Contents

0.1 Introduction 2

1 Background 3
1.1 Neural Networks 3
1.2 Convolutions 4
1.3 Modern Training 5
1.4 Residual Networks 5

2 Related Works 7
2.1 Parameter Deletion 7
2.2 Specialized Architectures 7
2.3 Network Expansion 8

3 Methodology 9
3.1 Dynamic Network Capacity 9
3.2 Layer-Specific Analysis 9

4 Experiments 10
4.1 Implementation 10
4.2 Function Regression 10
4.3 MNIST Classifier 10
4.3.1 Repeated Runs 10

5 Discussion 11

6 Conclusion 12

0.1 Introduction

The study of

Chapter 1

Background

In this section, we provide a general introduction to the relevant basics of deep learning.

1.1 Neural Networks

The foundational principle of neural networks is, in its purest form, inspired by the biology of the human brain. The field of AI has often modelled new algorithms after biological phenomena; in this field, genetic algorithms are based on evolution and particle swarm optimization is based on social behaviors. The history of neural networks dates back to the beginnings of artificial intelligence research, and from those times a few fundamentals still remain.

Firstly, the structure of a basic feedforward network was established. In a general sense, a neural network is a directed graph, with neurons as nodes and weights as edges. Every neuron activates (outputs) with a strength that is a function that is the element-wise multiplication of the inputs with the edge weights. That is, if $w_{i,j}$ is the weight value between nodes i and j, n_i is the activation of node i, and N_j is the list of node indices that are connected to j, then node j will activate with strength

$$n_j = F\left(\sum_{i \in N_j} w_{i,j} n_i\right)$$

This definition relies on an activation function F, which allows the network to produce nonlinear behaviors. We can provide input into the neural network by activating a set of nodes with specific values, and we can similarly read output from any subset of nodes. A feedforward network is then any acyclic neural network. These networks are typically organized in layers of neurons, which indicate the depth of each node. In this model, layers are typically fully connected, meaning that all nodes in one layer are connected to all nodes of the next layer. This allows a computationally-efficient model of weights as a matrix M, taking input vector V to output vector MV.

Throughout modern literature, feedforward networks are an important but rarely examined component; the structure is often considered fixed and serves to provide a final classification. Key limitations to fully connected layers prevent them from being suitable for use as the sole structure of larger networks. For example, because of the fully connected nature of the layers, they require an immense amount of memory. Such a layer between two sets of just 10000 nodes would require 100 million parameters, while modern networks often have a total of 10 million parameters [4]. This extra capacity, while being inefficient, can also be bad for training in general; there is no sense of locality in such a layer, as every node is treated individually. This means that it is difficult and nearly impossible to train higher level features that should be treated equally across all areas of the input (which is of particular interest to problems like image classification).

The other key insight of neural networks is backpropagation, which is an algorithm to let errors accumulated from the output layer of the network propagate backwards through the network, training it in the process. As in the example above, if the network's output is O, but the correct response would be C, we can calculate the error

$E = O - C$. From this, we need a cost function that determines how errors are judged; a typical example may be the L_2 loss

$$\text{Cost}(O - C) = \sum_{0}^{n} ||O_i - C_i||^2$$

However, since we know that

$$O = F\left(\sum_{0}^{n} w_i a_i\right)$$

it is possible to figure out the influence each weight had on the error by taking the partial derivative of the cost function with respect to the weight,

$$\frac{\partial \text{ Cost}}{\partial a_i} =$$

Modern training methods are far more advanced, but still rely on the basic algorithm described here, which is often termed gradient descent. The main

LeCun et al.'s seminal work in this field, *Gradient-Based Learning Applied to Document Recognition* [11], provided the first basis of using backpropagation methodologies to train visual classifiers. Even more importantly, it introduced the fundamental structure of the modern visual deep learning network. In its usage of convolutions as a method for extracting high-level features out of larger images, it set the framework for a new style of network that would prove to be far more efficient and scalable.

1.2 Convolutions

A convolution is a operator applied to two functions f and g, which provides a way of interpreting one function in the context of the other. The operation is generally defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(r)g(t - r)dr$$

In the perspective of modern deep learning, we are primarily interested in its usage as a matrix operator; in this context, we limit the range of g to the size of the matrix s such that

$$(f * g)(t) = \int_{0}^{s} f(r)g(t - r)dr$$

In this context, we refer to g as the *convolutional kernel*. Using a convolutional kernel to preprocess the image proves to be critical to the performance of modern deep learning methods, as a small kernel can operate over a large image in parallel.

