{\tau_n=a}^k h_n(\tau_1, \dots, \tau_n) \prod_{i=1}^n x(t - \tau_i) \quad (8)$$

Where $h_n(\tau_1, \dots, \tau_n)$ are the discrete Volterra Kernels in the form of matrices or multi-dimensional tensors. and since the operation needs to be causal, the kernels may form an upper triangular matrix or a super-diagonal tensor. In addition, the symmetrical kernels, can be formulated as in 9 in order to avoid the unnecessary computations of the triangular form. Although later in the implementation, due to software architectural decisions, kernels are fully computed and a triangular mask is applied for the causality.

$$y(t) = h_0 + \sum_{n=1}^N \sum_{\tau_1=0}^k \sum_{\tau_2=\tau_1}^k \sum_{\tau_3=\tau_2}^k \cdots \sum_{\tau_n=\tau_{n-1}}^k h_n(\tau_1, \dots, \tau_n) \prod_{i=1}^n x(t - \tau_i) \quad (9)$$

This discrete formula can be used for the practical problems of signal processing. According to the Stone–Weierstrass theorem any continuous nonlinear system can be approached by a discrete finite system where in our case we have the Volterra Series.

The convergence of an infinite Volterra series cannot be guaranteed for any input signals due to its power series nature, which have polynomial complexity. As a result, both the input and output signals must be limited to a certain degree. In our approach we will experiment with up to 3rd order non-linear degrees.

2.3 Volterra Convolution

The idea of non-linear convolutions can be extended to the 2D signals of the images and as a consequence to the Deep Convolutional Neural Networks. But first of all, we have to explain the technique with which non-linear convolutions can be applied with the use of Volterra Series.

The Volterra series is a set of approximations that attempts to simulate dynamic systems in the real world. Similarly, Volterra-based convolutions employ appropriate kernels to filter the input data. Volterra kernels perform the same functions as linear convolution kernels. The linear kernels are the first order kernels, and they are identical to conventional convolutions. Then the 2nd order kernel then takes into account the interactions between the input data two times. These interactions are then filtered by the 2nd degree kernel. This algorithm has a polynomial complexity since the input data is multiplied by itself at each degree to capture higher-order interactions. For our method we will incorporate 2nd and 3rd order Volterra convolutions. If we consider L as a set of elements that the convolution kernel ω of shape (k_h, k_w) computes on its

pass the these elements reshaped would form an array of size $k_h \cdot k_w$. Per patch of the sliding window these data would form an array x .

$$x = [x_1, x_2, \dots, x_n] \text{ where } n = k_h \cdot k_w \quad (10)$$

Then a simple calculation the 1st order of the Volterra convolution would result in the following expression:

$$y(x) = b + \sum_{i=1}^n \omega_i x_i \quad (11)$$

Where ω_i are the parameters of the 1st order Volterra kernel and x_i is the data in their receptive field. When we apply this theory to higher-order Volterra convolutions, we get the following forms:

$$y(x) = b + \sum_{i=1}^n \omega_i^{(1)} x_i + \sum_{i=1}^n \sum_{j=1}^n \omega_{ij}^{(2)} x_i x_j \quad (12)$$

$$y(x) = b + \sum_{i=1}^n \omega_i^{(1)} x_i + \sum_{i=1}^n \sum_{j=1}^n \omega_{ij}^{(2)} x_i x_j + \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \omega_{ijk}^{(3)} x_i x_j x_k \quad (13)$$

Equation 12 is responsible for the calculation of 2nd order Volterra convolutions while equation 13 is for the 3rd order Volterra convolution. Regarding the kernels, $\omega_{ij}^{(2)}$ consist the 2nd order kernel and it forms an upper triangular matrix because of the causality concerns, while $\omega_{ijk}^{(3)}$ is the 3rd order kernel and can be reshaped in a symmetrical 3rd order tensor of size n^3 .

Finally, if we take combine equation 2 and the equations 12 and 13 from Volterra convolutions, we can get the following form that details integration of Volterra convolutions in a CNN.

$$volterra_conv(x^{[l-1]}, \Omega) = \sigma^{[l]} \left(\sum_{c=1}^{n_C^{[l-1]}} \left(b + \sum_{i=1}^n \omega_{ci}^{(1)} x_i^{[l-1]} + \sum_{i=1}^n \sum_{j=1}^n \omega_{cij}^{(2)} x_i^{[l-1]} x_j^{[l-1]} \right) \right) \quad (14)$$

$$volterra_conv(x^{[l-1]}, \Omega) =$$

$$\sigma^{[l]} \left(\sum_{c=1}^{n_C^{[l-1]}} \left(b + \sum_{i=1}^n \omega_{ci}^{(1)} x_i^{[l-1]} + \sum_{i=1}^n \sum_{j=1}^n \omega_{cij}^{(2)} x_i^{[l-1]} x_j^{[l-1]} + \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \omega_{cijk}^{(3)} x_i^{[l-1]} x_j^{[l-1]} x_k^{[l-1]} \right) \right) \quad (15)$$

2.3.1 Volterra convolution and matrix notation

In this section we will explain how equations 12 and 13 can be written more efficiently by utilizing the matrix notation and products. But first we need to briefly define them.

As explained in [22] the *Kronecker product* of matrices $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{K \times L}$ is denoted as $A \otimes B$ and the result matrix will have a size of $\mathbb{R}^{IK \times JL}$.

The product can be defined as:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1J}B \\ a_{21}B & a_{22}B & \dots & a_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}B & a_{I2}B & \dots & a_{IJ}B \end{bmatrix} \quad (16)$$

The *Khatri – Rao product* [21] is the columnwise Kronecker product. If we have matrices $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{K \times L}$ then their Khatri-Rao product is denoted as $A \odot B$ and the result matrix will be a matrix of size $\mathbb{R}^{IJ \times K}$.

The product can be defined as:

$$A \odot B = [a_1 \otimes b_1 \quad a_2 \otimes b_2 \quad a_1 \otimes b_1 \quad \dots \quad a_K \otimes b_K] \quad (17)$$

Given the already mentioned notations and products the Volterra convolution operator of equations 12 and 13 can be rewritten more compactly as:

$$y(x) = b + \omega_1^T x + \omega_2^T (x \odot x) \quad (18)$$

$$y(x) = b + \omega_1^T x + \omega_2^T (x \odot x) + \omega_3^T (x \odot x \odot x) \quad (19)$$

Where the coefficients ω_1 , ω_2 and ω_3 have dimensions of $(n^1, 1)$, $(n^2, 1)$ and $(n^3, 1)$ correspondingly.

2.3.2 Volterra convolution and Self-attention mechanism

The Attention mechanism may be thought of as a mapping operation between three vectors: K, Q, and V. These three must have originated from the linear transformation of the same vector X at first. The vector V multiplied by the vector of similarities between K and Q will be the final outcome of the attention operation. The self-attention mechanism removes the last multiplication and keeps the compatibility between K and Q vectors. Consequently, either one is a special case of the other, although the consideration of attention instead of the self-attention does not affect our assumptions. To be more specific, self-attention is a sub-process of the attention procedure, yet it is

the most integral piece since it captures the interaction between the input data. This information is subsequently sent to the other attention mathematical processes, while maintaining the core practice. As a result, self-attention elements may be recognized using an attention model.

For our example will invoke to prove the similarity between Volterra convolution and attention with the most common and popular type of self-attention the Scaled Dot Product [42]. In 20 $\sqrt{d_k}$ is a scaling factor, while Q, K and V are our given vectors.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (20)$$

The similarities between the two techniques can be traced to the multiplication of the K and Q terms. These two terms resulted from a linear transformation of the image's input data. As a result, removing this step makes this approach very similar to the partial multiplication of the kernel patches of the picture to which the Volterra convolution is applied.

Indisputably, there are several foundational differences. Initially, such a resemblance may be considered for a Volterra convolution of the second order. The similarity with 3rd order convolution and multiplication V would be rather unstable because it is done after the application of the softmax function, which is the other notable difference between the two approaches. In the absence of softmax, the product's values move arbitrarily within their domain. Finally, the Volterra convolution is applied to patches of the image and shares the same weights, but the self-attention has different weights for every input values that are equally derived from the interactions of all the input pixels.

2.4 Volterra-Based Model

In this section, we will detail how we implemented the Volterra convolutional block and the model in which it is embedded.

2.4.1 Volterra Convolution Block

In this subsection, we will go through the approaches and methods utilized in PyTorch [31] to calculate the highest order interactions in a Volterra convolutional block in greater depth.

