https://doi.org/10.2352/ISSN.2470-1173.2018.10.IMAWM-338

Hierarchical Auto-associative Polynomial Convolutional Neural Network (HAP-CNN)

Patrick Martell, University of Dayton, Dayton, Ohio; Vijayan Asari, University of Dayton, Dayton, Ohio

Abstract

Convolutional neural networks (CNNs) tend to look at methods to improve performance by adding more input data, making modifications on existing data, or changing the design of the network to better suit the variables. The goal of this work is to supplement the small number of existing methods that do not use one of the previously mentioned techniques. This research aims to show that with a standard CNN, classification accuracy rates have the potential to be improved without changes to the data or major network design modifications, such as adding convolution or pooling layers. A new layer is proposed that will be inserted in a similar location as the non-linearity functions in standard CNNs. This new layer creates a localized connectivity for each perceptron to a polynomial of N^{th} degree. This can be performed in both the convolutional portion and the fully connected portion of the network. The proposed polynomial layer is added with the idea that the higher dimensionality enables a better description of the input space, which leads to a higher classification rate. Two different datasets, MNIST and CIFAR10 are utilized for classification, each containing 10 distinct classes and having similar training set sizes, 60,000 and 50,000 images, respectively. These datasets differ in that the images are 28×28 grayscale and 32×32 RGB, respectively. It is shown that the added polynomial layers enable the chosen CNN design to have a higher rate of accuracy on the MNIST dataset. This effect was only similar at a lower learning rate with the CIFAR10 dataset.

Introduction

The rise of the use of CNNs also brought with it a focus on large networks and large amounts of data. Because of this, some of the most popular networks are AlexNet [1], Oxford's Visual Geometry Group's VGGNet [2], and Google's GoogLeNet [3]. In order to use these large networks for other tasks, transfer learning gained traction. This has caused a focus on problems that fit within this mold. Disregarding the large networks with large datasets and transfer learning approaches, there is still much to learn about CNNs and how to get the most out of a network designed and trained from scratch. The research focuses seem to fall into one of three categories: data augmentation, network design, or network augmentation. Data augmentation comprises of techniques that seek to get the most out of the training data, such as rotation, flipping, and other geometric functions. Network design focuses on the ordering of convolution, activation, and pooling layers. This work will focus on network augmentation: functions or layer modifications that can be inserted into any design.

Before discussing the proposed addition into the collection of works of network augmentation, it is pertinent to look at the already proposed methods. Arguably the most famous and effective is Dropout [4] and DropConnect [5]. These items seek to reduce over-fitting in training by dropping either the nodes or connections, respectively. This essentially tries to force the network to learn more than one path for a feature. In this proposed work, a dropout rate of 0.5 is used as it is fairly standard in most CNNs.

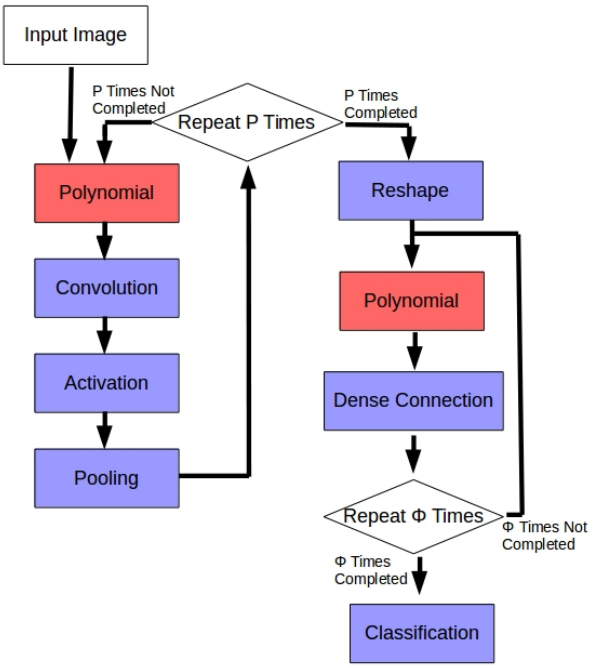

Figure 1. *HAP-CNN Flow Diagram*

Other interesting methods in the field look at enhancing the method of pooling. A work by Scherer, et al. [6] demonstrates that max pooling performs equally or better than aggregate pooling and overlapping pooling. Another work by Chen-Yu Lee [7] showed that max pooling was in fact not the best method, though methods that beat it were much more complex and had minimal gains. For this reason, along with their widespread use, this research will use max pooling.