For example, we consider the basic edge-detecting matrix

$$E = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

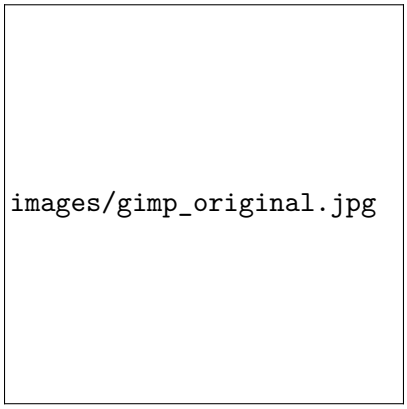
This convolution will perform the element-wise matrix multiplication of the kernel E with the immediate neighbors of each pixel, then aggregate the elements by summation. That is, if the pixel values around a specific pixel e are

$$P_e = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

then the convolution at that pixel will be

$$\begin{aligned} P_e * E &= 0a + 1b + 0c + 1d - 4e + 1f + 0g + 1h + 0i \\ &= (b + d + f + h) - 4e \end{aligned}$$

Accordingly, it will create a new matrix, with each element representing the convolutional kernel applied at that point. As shown above, the convolution $P_e * E$ will have the strongest activation when there is a strong difference between the pixel e and its neighbors (b, d, f, and h), thus performing a basic localized form of edge detection. Figure 1.1 shows this convolution applied to an arbitrary image.



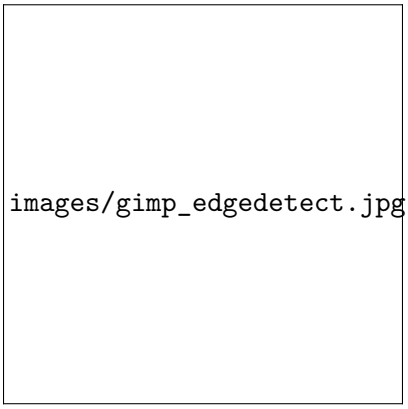


Figure 1.1: A demonstration of an edge-detecting convolution, from the GIMP User's Manual. [3]

A convolutional neural network is therefore the product of chaining together convolutions to perform efficient feature extraction with the standard feedforward neural network structure. LeCun's contribution to this structure was showing that the same backpropagation methods used to train other networks could also be applied to convolutional layers, allowing convolutional neural networks (CNNs) to learn their own feature extractors. This allows the CNN to determine what kinds of high-level feature extraction is necessary for the specific problem; applying this across a variety of filters demonstrates More importantly, this allows for networks to automatically chain convolutional layers, in which the initial information can pass through multiple layers of feature extraction, which are all automatically deteremind from the training data.

1.3 Modern Training

1.4 Residual Networks

As network architectures have changed over time, an ongoing goal has been to develop truly deep architectures. Even as better training algorithms have allowed network depth to increase to tens of layers, it is a generally-held principle that depth, not width, is crucial for allowing a network to learn complex features. At the same time, gradient-descent methods work poorly in networks with significant depth, as the gradient term (the partial derivative of the loss function with respect to the weight) decreases significantly by each layer. This means that after a certain amount of depth in the network, the gradient is so small that it is nearly entirely noise. The issue of disappearing gradient is in some form mitigated by training methods, some of which are more heavily biased towards the sign of the gradient rather than the magnitude. However, regardless of the specific algorithm, gradients are effectively unusable for networks with significant depth for typical network architectures.

To solve this problem, He et al. [6] developed Residual Networks. The insight in this work is that typical network layers are effectively performing two tasks simultaneously—the transfer of state alongside feature extraction/classification. The former requirement necessitates that the layer learn an encoding of its input, which is

inefficient. He et al. rewrite the typical neural network architecture to allow it to merely learn the latter task, and apply this as a residual (or equivalently, difference) to the inputs. That is, if a typical neural network layer takes input X, it will apply the layer L to produce $L(X)$. A residual network layer takes the input, then outputs the layer's contribution in summation with the original input to produce $L(X) + X$. Beyond the theoretical improvements to the "task" of the layer, it is also crucially important that this identity mapping for X in a residual layer allows the gradient to propagate backwards with its original magnitude. This ensures that the gradient is present at every layer with reasonable strength, allowing the calculations for error on the specific layer operation L to be done with less noise. He et al. used this structure to produce a 152-layer network, which is almost an order of magnitude increase over previous methods. This result was enough to win ILSVRC in 2015, an industry-standard annual image classification competition, demonstrating the efficacy of the algorithm.

With regards to this thesis, residual networks have a very important property that a layer which is entirely zero (where $L = 0$) results in a layer that simply produces the identity. This means that it is easier to insert and remove residual network layers than in typical architectures. Along these lines, Huang et al. [8] introduce Stochastic Depth, which randomly drops layers during the training phase as a form of regularization and ensuring that every layer learns something different. This is very similar to the usage of Dropout in training, except entire layers are dropped.