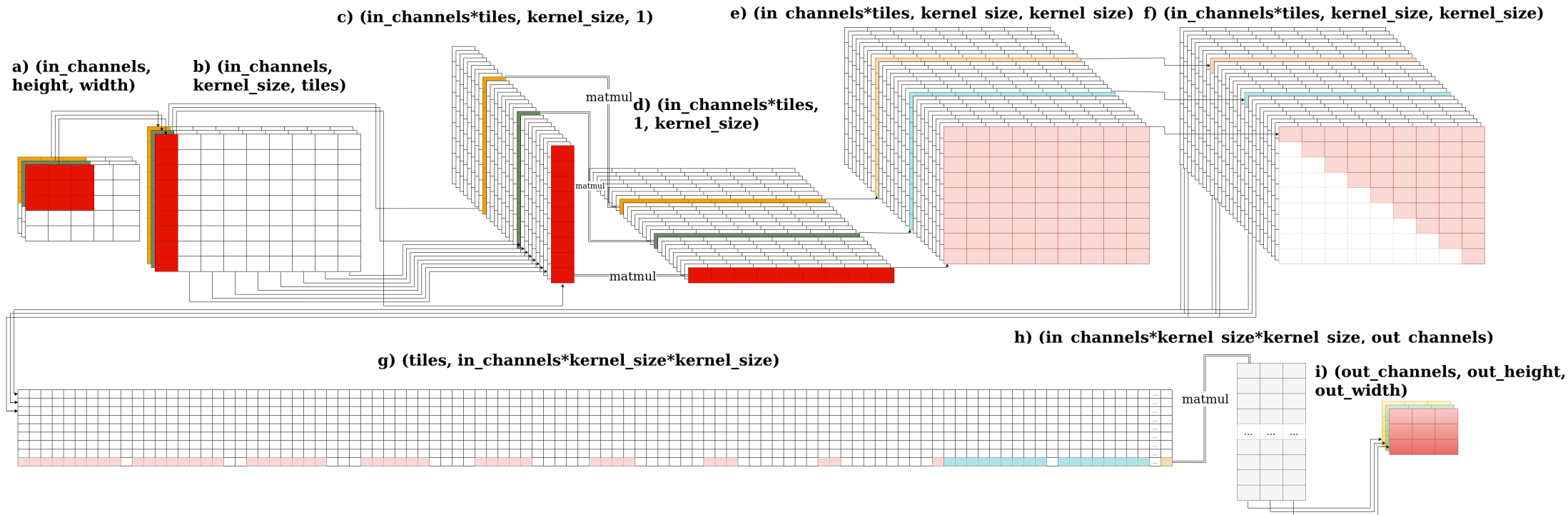


Figure 1: This figure describes in detail all the steps by which we can calculate the 2nd order Volterra convolution. The steps are consisted of reshaping and inner product operations, that update the tensors $A, B, C, D, E, F, G, H, I$

More precisely in Figure 1 tensor $\mathcal{A} \in \mathbb{R}^{in_channels,height,width}$ is our input image for the 2nd order Volterra convolution block. This image is then reshaped, with the PyTorch function $Unfold()$ which incorporates the $im2col()$ algorithm, into columns of the patches that were designated for the convolution operation by considering the padding, dilation and strides. In the unfolded tensor $\mathcal{B} \in \mathbb{R}^{in_channels,kernel_size,tiles}$, the $kernel_size$ originates from the $height$ and $width$ multiplication and the $tiles$ are the total convolution patches per channel of the image. After that the tensor is again reshaped in the shape of $(in_channels \cdot tiles, kernel_size, 1)$ so that the broadcast operations of PyTorch's $Matmul()$ function favors the desired multiplication of our vectors. Therefore, we permute the dimensions of tensor \mathcal{C} into a new tensor $\mathcal{D} \in \mathbb{R}^{in_channels \cdot tiles, 1, kernel_size}$. After that we calculate the inner product of tensor \mathcal{C} and \mathcal{D} as follows.

Let tensors \mathcal{C} , \mathcal{D} and a new tensor $\mathcal{E} \in \mathbb{R}^{in_channels \cdot tiles, kernel_size, kernel_size}$. Tensor \mathcal{E} will be filed like:

$$\mathcal{E}_{cij} = \prod C_{ci1} D_{c1j} \text{ for every } i, j \in (1, kernel_size) \quad (21)$$

Following that operation we end p with a tensor $\mathcal{E} \in \mathbb{R}^{in_channels \cdot tiles, kernel_size, kernel_size}$ that stores all the interaction between the data of the kernels for every channel. However, since we know from form 7 that the kernels must be causal, in tensor \mathcal{E} are stored data interactions from the same elements in different positions. Furthermore, in our previous from of the inner product 21 there is not any guarantee that i, j will overlap between them. This is due to the fact that PyTorch does not provide an already build and optimized function that computes the inner product in a given boundary of dimensions. For that reason and to avoid the use of for loops inside our block, we apply an upper triangular mask on tensor \mathcal{E} that cancels that interactions between the already computed elements, thus our operation maintains causal, as Volterra series need to be.

Additionally to the zero mask we further divide our inner product results by a scaling factor d_k . For other attention like operations this step is avoided like in [3]. However, we predict that for large values, the magnitude of the dot products increases, forcing the Volterra activations into locations with exceptionally small gradients. To compensate for this issue, just like authors do in [42], we multiply the dot products by $\frac{1}{\sqrt{d_k}}$ where d_k is the size of the dot product dimension. To illustrate this phenomenon, lets assume k and p as two independent random variables with mean 0 and variance 1. Then their dot product would result in the following mean and variance:

$$matmul(k, p) = \sum_{i=1}^{d_k} k_i q_i i, \text{ with mean 0 and variance } d_k \quad (22)$$

After the application of our zero causal mask and our scaling factor on tensor \mathcal{E} , \mathcal{F} is reshaped into a tensor $\mathcal{G} \in \mathbb{R}^{tiles, in_channels \cdot kernel_size \cdot kernel_size}$. We follow that process

in order to accelerate the inner product between our input features and interaction with our learning parameters $\mathcal{H} \in \mathbb{R}^{in_channels \cdot kernel_size \cdot kernel_size, out_channels}$. Finally, the input image is multiplied with the learning parameters and then the end product is reshaped with the use of the PyTorch function $Fold()$ and the final convoluted image $\mathcal{I} \in \mathbb{R}^{out_channels, out_height, out_width}$ is filled.

The process for the 3rd order Volterra convolution is mostly the same, whereas at the step of the inner product between tensors \mathcal{C} and \mathcal{D} , another inner product between their output is calculated to capture 3rd order interactions. The 3rd order tensor $\mathcal{Q} \in \mathbb{R}^{in_channels \cdot tiles, kernel_size, kernel_size, kernel_size}$ would have polynomially more terms than the 2nd order convolution, which accounts for a large number of computations and memory needed. Consequently, we have limited our search in only 3rd order Volterra convolutions.

2.4.2 Model Architecture

The modern architecture of Wide-ResNets [47] will serve as the backbone of our model. These models are build on Residual Networks of CNNs [16]. The basic mathematical form of the models can be written as:

$$x_{l+1} = x_l + \mathcal{F}(x_l, \mathcal{W}_l) \quad (23)$$

Where x_{l+1} and x_l are the inputs of the layers, \mathcal{W}_l are the parameters of that layer and \mathcal{F} are the residual connections.

It is proved [15] that these connections transfer the information and tackle the diminishing gradients problem. In other words the residual connection pass the information from the deeper to the first layers of a NN and thus they accelerate the learning procedure in the whole span of the model. Residual Networks are consisted of two types of blocks, the basic and the bottleneck. Basic blocks stack two consecutive convolutions with a kernel size of 3×3 following by Batch Normalization and ReLU layers. The bottleneck layer is placed between the residual connections and in order to tackle the dimensionality reduction and increment, it utilizes 1×1 convolutions, which can adapt on the desired dimension.

The order of batch normalization, activation, and convolution in the residual block was modified from conv-BN-ReLU to BN-ReLU-conv in comparison to the original [16] ResNet models. It has been empirically demonstrated in [47] that it yields better outcomes while being faster. In particular, [47] focuses on basic residual architecture and presents various improvements that increase the performance of the base models. These advancements include the inclusion of additional convolutional layers as well as the addition of more feature channels to the convolutional layers.

Table 1 depicts the overall structure of our networks: It is composed of an initial Volterra convolutional block `volterra_conv`, followed by three groups (each of size N) of

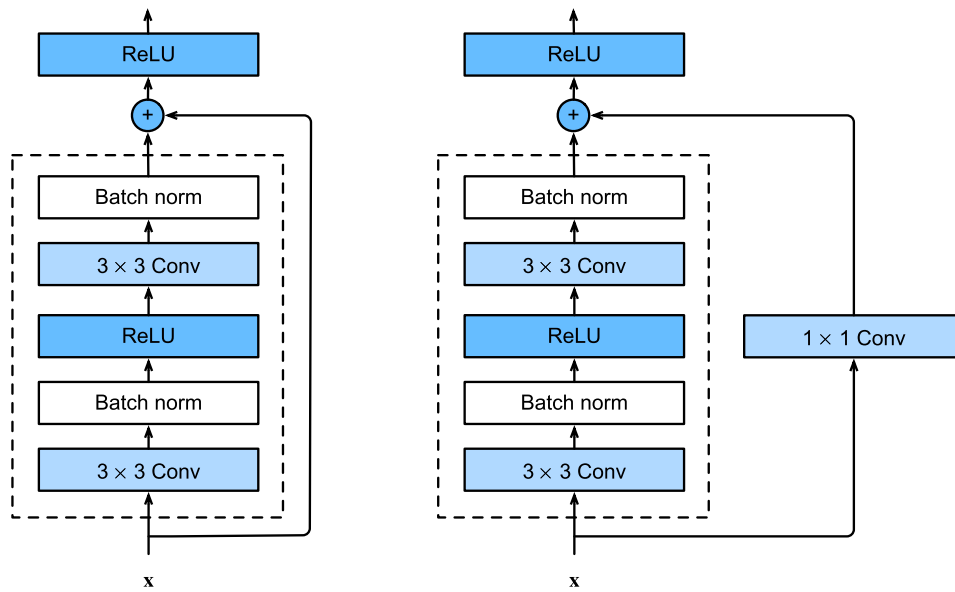


Figure 2: The ResNet blocks are depicted in this diagram. The basic block is given in the first column by the sequence of convolutional, batch normalization, and ReLU layers. The usage of 1×1 convolutions on residual connections is presented in the second column.

residual blocks group1, group2 and group3, followed by average pooling, and finally a classification head. The width of the remaining blocks of the in the three groups conv2-4 is scaled by a set widening factor $k = 10$. Also the N depth factor of the model is set to 4. We conduct and test many alterations to the number of feature channels of Volterra convolutions in order to determine the impact of representational power. These variations are explained in the following subsections.