Finally, there has been research into the different types of activation functions to use. Within AlexNet's research [1] the Rectified Linear Unit (ReLU) was shown to reach a training error rate six times faster than without. It also has a great appeal due to the incredible simplicity. Other methods such as Leaky ReLU [8], PReLU [9], and Maxout [10] are used. Many times these seem to be fixes to issues that arise from the standard ReLU. When activation is used, this work will use ReLU.

There has been little research performed looking into the effect of polynomials on neural networks. It seems to be first introduced by HAPNet [11], which inserts a region based concept

with polynomials into a standard multilayer perceptron (MLP). HAP-CNN sought to fully cross the polynomial aspect into CNNs and losing the region concept as this will be inherently done with CNNs. The only other related work was Convolutional Polynomial Neural Network (CPNN) [12], also performed within the same lab. This only took a very specific look at facial features. This work also had disjoint polynomial layers, which is not the case for HAP-CNN.

Hierarchical Auto-associative Polynomial Convolutional Neural Network (HAP-CNN) seeks to add a new layer that is as modular as possible, making it very easy to implement in any CNN to a varying degree. This will be done in the convolutional and fully connected portions of the network. In theory, adding more non-linear weights will enable a chosen CNN to more accurately predict the desired class. Out of the non-linear functions, polynomials were chosen for two reasons. Firstly, previous work in Hierarchical Auto-associative Polynomial Network (HAPNet) [11] makes the results more comparable and demonstrated the polynomial layer's usefulness in a similar context. Secondly, their simplicity makes the layer easy to implement.

This work demonstrates a new method that does not require large datasets or a specific design. It also reveals the effectiveness of a the proposed layer, a locally connected polynomial layer, that is greatly effective at lower learning rates and with careful use can also be effective at a higher learning rate. The new algorithm is a modification of a standard Convolutional Neural Network (CNN), focusing on the effect that the polynomial layer when compared to the same network without the polynomial layers. The overall design of the network can be seen in Figure 1.

The effectiveness of the polynomial layers will be analyzed from two different perspectives. As previously mentioned, the network will look at the differences compared to the same neural network without the polynomial layer. Due to implementation, this is a polynomial of order 1. Additionally, the results will be compared to the work that was adapted for this purpose, HAPNet.

Theoretical Description of HAP-CNN

HAP-CNN Overview

Research in this realm comprises of many different elements. These include but are not limited to network design, fine tuning, polynomial type, and the many parameter settings of CNNs. Due to this issue, the scope of this work is limited to a single network design, shown in Figure 2, two datasets, two learning rates, and most standard settings. In addition, only one polynomial is utilized. Through use of the polynomial layer described in this section, five different designs are created, simply labeled as designs 1 - 5.

Mathematical Modification of Forward Pass

In attempts to fully describe the network while keeping a concise description, only equations that deal with the polynomial layer will be discussed. Within the convolutional portion, the proposed polynomial is applied after the input image or pooling layer through the equation:

$$f_{\rho 1}(x,y,z,n) = f_{\rho 0}(x,y,z)^n \tag{1}$$

for $n = 1,2,...,N$. This is implemented in practice through concatenation along the z dimension in a regular CNN. The forward pass continues as normal for the rest of each iteration, $\rho = 1,2,...,P$. This will naturally mean that the weight dimension is now different as well.

The fully connected layers are done in a similar fashion:

$$g_{\phi 1}(k,n) = g_{\phi 0}(k)^n \tag{2}$$

Likewise, this is done for each iteration, $\phi = 1,2,...,\Phi$.

