Rochester Institute of Technology

RIT Scholar Works

Theses

12-2016

Hierarchical Decomposition of Large Deep Networks

Sumanth Chennupati
cv7148@rit.edu

Follow this and additional works at: <https://scholarworks.rit.edu/theses>

Recommended Citation

Chennupati, Sumanth, "Hierarchical Decomposition of Large Deep Networks" (2016). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

Hierarchical Decomposition of Large Deep Networks

by

Sumanth Chennupati

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Dr. Raymond Ptucha
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
Dec 2016

Approved By:

Dr. Raymond Ptucha
Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Christopher Kanan
Secondary Advisor – R.I.T. Dept. of Imaging Science

Dr. Dhireesha Kudithipudi
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dedication

“Things are never quiet as scary when you are by my side. Dedicated to my family and friends.”

Acknowledgements

“I sincerely thank Dr. Raymond Ptucha for his advice, patience and faith. I would also like to thank my committee Dr.Christopher Kanan and Dr. Dhireesha Kudithipudi for their valuable feedback. I also thank Dr. Shanchieh Yang and Dr. Andreas Savakis. Thanks to Computer Engineering faculty, staff and also, to others who offered to help.”

Abstract

Teaching computers how to recognize people and objects from visual cues in images and videos is an interesting challenge. The computer vision and pattern recognition communities have already demonstrated the ability of intelligent algorithms to detect and classify objects in difficult conditions such as pose, occlusions and image fidelity. Recent deep learning approaches in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) are built using very large and deep convolution neural network architectures. In 2015, such architectures outperformed human performance (94.9% human vs 95.06% machine) for top-5 validation accuracies on the ImageNet dataset, and earlier this year deep learning approaches demonstrated a remarkable 96.43% accuracy. These successes have been made possible by deep architectures such as VGG, GoogLeNet, and most recently by deep residual models with as many as 152 weight layers. Training of these deep models is a difficult task due to compute intensive learning of millions of parameters. Due to the inevitability of these parameters, very small filters of size 3x3 are used in convolutional layers to reduce the parameters in very deep networks. On the other hand, deep networks generalize well on other datasets and outperform complex datasets with less features or Images.

This thesis proposes a robust approach for large scale visual recognition by introducing a framework that automatically analyses the similarity between different classes among the dataset and configures a family of smaller networks that replace a single larger network. Classes that are similar are grouped together and are learnt by a smaller network. This allows one to divide and conquer the large classification problem by identifying the class category from its coarse label to

its fine label, deploying two or more stages of networks. In this way the proposed framework learns the natural hierarchy and effectively uses it for the classification problem. A comprehensive analysis of the proposed methods show that hierarchical models outperform traditional models in terms of accuracy, reduced computations and attribute to expanding the ability to learn large scale visual information effectively.

Table of Contents

Dedication	ii
Acknowledgements	iii
Abstract	iv
Table of Contents	vi
List of Figures	x
List of Tables	xiii
Chapter 1 Introduction	1
1.1. Motivation	1
1.2. Contributions	4
Chapter 2 Background.....	5
2.1. Convolutional Neural Networks	5
2.1.1 Convolution layer.....	6
2.1.2 Pooling	9
2.1.3 Building Block	10
2.1.4 Fully-Connected Layer.....	10
2.2. Activation Functions.....	11
2.2.1 Softmax	13
2.3. Loss Functions.....	13

2.3.1	Squared Loss	14
2.3.2	Absolute Loss	14
2.3.3	Hinge Loss.....	15
2.3.4	Negative Log Likelihood Loss	16
2.4.	Optimization	16
2.4.1	Gradient Descent	17
2.4.2	Stochastic Gradient Descent.....	18
2.4.3	Batch Gradient Descent.....	18
2.4.4	Mini-batch Gradient Descent	19
2.4.5	Momentum	19
2.5.	Regularization.....	21
2.5.1	L1 Regularization.....	22
2.5.2	L2 Regularization.....	23
2.5.3	Max Norm constraints	23
2.5.4	Dropout.....	24
2.5.5	Data Augmentation	25
2.6.	Back Propagation.....	25
Chapter 3	Deep Learning Models	27
3.1.	AlexNet.....	28