Table 1: This table covers the model’s design in depth. The model’s phases are listed in the first column. The size of the picture as it outputs at each stage is indicated in the second column. The third column goes into great depth on the parameters of each block. For the Volterra convolution stage, we have two options. One is to use Linear convolutions in the first layer of the model and the other is to replace them with Volterra convolutions on the first layer of the model.

Model Block	Output	Block Parameters	
Volterra convolution / Linear convolution	32×32	Volterra_conv $3 \times 3 / 5 \times 5, 16 / 160$ Linear convolution $3 \times 3, 16$	
Group1	32×32	Conv	$\begin{matrix} 3 \times 3, 16 \cdot k \\ 3 \times 3, 16 \cdot k \end{matrix} \times 4$ blocks
Group2	16×16	Conv	$\begin{matrix} 3 \times 3, 32 \cdot k \\ 3 \times 3, 32 \cdot k \end{matrix} \times 4$ blocks
Group3	8×8	Conv	$\begin{matrix} 3 \times 3, 64 \cdot k \\ 3 \times 3, 64 \cdot k \end{matrix} \times 4$ blocks
Pooling	8×8 8×8 1×1	BN ReLU Average Pooling 8x8	
Classification Head		MLP 10 / 100 CrossEntropyLoss 10 / 100	

3 EXPERIMENTS

In this section, we'll look at the many versions of the models that were tested, the experiments and methodology that were used with the datasets, and eventually, we'll discuss the experiment findings.

3.1 Experiments Setup

Several experiments were performed. Initially, by keeping frozen the basic parameters of the models, the potential of non-linear in relation to linear convolutions was tested. Then, after we have detected the best non-linear convolutions, the quadratic one, we test with grid search a specific parameter in the model in order to fine tune it. For that reason, the selection of the model and training parameters needs to be explained in depth.

3.1.1 Model Hyper-parameters

Table 1 shows the various settings that will be used to test the model. To evaluate the performance of the model without the Volterra ones, we first use linear convolutions with no non-linear components. We set the linear model to the test with 16 and 160 original channels. Then, at the position of the first convolutional layer, we alter the linear ones and apply non-linear convolutions of the second and third orders in square and cubic form. The non-linear convolutions are also tested with 160 and 16 different channels to further comprehend their significance in the model's operation. We do not use skip connections between the first layer and the first group of the model when the model contains 160 starting channels. This is done to understand the non-linear contribution to the model and to transfer more easily the higher order features, to the deeper layers of the model.

We experiment with the size of the kernels and their consistency with the dilations in addition to the model's channels. Table 5 details the particular adjustments made to the parameters of the first layer of nonlinear convolutions. We experiment with the kernel sizes of 3×3 pixels and 5×5 pixels in Volterra convolutions. We also attempted 7×7 , however due to the square and cubic memory requirements, this non-linear convolution could not be calculated with our GPUs. The size of the kernels is very essential since it allows us to acquire more information and detect interactions between picture elements at greater distances. As a result, their testing is seen as very crucial.

Also the next parameter which is considered to play a decisive role in the expressive property and capacity of the model are the dilated convolutions [45]. The visuals of the dilated convolutions can be found in figure 3. Dilated convolutions, as stated in [46], enhance the receptive field of the top layers, accounting for the receptive field loss

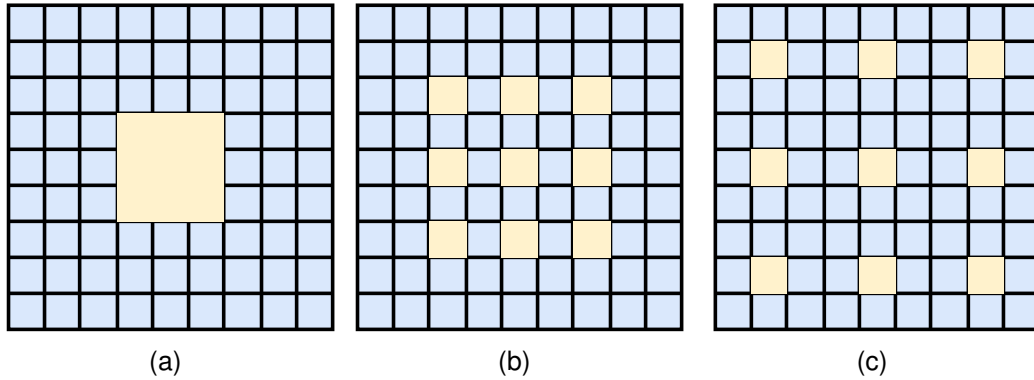


Figure 3: In these pictures the kernels are represented at different values of dilation. (a) represents the kernel of dilation 1; (b) depicts the kernel of dilation 2; (c) depicts the kernel of dilation 3.

caused by the reduction of convolutional patches. By incorporating these formulas we can, in addition to these observations, especially in the case of Volterra convolutions, take into account correlations in the data between unrelated points in the image. These points may have some interesting property or interaction which is located very far between the pixels of the image. As a consequence of this distance the simple non-dilated kernels could not perceive it. For this reason we are experimenting with the introduction of grade 2 and 3 dilatations in our non-linear convolutions to benefit from that phenomenon.

In parallel to exploring with the structure of our model, we also attempt different masking and scaling techniques. With the method of masking we essentially exclude the interactions that have been calculated twice due to the multiplication of the matrices. These interactions are unnecessary and will theoretically be the same values and will only add noise to the model parameters. Nevertheless we do experiments with and without this stage in our non-linear convolutions.

Lastly, we try to remove the scaling factor. This is vital as it regulates the number of parameters and gradients from the large numbers resulting from the quadratic and cubic forms. With these large values, problems such as vanishing and exploding gradients [30] could occur, however, it is worth trying the models without this parameter and observing the non-linear behavior in conditions of large factors.

3.1.2 Training Hyper-parameters

To train the model, many current approaches were applied. Table 2 specifies the parameters. For the initial learning rate, we employed a scheduled learning rate plan beginning with 0.1 while having a degree of shrinkage of $\gamma = 0.2$ for every 60 epochs at the start and every 20 epochs at the end of the training. For the optimizer we used the SGD optimizer with momentum set to 0.9 [39]. This is an excellent parameter

that has been found to adequately and successfully converge models in a number of circumstances. The training takes place for 220 epochs and the batch size is set to 128.

For the loss function we have used the Cross Entropy loss. When the model need to decide amongst a large number of classes, this is an essential technique. The Soft-max and Negative Likelihood functions are combined in this loss function. The first normalizes the values from 0 to 1 so that we can see the model's result clearly, and the second adjusts the error based on how near or distant the forecast is in the result vectors.

The parameters were generated via Xavier initialization. This has been demonstrated to improve model convergence by avoiding the issues of exploding and diminishing gradients. As a result, we put our trust in it for the special task of initializing the non-linear parameters. Xavier initialization can be found in 24 where W are the weights and n is the size of the input dimension.

$$W = U \left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}} \right) \quad (24)$$

For the regularization of the model's parameters L2 regularization [25] has been employed, set to 0.0005. Weight decay has been shown to improve model generalization by lowering the number of weights and so minimizing noise and biases that may have built up in them.

Moreover, another regularization approach was used in Dropout. Dropout [38] is a regularization technique that zeroes randomly the activation values. This limitation drives the network to learn more robust properties rather than depending on a small selection of neurons in the network's predictive capabilities. We set the probability of dropout to 0.3.

Regarding the training data, there has been a reprocessing step before the training. More specifically, the CIFAR10 images have been normalized by subtracting their means and dividing by their variance. As result the input data has a mean of 0 and variance of 1, which ameliorates the training. Finally, on the training data there have been applied mild data augmentations. These lend themselves positively to the training process as on the one hand they produce new data from the existing ones and secondly they have a regularisation role as the parameters are exposed to new altered data. The data augmentations that we apply like [47], are:

- Horizontal flipping with a probability of 0.5
- Reflection padding of 4 pixels
- Random crop of 32×32 pixels

Table 2: This table explains the settings of the learning rate, weight decay, optimizer, and momentum per epoch period.

Epoch	Learning rate	L2 regularization	Optimizer	Momentum
0-60	0.1	0.0005	SGD	0.9
61-120	0.02	0.0005	SGD	0.9
121-160	0.004	0.0005	SGD	0.9
161-200	0.008	0.0005	SGD	0.9
201-220	0.00016	0	SGD	0.9

3.1.3 Training Process

The model and non-linear convolutions are implemented in PyTorch [31]. The experiments have been carried out in the Google Colab Pro environment and the Nvidia Tesla P100 and K80 GPUs have been used. A total of 30 computational days were spent for the early and final experimentation of these models. The most detailed results, the graphs for each metric, comparative graphs as well as the trained parameters of the best models can be found in the online platform of Weights and Biases in this link https://wandb.ai/andreas_giannoutsos/Volterra-Convolution.

3.2 Datasets

For the experiments we use the CIFAR10 and CIFAR100 datasets [23]. This datasets consists of 60000 images of size 32×32 and there are two versions either with 10 or with 100 classes. Of the 60000 images, 50000 are indented for training and 10000 for model testing. The classes are evenly distributed as in CIFAR10 for each class it has 6000 different images and CIFAR100 has 600 images for every class. In CIFAR100 the 100 classes form 20 large categories from which smaller sub-classes. The objects in these 10 classes represent airplanes, dogs, cats, cars, horses, frogs, trucks, deer, ships and birds. CIFAR100 as with the huge number in classes and the limited number of examples that represent them is a very demanding dataset and requires from the models capable generalization properties quickly without much data.