Mathematical Modification of Backward Propagation

When analyzing the modifications in backward propagation, the polynomial layer will be seen as if it is the previous layer it modified. This is due to the fact that it has the same types of connections that would exist if there were no polynomial. The loss in the output layer is sent through the polynomial layer using the formula:

$$e_{\phi prepoly}(k) = \sum_{n=1}^{N} e_{\phi postpoly}(k,n) \tag{3}$$

where $e_{\phi prepoly}(k) \propto e_{\phi-1 postpoly}(k)$, as it is modified by activation and pooling functions. With this equation, the error is propagated backward using the formula:

$$e_{\phi}(k,n) = \sum_{l=1}^{L} e_{\phi}(l) w_{\phi}(l,k,n) \tag{4}$$

Where $e_{\phi}(k,n) = e_{\phi+1}(l)$. The weights for the layer preceding the polynomial are found by summing the locally connected weights that were modified in the polynomial layer.

$$w_{\phi}(l,k) = \sum_{n=1}^{N} w_{\phi}(l,k,n) \tag{5}$$

It is important to note that k will change size as n is being summed into the k dimension. This will also modify the equation finding the change in weight.

$$\Delta w_{\phi}(l,k,n) = \eta e_{\phi}(l) g_{\phi 2}(l) g_{\phi 1}(k,n) \tag{6}$$

The implementation of this equation is the same as any other CNN, with the preceding layer merely having more elements due to the polynomial that was added.

This process is continued for each polynomial and continuing as normal otherwise. Continuing to the beginning of the network, the convolutional layers are done in a similar fashion, accounting for the local connectivity of convolution and the extra dimension.

Results

Overview

In testing, two different datasets were used: MNIST and CIFAR10. MNIST [13] is a dataset comprising of handwritten digits, 0 to 9. It contains 60,000 training and 10,000 testing images of gray-scale 28 $\times$ 28 pixels. A sample of the data can be seen in Figure 8. The CIFAR10 dataset [14] contains 50,000 training and 10,000 testing images of objects such as dog, frog, airplane, and truck. Each image in this set is a 32 $\times$ 32 pixel RGB image. A sample of the CIFAR10 dataset can be seen in Figure 9.

Within this section, there are many tables detailing the results. Each bold value is the highest accuracy when the differing