Further expanding on He et al.'s work, Zaguruyko and Komodakis [16] assert that residual networks are equally suited to creating wide networks as they are for deep ones; their testing indicates that it is possible to use the residual network framework effectively for networks of comparatively shallow depth (16 layers). This is of particular interest because it indicates the difficulty of determining optimal network architectures; the benefits of wide residual networks are dependent on the specific classification problem, and the

Chapter 2

Related Works

In this work, we are primarily interested in optimizing neural network architectures and other hyperparameters. Towards this end, we investigate current findings in the literature. To the best of our knowledge, neural network architecture self-optimization is a very new topic, and many results are preliminary or perhaps somewhat incomplete. Nevertheless, they provide an important glimpse into the contemporary research space, and are highly motivational to the specific topic of this thesis. The aim of this section is to cover some of the existing work that specifically focuses on network optimization, and to provide some grounding for our contributions.

2.1 Parameter Deletion

Ever since neural networks have been developed, experts have wondered how to make them more efficient. The fixed initial structure required to train a network is one that is inherently overparametrized, because the minimum number of parameters needed is not known ahead of time. Making the problem worse, neural network training is often slow and requires significant computational power, limiting the ability to test out differing numbers of parameters. The natural solution is, therefore, to train an oversized network, and to somehow whittle it down to size. Initial practices were based on heuristic deletion; that is, algorithms that deleted all weights w where $w < p$ for some low-pass filter p. These methods generally result in a sparse network (where the network has missing connections), which are difficult to represent and operate on efficiently.

LeCun et al.'s early work from 1989, *Optimal Brain Damage* [12], showed that these heuristic-based methods were inefficient and could irreparably destroy a network. He proposed a method based on error gradients that could more accurately find weights that contribute By what is effectively the butterfly effect, the deletion of a weight with small magnitude could actually prove to have a significant impact on the network.

This was taken a step further by Hassibi et al. [5] in their followup work, *Optimal Brain Surgeon*. By analyzing the Hessian matrix of the network, Hassibi et al.'s algorithm is among the most detailed methods shown to delete weights from a network, and they show that their algorithm is in fact optimal for specific small networks. However, the calculation of the Hessian is an $O(n^2)$ operation in both space and time, making it largely unsuitable for networks in the modern age, where n (the number of parameters) is often in the millions or tens of millions. At the same time,

2.2 Specialized Architectures

Another key direction taken by researchers is to design the network specifically to minimize

Google's Inception network [14], developed by Szegedy et al., has gone through various iterations, which all involve complex pooling of different convolutional kernel sizes. In their 2016 update to the architecture [15], they focus on tuning the inefficiently large filter sizes used in the previous revision. They note that a 5×5 convolution is effectively the same (covers the same area) as two 3×3 convolutions while requiring more parameters (25 versus

$9 \cdot 2 = 18$), dubbing this reduction as filter factorization. In the same vein, it is possible to reduce a 3×3 convolution to a 3×1 convolution followed by a 1×3 convolution, which requires a third less parameters.

There are various benefits to an increased number of smaller layers beyond parameter reduction. Firstly, it allows the increased application of nonlinear activation functions, which are generally regarded as critical for learning complex problems. Secondly,

2.3 Network Expansion

Chapter 3

Methodology

In this section, we provide a description of the kinds

3.1 Dynamic Network Capacity

3.2 Layer-Specific Analysis

Chapter 4

Experiments

4.1 Implementation

4.2 Function Regression

4.3 MNIST Classifier

4.3.1 Repeated Runs

Chapter 5

Discussion

Chapter 6

Conclusion

Bibliography

[1] Changpinyo, S., Sandler, M., and Zhmoginov, A. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257* (2017).

[2] Courbariaux, M., and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[3] GIMP. Convolution matrix. `https://docs.gimp.org/en/plug-in-convmatrix.html`.

[4] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems* (2015), pp. 1135–1143.

[5] Hassibi, B., Stork, D. G., et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems* (1993), 164–164.

[6] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[7] Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[8] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European Conference on Computer Vision* (2016), Springer, pp. 646–661.

[9] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 1mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[10] Lebedev, V., and Lempitsky, V. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2554–2564.

[11] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[12] LeCun, Y., Denker, J. S., Solla, S. A., Howard, R. E., and Jackel, L. D. Optimal brain damage. In *NIPs* (1989), vol. 2, pp. 598–605.

[13] Murray, K., and Chiang, D. Auto-sizing neural networks: With applications to n-gram language models. *arXiv preprint arXiv:1508.05051* (2015).

[14] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1–9.

[15] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2818–2826.

[16] Zagoruyko, S., and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).