3.3 Results and Discussions

First, we examine the findings of tests on different non convolutions, such as quadratic and cubic forms, and compare them to linear ones. They were tested in the datasets CIFAR10 and CIFAR100, as shown in graphs 4 and 5, respectively. In all datasets, we employed 160 channels for the initial convolutions and kept them frozen throughout the tests. For each model, we tried three distinct training approaches, each time randomizing its parameters from the start. We repeated the process twice for the

CIFAR100 dataset. Tables 3 and 4 show the results for the CIFAR10 and CIFAR100 datasets, respectively.

Table 3: This table lists the accuracy and mean accuracy on CIFAR10 for each model evaluated between Linear, Volterra 2nd, and Volterra 3rd order, together with the number of parameters for each model in millions. Volterra 2nd order had the greatest single and mean accuracy, indicating that Volterra convolutions enhance the model’s performance.

Model	CIFAR10 Accuracy			Mean Accuracy	Parameters
Linear	95.39	95.21	95.20	95.26	36.6M
Volterra 2nd order	95.22	95.50	95.44	95.38	36.7M
Volterra 3rd order	95.31	95.20	95.39	95.30	36.8M

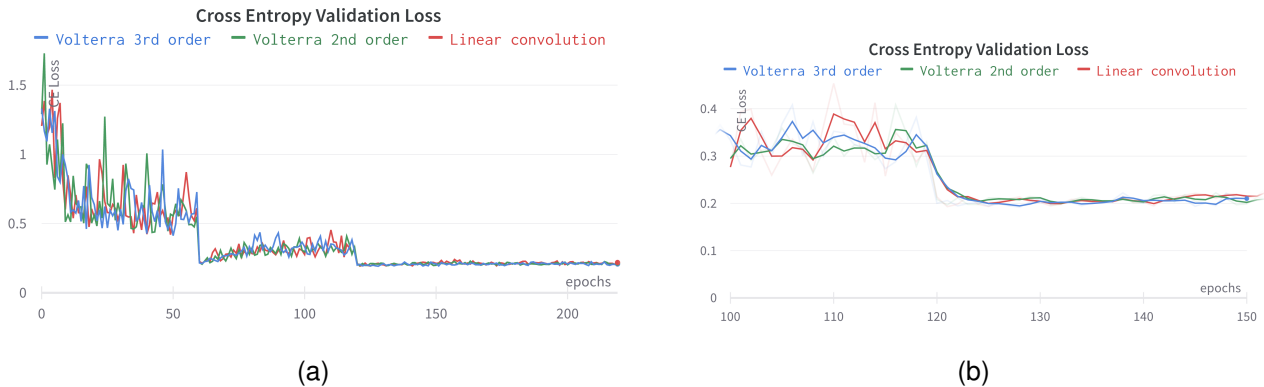


Figure 4: These plots show the Validation Cross Entropy Loss of our three top performing models on the CIFAR10 dataset per epoch of our training process; (a) represents the total loss from the beginning of the training to the end; (b) depicts a portion of the training process between 100 and 150 epochs in which a decrease in learning rate is observed. This diagram’s values are also normalized and averaged using exponential moving averaging.

As can be seen from the findings, using nonlinear convolutions rather of linear ones led to a considerable increase over the accuracy of the CIFAR10 dataset. This enhancement is accomplished by the use of 2nd order Volterra convolutions. Simultaneously, the cubic ones outperform the linear ones in terms of average accuracy performance. However, despite having more parameters and a more robust capacity to collect information from pictures, they perform no better than quadratic.

In contrast to CIFAR100, there is no gain in performance, but rather a decline. This phenomena can be explained by a variety of factors. Initially, the CIFAR100 dataset is significantly more complex. This is demonstrated by graph 6, which depicts the loss difference between CIFAR10 and CIFAR100. We may deduce from this that the linear and non-linear models have a tough time adjusting to so many distinct classes with so little data.

Furthermore, the potential of Volterra convolutions to record the interactions between data is clear. To do so, we’ll need enough data for each class, as the CIFAR100

Table 4: This table shows the accuracy and mean accuracy on CIFAR100 for each model evaluated between Linear, Volterra 2nd, and Volterra 3rd order together with the number of parameters for each model in millions. The Linear model obtained the highest single and mean accuracy.

Model	CIFAR100 Accuracy		Mean Accuracy	Parameters
Linear	77.91	77.74	77.50	36.8
Volterra 2nd order	77.57	77.19	77.38	36.8
Volterra 3rd order	77.64	76.93	77.28	37.1

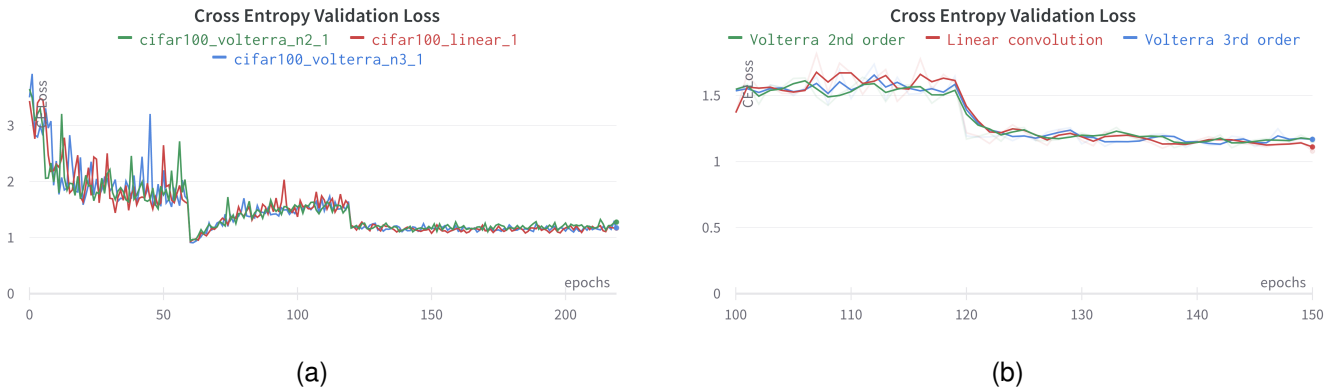
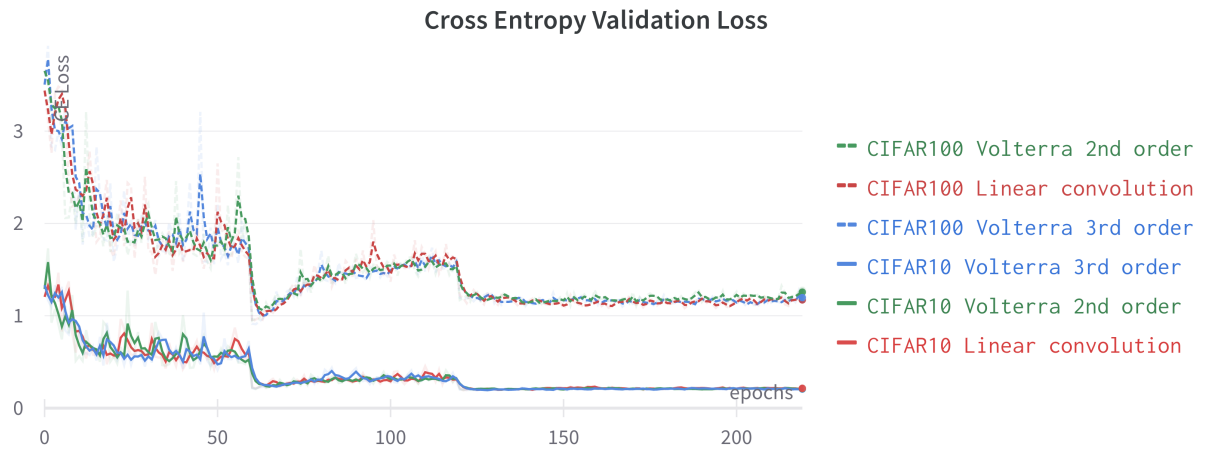


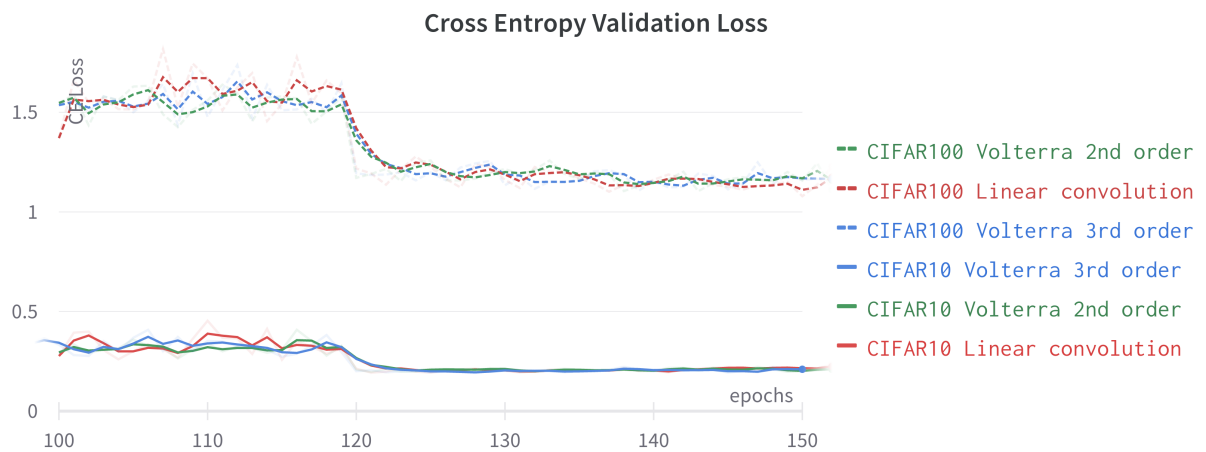
Figure 5: These charts depict the Validation Cross Entropy Loss of our three best performing models on the CIFAR100 dataset each training epoch; (a) indicates the overall loss from the start of the training to the finish. A section of the training process between 100 and 150 epochs is illustrated in (b), where a reduction in learning rate is visible. The numbers in this chart are also adjusted and averaged using exponential moving averaging.

limit only allows for a few hundred images per class. This concept of convolutional generalization will be investigated further in the activations discussion section 3.3.2.