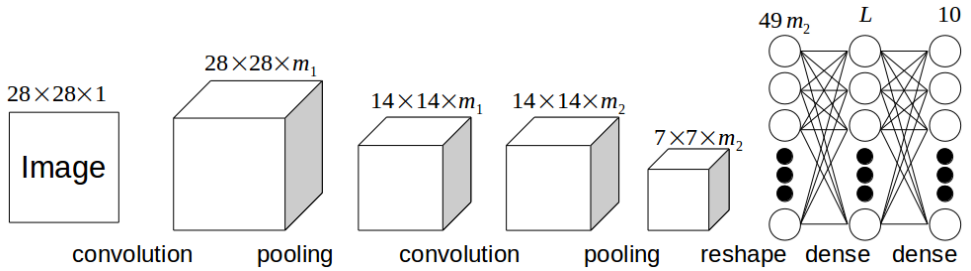

Figure 2. *Original CNN Design*

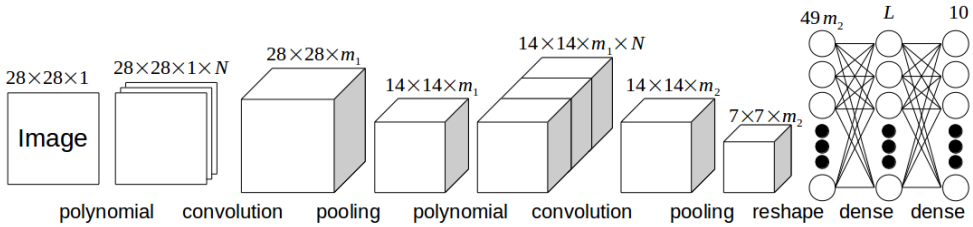

Figure 3. *HAP-CNN Design 1*

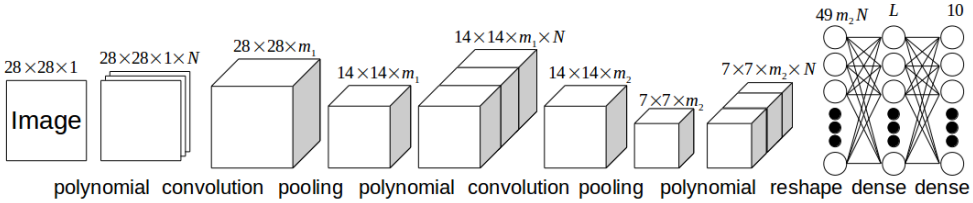

Figure 4. *HAP-CNN Design 2*

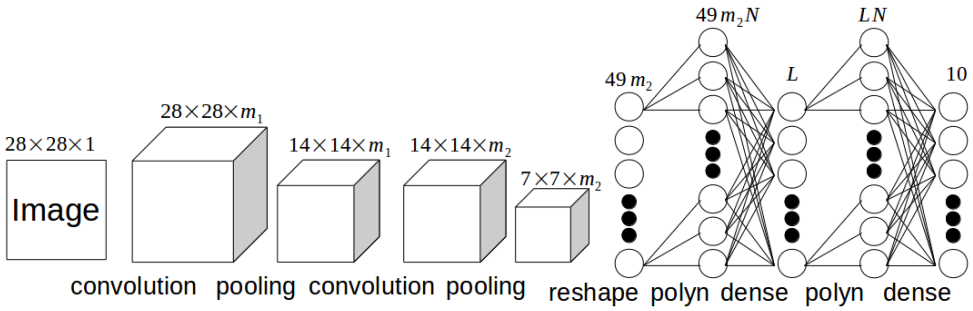

Figure 5. *HAP-CNN Design 3*

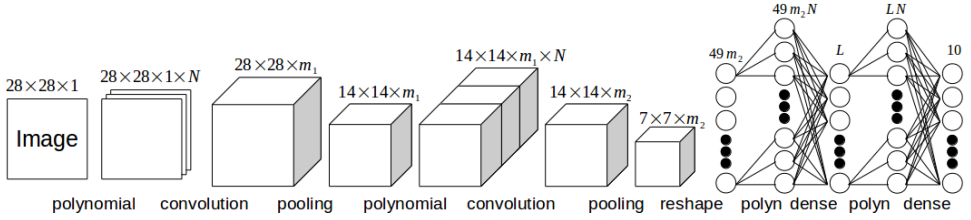

Figure 6. *HAP-CNN Design 4*

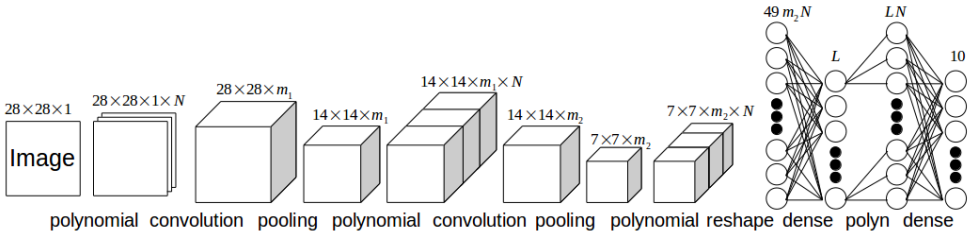

Figure 7. *HAP-CNN Design 5*

orders are compared, unless otherwise stated. It is important to reiterate that the first order is equivalent to a standard CNN design, shown in Figure 2