3.2.	VGG.....	30
Chapter 4	Datasets.....	33
4.1.	CalTech 101 & 256.....	33
4.2.	CIFAR-100	34
4.3.	ImageNet: ILSVRC-2012.....	35
Chapter 5	Hierarchy Models	36
5.1.	Hierarchy Clustering.....	37
5.2.	Hierarchy Classifier	40
5.3.	Class Assignment Classifier	41
5.4.	Adaptive Hierarchical Models.....	43
5.4.1	Transfer Learning.....	44
5.4.2	Joint Model.....	45
Chapter 6	Experiments & Results	46
6.1.	Hierarchical Model Experiments.....	46
6.1.1	CalTech 101 Results.....	47
6.1.2	CalTech 256 Results.....	51
6.1.3	CIFAR-100 Results	54
6.2.	Spectral Clustering	57
6.3.	Adaptive Hierarchical Models.....	58

6.3.1	CalTech 101 Results.....	61
6.3.2	CalTech 256 Results.....	62
6.3.3	CIFAR 100 Results	62
6.3.4	ImageNet Results	64
6.4.	Discussion.....	67
Chapter 7	Conclusion	69
	Bibliography	70

List of Figures

Fig 1. Example of a convolutional neural network	6
Fig 2. Convolution operation over the height and width of an image.....	6
Fig 3. Convolution output volume $3 \times 3 \times 2$ obtained from an image volume of $5 \times 5 \times 3 (+1 \text{ pad})$ when convolved with two filters w_1 and w_2 of size $3 \times 3 \times 3$. [16].....	7
Fig 4. Demonstrating output (right) of a convolution operation on an image (left).....	8
Fig 5. Illustration of a pooling operation on a 2-d convolution map sampled to a lower size.	9
Fig 6. Building block of convolutional neural network containing a convolution layer followed by pooling layer [17].	10
Fig 7. Output of the forward propagation showing cnn architecture and outputs [16].	11
Fig 8. Activation functions (in order of a,b,c) sigmoid, tanh and relu.	12
Fig 9. Plot of a square loss function.....	14
Fig 10. Plot of an absolute loss function.....	15
Fig 11. Plot of hinge loss function.	15
Fig 12. Illustration of gradient descent algorithm where steps are taken from x_0 to x_4 in the direction of negative of gradients at each position. Here x_0 to x_4 are data points on the contour plot of a loss function whose minimum is at the center.	17
Fig 13. Illustration of a situation where a network stuck at some local minima.[19]	20
Fig 14. (a) & (b) are illustrations of how momentum help moving over local minimum. [20]	20
Fig 15. Illustration of properly fitted/well generalized data vs overfitted data. [21]	21
Fig 16. Illustration of cost function during over fitting, where training cost is very less compared to validation error. [21]	22
Fig 17. Illustration of dropped connections. [22]	24
Fig 18. Algorithm 1 explains the back propagation.....	26

Fig 19. Shows the progress of classification performance (top-5 error %) on imagenet dataset over past few years. Dashed line indicates human performance (5.1%).	27
Fig 20. Illustration of alexnet cnn configuration.	29
Fig 21. Average of images of 100 different classes in caltech 101 dataset.	33
Fig 22. Collage of images from different object categories in imagenet:ilsvrc2102 dataset [1].	35
Fig 23. Algorithm 2, hierarchy class clustering algorithm.	38
Fig 24. Illustration of hierarchy clustering on toy data having 11 classes. (a) shows a dendrogram with dissimilarity among the classes. (b) shows the 3 non-overlapping clusters formed with similar classes grouped together (left) and the overlapping cluster (right).	39
Fig 25. Illustration of hierarchical deep network framework.	40
Fig 26. Flow of classification using hierarchal deep networks.	41
Fig 27. Illustration of hierarchical models.	42
Fig 28. Illustration of network selection manager that selects the best network configuration.	43
Fig 29. Illustration of adaptive hierarchical model.	44
Fig 30. Illustration of network selection manager that chooses between a convolutional neural network or a multi-layer perceptron neural network.	45
Fig 31. Illustration of hierarchical model.	46
Fig 32. Example of a dendrogram with 102 classes of caltech101 dataset generated using confusion matrix obtained from a single cnn (better viewed in color).	48
Fig 33. Example of a dendrogram with 257 classes of caltech257 dataset generated using confusion matrix obtained from a single cnn (better viewed in color).	52
Fig 34. Example of a dendrogram with 100 classes of cifar-100 dataset generated using confusion matrix obtained from a single cnn (better viewed in color).	55
Fig 35. Illustration of hierarchical model generated using spectral clustering.	57
Fig 36. Example of a dendrogram with 100 classes of imagenet 100 dataset generated using confusion matrix obtained from a single cnn (better viewed in color).	65