We can also see a drop in loss from the graphs 4 and 5 when the scheduler reduces the learning rate. As a result, the optimizer falls into smaller curves of the loss plain that were relatively unexplored because to the higher learning rate. The graphs demonstrate that the training curve converges for every model, thus we can be more certain that we will not stick to local minimums using this strategy.



(a)



(b)

Figure 6: The Validation Cross Entropy Loss of all of our best performing models on both the CIFAR10 and CIFAR100 datasets is shown in these figures; (a) displays the total loss curve from the beginning to the completion of the training; (b) represents a section of the training process between 100 and 150 epochs, during which a reduction in learning rate is evident, along with the loss. The data in this diagram are also scaled and averaged using exponential moving averaging.

3.3.1 Grid Search

We attempted to fine tune our models with two extensive grid searches in order to maximize model experimentation while also clarifying the function of non-linear convolutions better. In the first, we look at the structure of the initial convolution, and in the second, we look at the structure of the Volterra block’s usage of masking and scaling.

Table 5: This table shows the grid search parameter values used to identify improved channel, kernel size, and dilation parameters in Volterra convolutions.

Parameter	Values
Volterra Channels	16, 160
Kernel size	3×3 , 5×5
Dilation	1, 2, 3

Table 6: This table describes the parameter values for grid search in order to find better parameters of scaling and masking in Volterra convolutions.

Parameter	Values
Scaling	True, False
Masking	True, False

Tables 5 and 6 show the hyper-parameters that were examined at each grid search. From the examinations of the results and the corresponding hyper-parameters we can calculate their correlation. The correlation can be found at figure 7. The calculation of the correlation together with an importance factor determined by the importance of every hyper-parameter with the use of random forests is performed using the WandB platform ¹.

Regarding the first general grid search of the convolution’s hyper-parameters, we can clearly see a strong negative connection between the number of channels, with the model performing better as the number of channels is reduced. It is worth noting that the model performs best in non-linear convolutions with 16 channels rather than 160. There is also a minimal association between the number of dilations and the size of the kernel, which renders them ineffective for the ultimate performance and capabilities of the model.

The determination of the number of channels is a difficult task. As explained in [41], the number of channels must be determined by maintaining a balance between the depth of the model, the size, and the number of channels. Furthermore, in [47], the optimal number for the channels that prompt is 16, therefore the non-linear co

¹https://wandb.ai/andreas_giannoutsos/Volterra-Convolution/sweeps

Table 7: The optimal parameter settings for the Volterra convolution are described in this table, for the 12 best performing models.

Best Volterra Order	Channels	Kernel size	Dilation	CIFAR10 Accuracy
Volterra 2nd order	16	3	2	95.53
Volterra 2nd order	16	5	2	95.51
Volterra 2nd order	16	5	2	95.48
Volterra 2nd order	16	3	2	95.46
Volterra 2nd order	16	5	3	95.42
Volterra 2nd order	16	3	3	95.40
Volterra 2nd order	160	3	3	95.29
Volterra 2nd order	160	3	2	95.29
Volterra 2nd order	160	5	2	95.29
Volterra 2nd order	160	5	3	95.18
Volterra 2nd order	160	5	1	95.14
Volterra 2nd order	160	3	1	95.13

Table 8: This table describes the optimal scaling and masking settings, resulting in the best 4 models of our experimentation.

Best Volterra Order	Scaling	Masking	CIFAR10 Accuracy
Volterra 2nd order	True	False	95.60
Volterra 2nd order	True	True	95.45
Volterra 2nd order	False	True	85.12
Volterra 2nd order	False	False	84.67

could not escape using this strategy. Furthermore, the physics of the nonlinear convolution is challenging since multiplication with big numbers can introduce numerical instability into the model. As a result, fewer channels can be preferable. This phenomena is studied in greater depth in the second grid search. Although dilatation was significant, the size of the kernels did not play a major influence. This is due to the fact that there are numerous interactions in a huge 5×5 kernel, and a lot of information might be lost during the process of adding the elements. However, dilatation can convey information to distant locations in the image since the objects of interest may be far away and may not be perceived with a tight kernel.

Scaling is clearly important to the performance of the non-linear models in the second grid search. The mask and upper triangular panels, on the other hand, aren't really useful in the model. The decision to impose scaling on our model proved to be correct, since high values of the inner product cause instability in our computations and the gradients become exceedingly tiny or huge, posing a challenge in training our model.

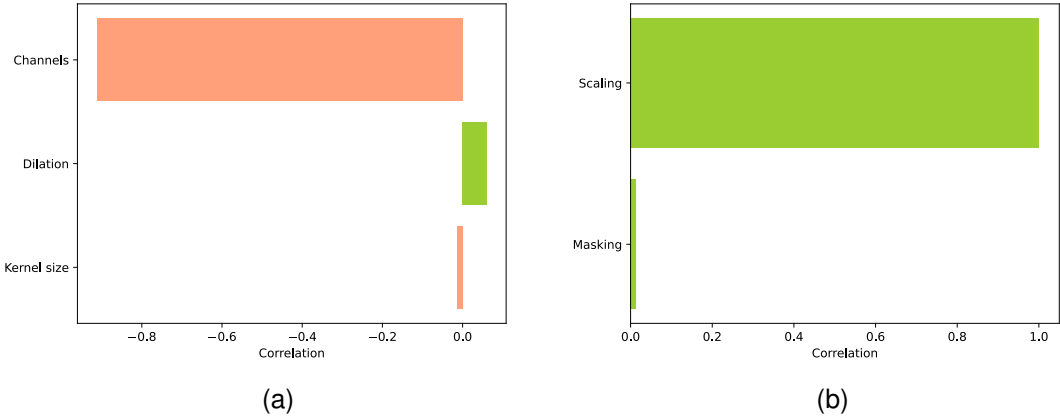


Figure 7: These graphs show the correlation between the parameters of the grid searches. (a) describes the values for the first general grid search while (b) describes the correlation in the parameters of the 2nd grid search with scaling and masking factors.

3.3.2 Linear and Volterra Activations

The observation and visualization of Volterra convolutions' activations is an intriguing feature. We acquired pictures from the activations of the 2nd and 3rd order Volterra convolutions in the following experiment. We took samples from the beginning and completion of the training for these images. The visuals are generated by the model's initial activation channel for each layer. Figures 9 and 10 depict activations of 2nd order non-linear convolutions at the start and end of training, respectively, while Figures 11 and 12 illustrate activations of 3rd order non-linear convolutions.

On our figures of activation visualizations we can identify at the competition of the training process an interesting pattern at the Volterra activations. The points that defy an object in an image are most of the time highlighted with grater values. This pattern continues on both the 2nd and 3rd order Volterra convolutions, while in the 3rd order there is an additional layer which follows that pattern more clearly. By analyzing these samples, we can detect many similarities with transformer visualization techniques. In these papers, the attention activations of the compatibility between the input vectors are illustrated. It is crucial to note in this case that some of the examples we will examine at below primarily employ attention layers rather than self-attentions. Self-attention, on the other hand, is a sub-procedure of the attention process. Attention, on the other hand, has additional processes that boost its expressive ability, as we already discussed in the section 2.3.2. As a result, the comparison of attention activations is nearly comparable to the comparison of self-attestation activations. To begin with, similarities between the activations of the non-linear Volterra layers and the attention maps of [8] can be identified. The latter incorporates self-attention to the visual problems and in its images it depicts the attention probabilities per input image. Furthermore, many resemblances can be detected with [5]. In this work, many illustration of the attention maps of images presented. The points of interest in an image are notated with higher values. Finally, these patterns can be also found in [4] which is a notable research work that exploits the vastness of unsupervised learning on a Vision Transformer model and emphasizes the visualization of the attention maps.

We may deduce the partial resemblance between Volterra convolutions and self-attention layers theoretically and optically by studying the previous facts. The self-attention approach highlighted the points of interest in each of the cases we examined. This is a common characteristic of attention models since it assesses the correlation between visual elements. If there is importance, this aspect of the image may contribute more to its classification. The presence of this property, as well as the similarities in the mathematical formula, demonstrate the non-linear convolutions' overall resemblance to self-attention layers.

Table 9: This table depicts the early initial activations of our Volterra 2nd order network for distinct CIFAR10 classes at the commencement of our training procedure. The first column displays the original input image; the second column demonstrates the output of the linear convolution layer; the third column exhibits the activations of the 2nd order Volterra layer; and the fourth, fifth, and sixth columns display the activations of the group1, group2, and group3 convolutional blocks, respectively.