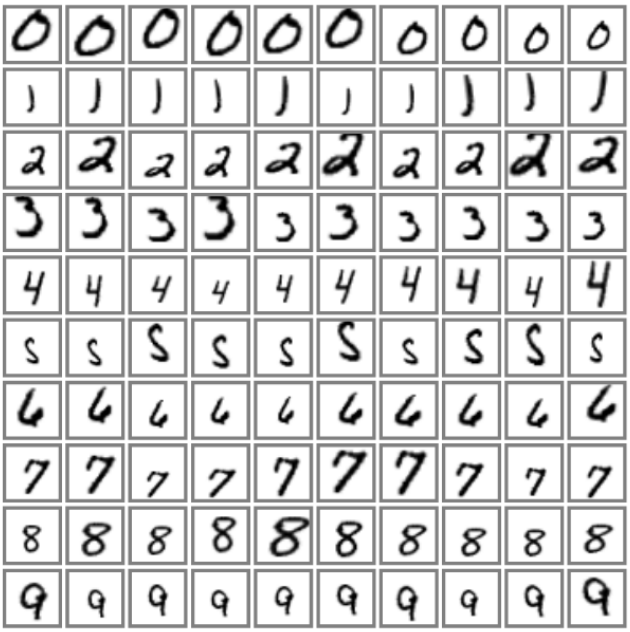

Figure 8. *MNIST Sample Images [13]*

MNIST

The first five tables each show the results of the different designs of HAP-CNN and how they each performed on the MNIST dataset. The polynomials ranges from orders 1 to 4, with learning rates (LR) of 0.01 and 0.001. These were analyzed both with and without a ReLU activation function. Other important parameters include dropout of 0.5, 3×3 weights of size 32 and 64, a stride of 1, and zero padding. Additionally, no data augmentation techniques were used.

Furthermore, no two trials used the same seeding. This creates a small amount of randomness in the results, but ensures that the overall results were not due to a specific seeding. As the first order will be identical for each HAP-CNN design, the amount of variability can be seen. For the MNIST dataset, in testing the random seeding caused the accuracy to vary between 96.16 and 96.31 percent.

Design 1 of HAP-CNN did not have an order that performed better overall. It is noticeable that any order greater than one caused an improvement.

Table 1: MNIST Results, Design 1

HAP-CNN 1	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	96.16	96.46	99.03	98.76
O2	97.81	97.99	**99.10**	98.86
O3	**98.30**	98.25	99.01	98.81
O4	98.14	**98.26**	98.93	**98.87**

Table 2: MNIST Results, Design 2

HAP-CNN 2	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	96.27	96.40	98.89	98.92
O2	98.20	98.68	**99.22**	**99.16**
O3	**98.88**	98.83	99.15	99.08
O4	98.86	**98.98**	99.20	99.12

Design 2, unsurprisingly has similar results to design 1. It is notable that the results seem to be slightly better than design 1 overall, accounting for the small variability. This seems to suggest that the extra polynomial lay was beneficial in the learning process.

Design 3 only utilizes polynomials in the fully connected portion. This makes it the most similar to HAPNet. It performed similarly to design 1, which contained the same number of polynomial layers in a different location.

Combining designs 1 and 3, design 4 outperformed the other two individual designs at the higher learning rates and underperformed when at the lower learning rate.

Design 5 seems to be an average of the other results. This seems to suggest that the effects of the polynomial layer are not directly additive. A new item in the tables is the result: X. This denotes that a test was not able to overcome divergence problems, which tend to be an issue at higher orders of polynomials. One method used to overcome this problem was running a few hundred images through the network at a lower learning rate to better initialize the weights before using the higher rate.

Table 3: MNIST Results, Design 3

HAP-CNN 3	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	96.25	96.31	98.81	98.82
O2	98.16	**98.30**	99.05	**99.13**
O3	**98.32**	97.81	**99.07**	99.06
O4	97.32	97.89	98.75	98.91

Table 4: MNIST Results, Design 4

HAP-CNN 4	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	96.21	96.30	98.83	98.90
O2	98.05	**98.10**	**99.12**	**99.20**
O3	**98.07**	97.88	98.98	99.03
O4	94.25	96.56	98.52	98.53

Table 5: MNIST Results, Design 5

HAP-CNN 5	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	96.31	96.33	99.03	98.85
O2	**98.55**	**98.65**	**99.16**	**99.20**
O3	98.46	98.37	99.11	98.89
O4	97.97	97.47	X	98.15

Looking at the MNIST results as a whole, there are a few important trends. It should be apparent that there is an improvement between the first order and second order. A closer analysis shows that the second order performs better overall, winning 12 of the 20 tests and the first order, original CNN, never had the highest classification accuracy. Additionally, the accuracy and learning rate were closely linked. It was demonstrated that even in the spread of an order of magnitude, $\eta = 0.001$ and 0.01 the polynomial layer improved results for the MNIST dataset.

When comparing to the state of the art networks, this design does not perform as well as the current record is 0.21 percent error. However many of these networks utilize many different methods not used here along with large networks and fine tuning methods. Additionally, human performance in this test is not considered to be at 100 percent, as some handwritten digits are incredibly ambiguous.