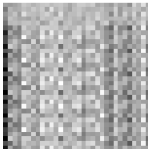

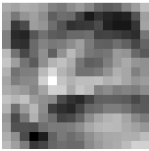



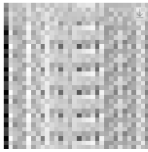

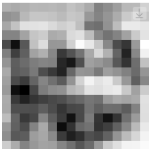
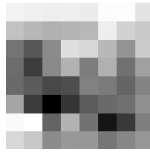


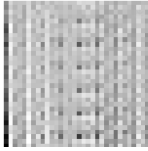
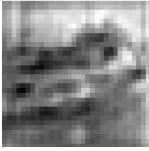
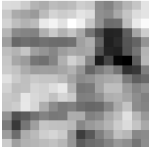
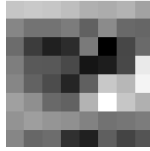


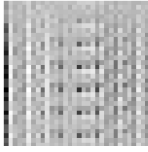


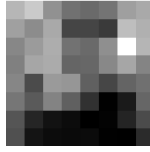
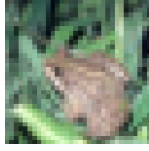
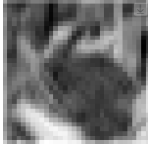
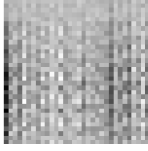
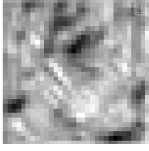
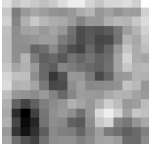

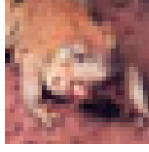

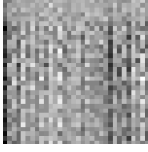
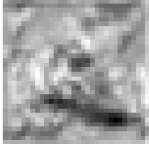
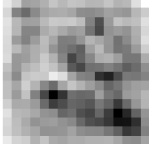

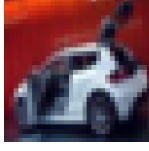
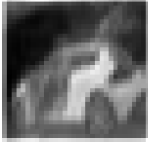
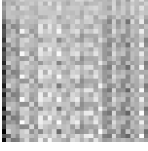
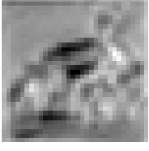
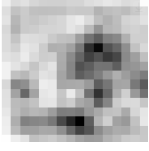
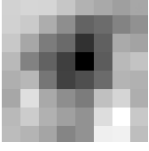
	Input	Linear	2nd order	Group1	Group2	Group3
Cat						
Ship						
Ship						
Airplane						
Frog						
Frog						
Car						

Table 10: This table illustrates the later intermediary activations of our Volterra 2nd order net for different CIFAR10 classes at the completion of our training phase. The first column displays the original input image; the second column shows the output of the linear convolution layer; the third column showcases the activations of the 2nd order Volterra layer; and the fourth, fifth, and sixth columns demonstrate the activations of the group1, group2, and group3 convolutional blocks, accordingly.


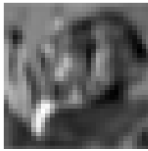
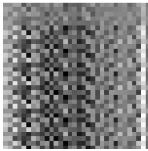
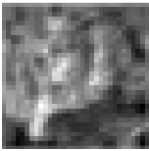
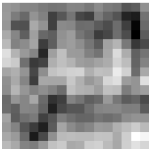
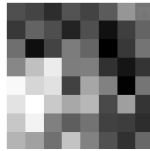


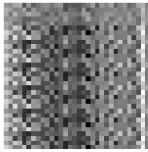
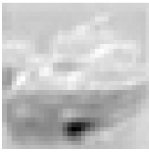




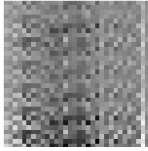
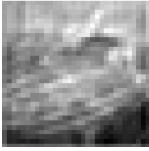
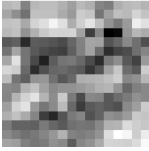



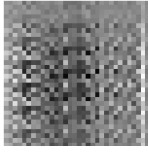



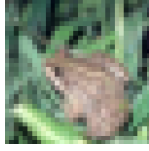
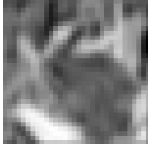
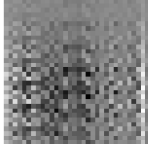
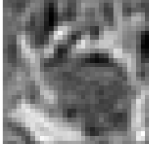
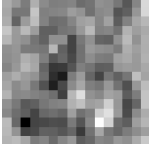
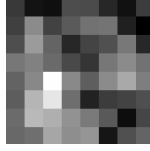
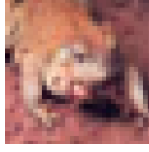

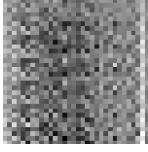
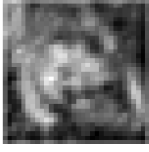
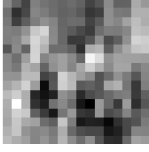

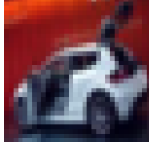

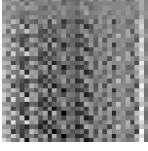

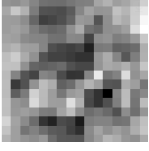
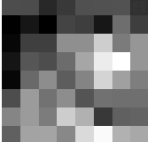
	Input	Linear	2nd order	Group1	Group2	Group3
Cat						
Ship						
Ship						
Airplane						
Frog						
Frog						
Car						

Table 11: This table illustrates the early intermediate activations of our Volterra 3rd order network at the beginning of our training process for various CIFAR10 pictures. The original input image is shown in the first column; the output of the linear convolution layer can be seen in the second column; the activations of the 2nd order Volterra layer are included in the third column; the activations of the 3rd order Volterra layer are shown in the fourth column; and the activations of the group1, group2, and group3 convolutional blocks are depicted in the fourth, fifth, and sixth columns correspondingly.


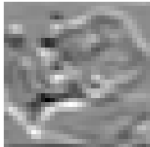
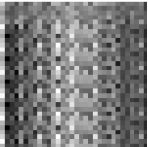
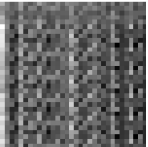

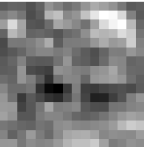



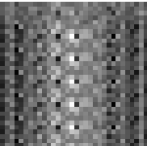
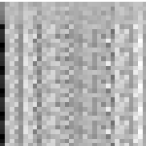
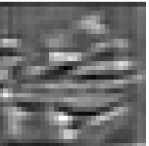
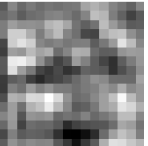
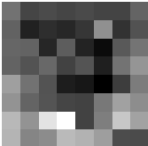
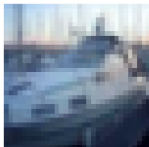
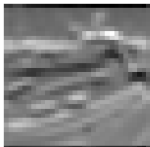
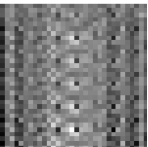
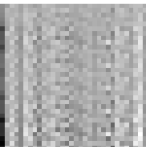
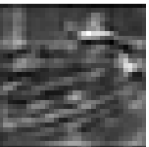
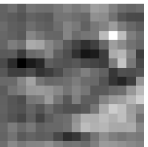



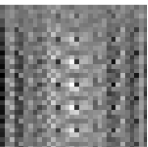
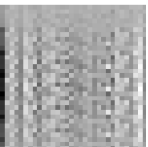

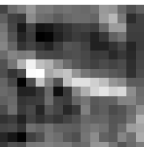


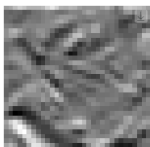
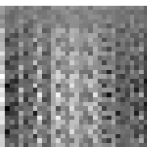
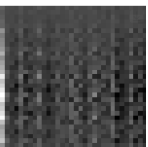
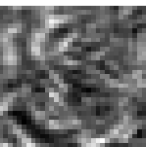
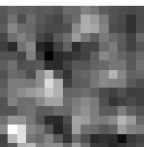

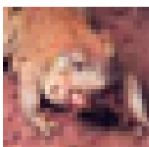
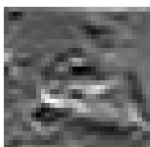
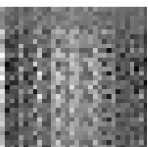
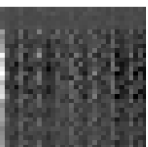
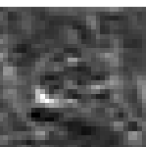
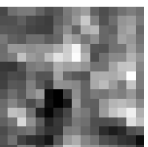



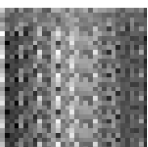
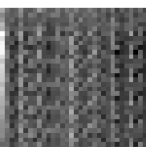

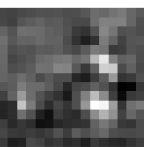
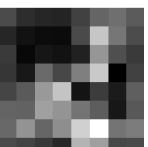

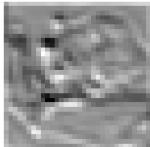
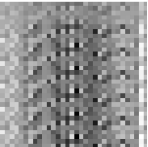
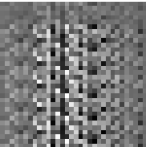
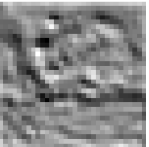
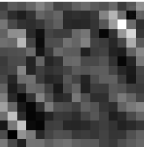
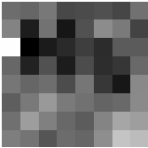


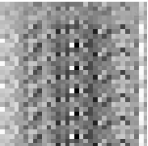
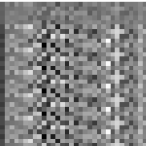