CIFAR10

To gain a fuller understanding of the usefulness of the network, a second dataset was used, CIFAR10, which yielded surprising results. Overall, the polynomial layers aided in the lower learning rates, but were still surpassed by the higher learning rates where polynomials had a detrimental effect, which will be analyzed in more detail at each test.

Design 1 continued the expected trend at the lower learning rates, with a jump in accuracy after the first order. One difference is the lower accuracy due to the difficulty of the problem. In a situation like this, a new network design would most likely aid in getting a higher accuracy. However, the ability to keep consistency is paramount and the design therefore remains the same.

Design 2 performs similarly, giving an even better accuracy. However, the network already begins to struggle with divergence at the higher learning rate.

In another surprising result, the polynomial layers in design 3 had a universal negative impact on accuracy, with the results becoming worse as the order increases. This demonstrates the partial usefulness within the convolutional portion and harm in the fully connected portion.

Designs 4 and 5 combine the results of the convolutional portion and fully connected portion. At the lower learning rates an increase in accuracy is seen before causing a negative impact. At

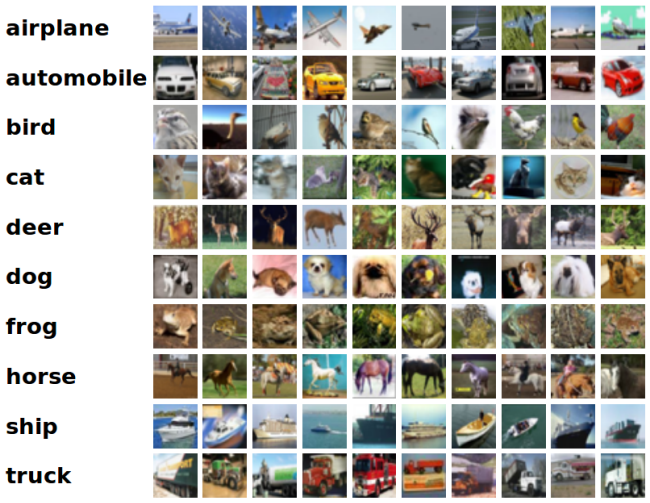

Figure 9. *CIFAR 10 Sample Images [14]*

Table 6: CIFAR10 Results, Design 1

HAP-CNN 1	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	50.59	49.41	**70.95**	**70.14**
O2	54.61	54.20	70.28	69.95
O3	56.30	56.57	70.65	68.98
O4	**56.44**	**57.13**	69.69	68.04

Table 7: CIFAR10 Results, Design 2

HAP-CNN 2	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	50.72	49.05	**70.51**	**69.75**
O2	58.50	55.93	69.31	69.36
O3	61.58	59.93	61.11	68.23
O4	**63.57**	**62.20**	X	68.44

Table 8: CIFAR10 Results, Design 3

HAP-CNN 3	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	**50.72**	**47.96**	**70.69**	**70.28**
O2	49.57	47.92	68.22	69.40
O3	45.14	39.52	61.19	60.83
O4	39.69	33.06	53.55	55.30

Table 9: CIFAR10 Results, Design 4

HAP-CNN 4	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	50.41	48.99	**71.51**	**69.75**
O2	**55.21**	**54.35**	69.94	69.71
O3	51.13	51.02	X	65.44
O4	46.54	X	X	X

Table 10: CIFAR10 Results, Design 5

HAP-CNN 5	LR0.001 None	LR0.001 ReLU	LR0.01 None	LR0.01 ReLU
O1	50.14	48.62	70.29	**70.97**
O2	**58.95**	**56.91**	**70.63**	69.19
O3	56.15	55.26	X	66.82
O4	X	X	X	X

Table 11: MNIST Results, HAP-CNN Design 3 (No Activation) versus HAPNet

	LR0.001 HAP-CNN	LR0.01 HAP-CNN	HAPNet
O1	96.25	98.81	98.12
O2	98.16	99.05	**98.47**
O3	**98.32**	**99.07**	97.87
O4	97.32	98.75	96.33

the higher learning rates, the second order overall causes only a very small loss in accuracy before massively hurting accuracy or diverging.