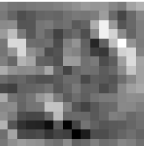

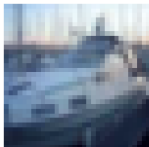
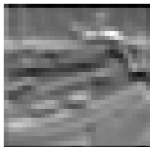
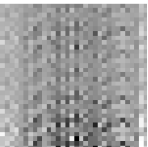
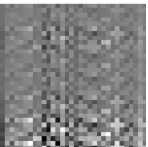
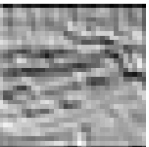
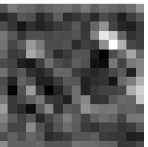



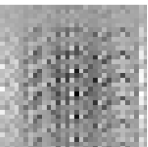
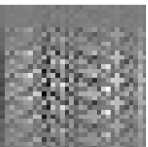

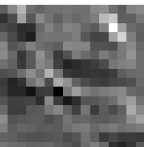


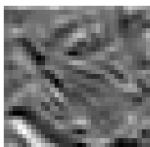
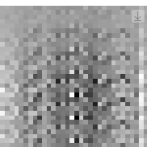
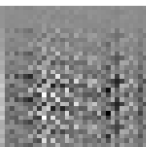
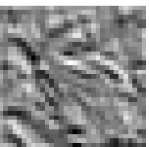
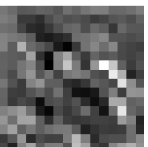

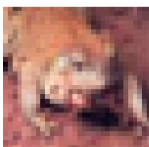
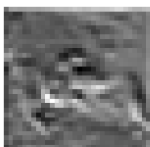
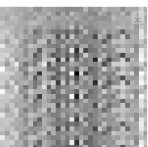
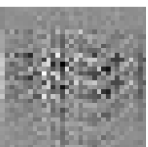
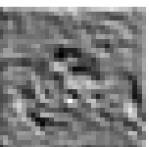
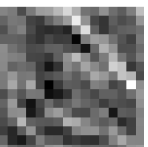
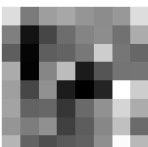


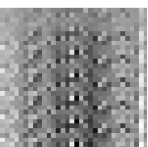
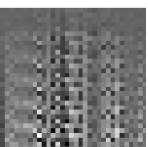

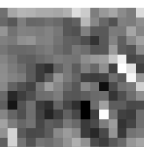
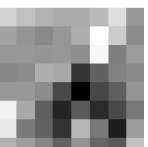
	Input	Linear	2nd order	3rd order	Group1	Group2	Group3
Cat							
Ship							
Ship							
Airplane							
Frog							
Frog							
Car							

Table 12: This table illustrates the late intermediate activations of our Volterra 3rd order network at the end of our training process for various CIFAR10 classes. The original input image is shown in the first column; the output of the linear convolution layer can be seen in the second column; the activations of the 2nd order Volterra layer are included in the third column; the activations of the 3rd order Volterra layer are shown in the fourth column; and the activations of the group1, group2, and group3 convolutional blocks are displayed in the fourth, fifth, and sixth columns accordingly.

	Input	Linear	2nd order	3rd order	Group1	Group2	Group3
Cat							
Ship							
Ship							
Airplane							
Frog							
Frog							
Car							

3.3.3 Linear and Volterra Parameter Distributions

In this sub-section we will explain the confirmation of the theory of the existence of polynomial approximation functions which can simulate neural network models with the observation of the parameter distributions at various layers and depths of our models. In respect of sampling, we collected data at the end of training from the top models in each type of non-linear convolutions. The weights are obtained from many different layers of the models and are shown as histograms, with the horizontal axis displaying their values and the vertical axis indicating the probability density of each value's occurrence. The horizontal axis is very essential since it displays the variance of the weights.

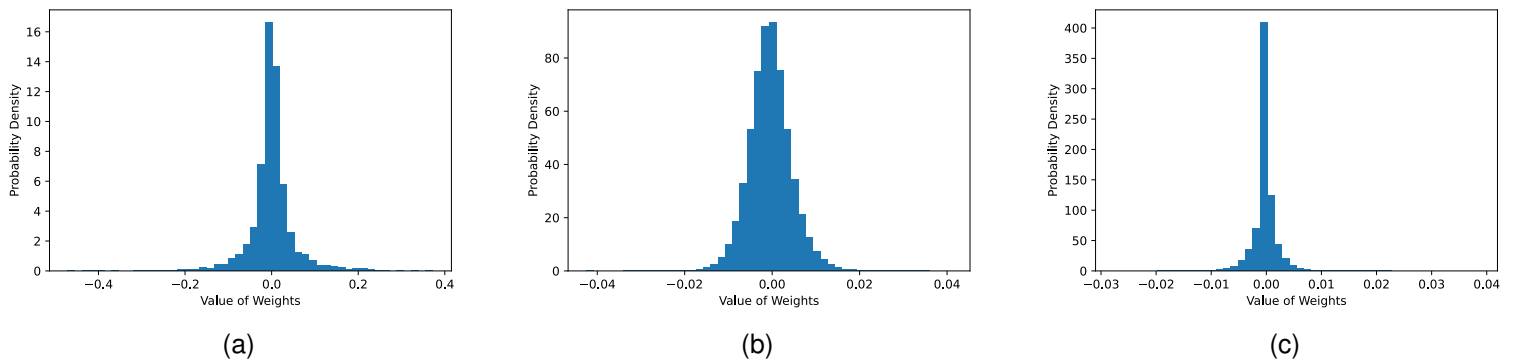


Figure 8: The probability density of the weights in three distinct convolution layers of the Linear model is shown in these histograms; The weights of the first Linear convolutional layer are represented by (a); the weights of the second group layer of convolutions are represented by (b); the weights of the third group layer are represented by (c).

In figure 8 the distributions of the linear model's weights are illustrated. As the distributions of the CNN model's first, second, and third layers are collected, the weights follow similar distributions, with the main mass having values close to zero. The variance, on the other hand, has reduced by ten times from the first to the second layer, while it has fallen by two times from the second to the third layer. This is to be expected because deep models are multi-layered and include multiple non-linear processes that vary and reduce values to keep the models stable. As we move further into the model, the contributions of the layers to the final output get reduced as the filters capture less important characteristics [48].

In the weight distributions of non-linear convolutions in figures 9 and 10, we see a similar trend. In figure 9 the linear, the 2nd order Volterra convolution and the group3 weights distributions are depicted. In figure 10 the linear, the 2nd order Volterra convolution and the the 3rd order Volterra convolution weights distributions are presented. Of course, there is a significant increase in the values of 0 in the weights of the 2nd and 3rd order non-linear convolutions. This is due to the zero mask applied to the weights, which prevents the provided data interactions from multiplying again. This,

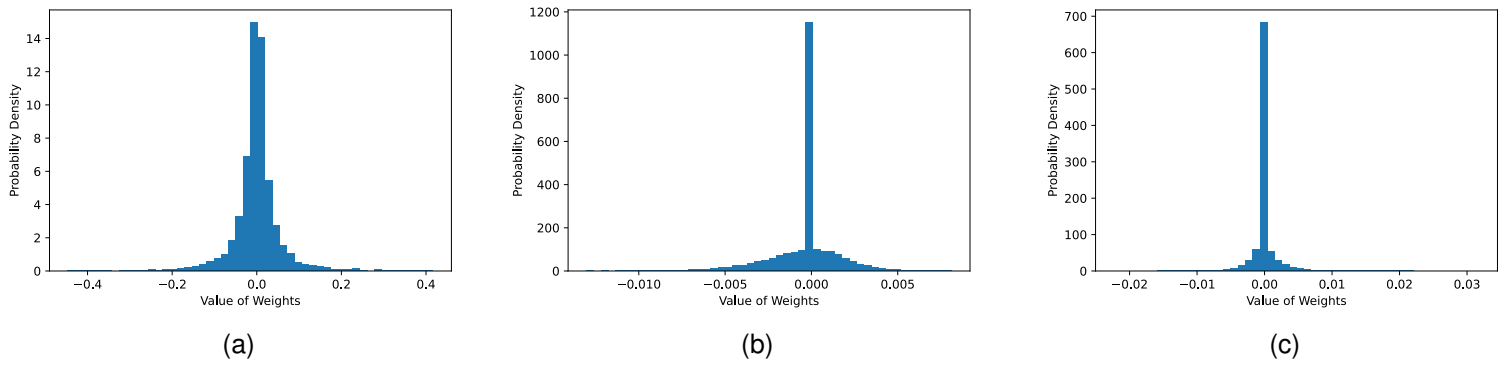


Figure 9: In these histograms, the probability density of the weights in 3 different convolution layers of the Volterra 2nd order model are presented; (a) includes the weights of the Linear Volterra convolutional layer; (b) shows the weights of the non-linear weights of the 2nd order Volterra convolution; (c) presents the weights of the third convolutional group layer.

however, does not preclude us from investigating the variations in these models. In these histograms we notice something very crucial. The weights between the 2nd and 3rd order non-linear convolutions have a decrease in their variance similar to that when we go from the first to the second and the third layer of the CNN model. This reduction in the order of 10 can be seen throughout the non-linear convolutions. Nonetheless, this behaviour is also normal, since the higher order convolutions have quadratic and cubic terms which need to be multiplied by smaller parameters in order to maintain the numerical stability that the network needs. Simultaneously, non-linear Volterra convolutions are a series of additions of terms multiplied by each other on the first, second, and third order. This structure is identical to that of the polynomial, as we mentioned in the introduction. Meanwhile, in the introduction, we detailed the attempts to compare neural network models with polynomial functions. Assuming that the non-linear model is a polynomial with the degree of the parameters decreased by ten times in each term, the linear CNN model is similarly a polynomial. Its terms are the many layers that it incorporates since their parameters have the same variance reduction rate as non-linear ones. Volterra convolutions, as a polynomial form, behave similarly to the several distinct layers of a linear CNN model. As a result, linear CNN models behave similarly to polynomials. This is a simple observation, based on the parameter distributions of these models.