Comparison with HAPNet

As the research and findings from HAPNet were a large factor in the design choices of HAP-CNN, it is of interest to compare the results for the two designs. In comparing both networks at their highest accuracy and similar depths, HAP-CNN's error rate was 0.78, compared to HAPNet's 1.33 on the MNIST dataset. The closest possible design match is design 3 of HAP-CNN where the convolutional portion of the CNN acts replaces the region concept of HAPNet and then both contain essentially a multilayer perceptron with polynomial layers in-between. A comparison of these designs can be seen in Table 11. With the higher learning rate, it is apparent that HAP-CNN outperforms HAPNet, regardless of the polynomial order chosen.

Discussion

The networks, regardless of design seem to usually benefit from a low order polynomial. This is almost universally true for the MNIST dataset. When the CIFAR10 dataset was used, this was the trend for only the lower learning rate. The testing does not point to a singular polynomial order that should be used. This should not be a surprise, as CNNs often do not have a specific item that is universally beneficial. What can be demonstrated is that the polynomial layers in some cases are of a benefit to CNN designs.

Conclusions and Future Work

HAP-CNN adds to the varying tools that researchers can use when designing their own networks. Requiring no data modification techniques, or a specific design structure, this proposed layer can be inserted into any feed forward neural network. While the theory behind HAP-CNN enables any order polynomial, the second order is considered the default value. Furthermore, one of the most important aspects of this work is the ability to insert as many or few layers as desired. The work presented does not universally improve CNNs in every circumstance. However, in many situations provides better results. In future adaptations of this work, there are many places of interest. An analysis using more datasets should provide a fuller understanding of where polynomial layers are most effective. Additionally, other network designs, including transfer learning is of great interest.

Acknowledgments

A special thank you to Theus Aspiras, whose guidance was of incredible help.

References

[1] Alex Krizhevsky Ilya Sutskever, and Geoffrey Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Proc. Advances in Neural Information Processing Systems 25, pg. 1097 (2012).

[2] Karen Simonyan and Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, J. CoRR, abs/1409.1556 (2014).

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, Going Deeper with Convolutions, J. CoRR, abs/1409.4842 (2014).

[4] Nitish Srivastava, Geoffery Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, J. Journal of Machine Learning Research, (2014).

[5] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus, Regularization of Neural Networks using DropConnect, Proc. Proceedings of the 30th International Conference on Machine Learning, pg. 1058 (2013).

[6] Dominik Scherer, Andreas Muller, and Sven Behnke, Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition, Proc. 20th International Conference on Artificial Neural Networks (ICANN) (2010).

[7] Chen-Yu Lee, Patrick Gallagher, and Zhuowen Tu, Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree, J. Corr, abs/1509.08985 (2015).

[8] Andrej Karparthy, CS231n: Convoltuional Neural Networks for Visual Recognition, http://cs231n.github.io/ (2017)

[9] Kaiming He, Xiangyu Zhang , Shaoqing Ren, and Jian Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, J. CoRR, abs/1502.01852 (2015)

[10] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio, Maxout Networks, J. CoRR, abs/1302.4389 (2013)

[11] Theus Aspiras, Hierarchical Auto-associative Polynomial Network for Deep Learning of Complex Manifolds, University of Dayton (2015)

[12] Chen Cui, Convolutional Polynomial Neural Network for Improved Face Recognition, University of Dayton (2017).

[13] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, Gradient-Based Learning Applied to Document Recognition, J. Proceedings of the IEEE, 86, 11 (1998).

[14] Alex Krizhevsky, Learning Multiple Layers of Features from Tiny Images, University of Toronto (2009).

Author Biography

Patrick Martell received his BS in Electrical Engineering at Cedarville University and his MS in Electrical Engineering at the University of Dayton. His research has focused on machine learning, specifically convolutional neural networks.

Vijayan Asari is a Professor and Endowed Chair in electrical and computer engineering at the University of Dayton, Dayton, Ohio. He is the Director of the Vision Lab at UD. Dr. Asari received his PhD in Electrical Engineering from the Indian Institute of Technology, Madras in 1994. He holds three patents and has published more than 600 articles in the areas of image processing, computer vision, pattern recognition and machine learning. Dr. Asari is a member of IS&T and a senior member of SPIE and IEEE.