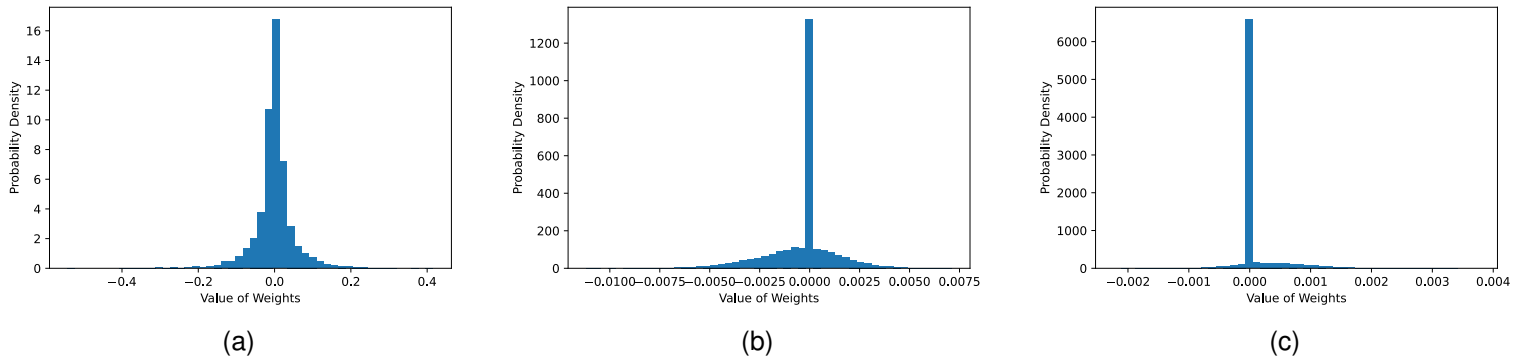


Figure 10: The probability density of the weights in three distinct convolution layers of the Volterra 3rd order model is shown in these histograms; (a) includes the weights of the Linear Volterra convolutional layer; (b) explains the weights of the non-linear weights of the 2nd order convolution of the 3rd order Volterra model; The weights of the non-linear weights of the 3rd order Volterra convolution can be seen in (c).

4 CONCLUSION

Non-linear convolutions were developed in response to the desire to create models of greater capacity that are more susceptible to data variations. Initially, we presented a PyTorch implementation of the non-linear Volterra convolutions, which was supplied with thorough mathematical notes and techniques for the model's architecture. Then we trained the non-linear models, which outperformed the linear ones in image classification. Following that, we attempted to optimize the model by running several grid searches, and we arrived at the optimal configuration of the model's hyper-parameter. Finally, we demonstrated the resemblance of non-linear Volterra convolutions to the self-attention mechanism, as well as the link of high order multi-linear polynomial approximation functions with deep CNNs.

Non-linear convolutions have the potential to make significant contributions to the field of computer vision. Initially, an interesting question would be the degree of their contribution to the recognition process with appropriate coefficients. Then, as we referred to their similarity to self-attentions, further investigation of this direction is of particular importance. For example, we could introduce new hybrid architectures where self-attentions and co exist under the same mathematical framework. Finally, these hybrid models could be part of a polynomial approximation function by integrating all these ideas into a mathematical model with known properties which can be extended to other machine learning problems and modalities. This is an intriguing future path, the unification of neural network components under a mathematical framework that might yield great achievements in the future.

ABBREVIATIONS-ACRONYMS

BN	Batch Normalization
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
GCNN	Graph Convolutional Neural Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
matmul	Matrix Multiplication
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
NN	Neural Network
ReLU	Rectified Linear Unit
ResNets	Residual Networks
SGD	Stochastic Gradient Descent
ViT	Vision Transformer

REFERENCES

- [1] P. Alper. “A consideration of the discrete Volterra series”. In: *IEEE Transactions on Automatic Control* 10.3 (1965), pp. 322–327.
- [2] Irwan Bello, Barret Zoph, Quoc Le, Ashish Vaswani, and Jonathon Shlens. “Attention Augmented Convolutional Networks”. In: *International Conference on Computer Vision (ICCV)*. 2019, pp. 3285–3294.
- [3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. “Massive Exploration of Neural Machine Translation Architectures”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 1442–1451.
- [4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jegou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. “Emerging Properties in Self-Supervised Vision Transformers”. In: *International Conference on Computer Vision (ICCV)*. 2021, pp. 9630–9640.
- [5] Hila Chefer, Shir Gur, and Lior Wolf. “Transformer Interpretability Beyond Attention Visualization”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 782–791.
- [6] Grigorios Chrysos, Stylianos Moschoglou, Yannis Panagakis, and Stefanos Zafeiriou. “PolyGAN: High-Order Polynomial Generators”. In: *CoRR* abs/1908.06571 (2019). arXiv: 1908.06571. URL: <http://arxiv.org/abs/1908.06571>.
- [7] Grigorios G. Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. “P-nets: Deep Polynomial Neural Networks”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 7323–7333.
- [8] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. “On the Relationship between Self-Attention and Convolutional Layers”. In: *International Conference on Learning Representations*. 2020.
- [9] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. “CoAtNet: Marrying Convolution and Attention for All Data Sizes”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. 2021.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2019, pp. 4171–4186.
- [11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021.
- [12] William Fedus, Barret Zoph, and Noam Shazeer. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity”. In: *CoRR* abs/2101.03961 (2021). arXiv: 2101.03961. URL: <https://arxiv.org/abs/2101.03961>.
- [13] Kunihiko Fukushima. “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. In: *Neural Networks* 1.2 (1988), pp. 119–130.
- [14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587.
- [15] Fengxiang He, Tongliang Liu, and Dacheng Tao. “Why ResNet Works? Residuals Generalize”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.12 (2020), pp. 5349–5362.

- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- [17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: <http://arxiv.org/abs/1704.04861>.
- [18] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. “Local Relation Networks for Image Recognition”. In: *International Conference on Computer Vision (ICCV)*. 2019, pp. 3463–3472.
- [19] D. Hubel and T. Wiesel. “Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex”. In: *Journal of Physiology* 160 (1962), pp. 106–154.
- [20] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando A. Mujica, Adam Coates, and Andrew Y. Ng. “An Empirical Evaluation of Deep Learning on Highway Driving”. In: *CoRR* abs/1504.01716 (2015). arXiv: 1504.01716. URL: <http://arxiv.org/abs/1504.01716>.
- [21] C. G. Khatri and C. Radhakrishna Rao. “Solutions to Some Functional Equations and Their Applications to Characterization of Probability Distributions”. In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 30.2 (1968), pp. 167–180.
- [22] Tamara G. Kolda and Brett W. Bader. “Tensor Decompositions and Applications”. In: *SIAM Review* 51.3 (2009), pp. 455–500.
- [23] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: 2009.
- [24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.
- [25] Anders Krogh and John Hertz. “A Simple Weight Decay Can Improve Generalization”. In: *Advances in Neural Information Processing Systems*. Vol. 4. 1991.
- [26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [27] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems*. Vol. 2. 1989.
- [28] S. Marčelja. “Mathematical description of the responses of simple cortical cells*.”. In: *J. Opt. Soc. Am.* 70.11 (1980), pp. 1297–1300.
- [29] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, pp. 807–814.
- [30] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the Difficulty of Training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. 2013, pp. 1310–1318.
- [31] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. “Automatic differentiation in PyTorch”. In: *NIPS-W* (2017).
- [32] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. “Stand-Alone Self-Attention in Vision Models”. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

- [33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.
- [34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *MICCAI*. 2015.
- [35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.
- [36] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: <http://arxiv.org/abs/1409.1556>.
- [37] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. “Bottleneck Transformers for Visual Recognition”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 16514–16524.
- [38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [39] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. “On the Importance of Initialization and Momentum in Deep Learning”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. 2013, pp. 1139–1147.
- [40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.
- [41] Mingxing Tan and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. PMLR, Sept. 2019, pp. 6105–6114.
- [42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.
- [43] Volterra Vito. *Sopra le funzioni che dipendono de altre funzioni*. 1887.
- [44] Mohsen Yavartanoo, Shih-Hsuan Hung, Reyhaneh Neshatavar, Yue Zhang, and Kyoung Mu Lee. “PolyNet: Polynomial Neural Network for 3D Shape Recognition with PolyShape Representation”. In: *International Conference on 3D Vision* (2021).
- [45] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *International Conference on Learning Representations (ICLR)*. 2016.
- [46] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. “Dilated Residual Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 636–644.
- [47] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2016, pp. 87.1–87.12.
- [48] Matthew D. Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *In Computer Vision—ECCV*. 2014, pp. 818–833.
- [49] Wei Zhang. “Shift-invariant pattern recognition neural network and its optical architecture”. In: *Proceedings of Annual Conference of the Japan Society of Applied Physics* (1988).
- [50] Georgios Zoumpourlis, Alexandros Doumanoglou, Nicholas Vretos, and Petros Daras. “Non-linear Convolution Filters for CNN-Based Learning”. In: *International Conference on Computer Vision (ICCV)*. 2017, pp. 4771–4779.