Fig 37. Example of a dendrogram with 200 classes of imagenet 200 dataset generated using confusion matrix obtained from a single cnn (better viewed in color)	66
Fig 38. Example of a dendrogram with 1000 classes of imagenet dataset generated using confusion matrix obtained from a single cnn	67

List of Tables

Table 1. Comparison of performance of different regularization techniques on mnist data.	24
Table 2. Alexnet configuration [5].	29
Table 3. Case study on alexnet parameters & memory [16].	30
Table 4. Parameters of various vgg net configurations.	31
Table 5. Multiple vgg net configurations [6].	31
Table 6. Case study on vgg-d model.	32
Table 7. Shows list of superclasses and classes in cifar-100 dataset [38].	34
Table 8. Shows performance of hierarchical model using an mlp on caltech101 dataset.	47
Table 9. Estimation of memory and computations in a single mlp on caltech101 dataset during testing of an image.	48
Table 10. Estimation of memory and computations in a hierarchy classifier on caltech101 dataset during testing of an image.	49
Table 11. Estimation of worst case memory and computations in a class assignment classifier on caltech101 dataset during testing of an image.	49
Table 12. Comparison of worst case memory and computations between single mlp and hierarchical model on caltech101 dataset during testing of an image.	49
Table 13. Shows performance of hierarchical model using a cnn on caltech101 dataset.	50
Table 14. Shows performance of hierarchical model using an mlp on caltech256 dataset.	51
Table 15. Shows performance of hierarchical model using a cnn on caltech256 dataset.	51
Table 16. Estimation of memory and computations in a single mlp on caltech256 dataset during testing of an image.	52
Table 17. Estimation of memory and computations in a hierarchy classifier on caltech256 dataset during testing of an image.	53
Table 18. Estimation of worst case memory and computations in a class assignment classifier on caltech256 dataset during testing of an image.	53

Table 19. Comparison of worst case memory and computations between single mlp and hierarchical model on caltech256 dataset during testing of an image.....	53
Table 20. Shows performance of hierarchical model using an mlp on cifar-100 dataset.	54
Table 21. Shows performance of hierarchical model using a cnn on cifar-100 dataset.....	54
Table 22. Estimation of memory and computations in a single mlp on cifar100 dataset during testing of an image. .	55
Table 23. Estimation of memory and computations in a hierarchy classifier on cifar-100 dataset during testing of an image.....	56
Table 24. Estimation of worst case memory and computations in a class assignment classifier on cifar-100 dataset during testing of an image.	56
Table 25. Comparison of worst case memory and computations between single mlp and hierarchical model on cifar-100 dataset during testing of an image.	56
Table 26. Shows performance of hierarchical model using an mlp on caltech101 dataset.....	58
Table 27. Pre-defined convolutional neural network configurations for caltech101, caltech256.....	59
Table 28. Pre-trained configurations for imagenet datasets.....	60
Table 29. Comparison of performance between different models on caltech 101 dataset.....	61
Table 30. Comparison of performance between different models on caltech 256 dataset.....	62
Table 31. Comparison of performance between different models on cifar 100 dataset.....	63
Table 32. The performance of a single cnn vs. Hierarchical models generated from hierarchical clustering and cifar taxonomy.....	64
Table 33. Comparison of performance between different models on imagenet 100 dataset.	64
Table 34. Comparison of performance between different models on imagenet 200 dataset.	65
Table 35. Comparison of performance between different models on imagenet dataset.	66

Chapter 1 Introduction

Most people can easily identify objects. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing 140 million neurons, with tens of billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing. We carry in our heads a marvel of the universe, tuned by evolution over hundreds of millions of years. Recognizing objects isn't easy. Rather, we humans are stupendously, astoundingly good at making sense of what our eyes show us.

Teaching computers how to perform such object recognition is a challenging task. Understanding the visual information from ImageNet [1] (a dataset with more than 14 million images and 21,841 object categories) to perform visual recognition is a very difficult task. It is more difficult to train a fast and efficient classifier for such tasks. Convolutional Neural Networks (CNNs) offer solutions to such highly complex object recognition challenges. They have already revolutionized the fields of computer vision and pattern recognition. With respect to object detection, CNNs have demonstrated extraordinary performance on the 1000-class ImageNet [2] dataset, recently surpassing human-level performance [3],[4]. In this thesis, a hierarchical framework is proposed to scale up the existing deep learning frameworks to perform large-scale visual recognitions tasks more efficiently.

1.1. Motivation

In past few years with the advances of deep learning mainly due to convolutional neural networks, performance on large scale visual recognition challenge has been improved significantly. Deep architectures [5],[6],[7] with hierarchical frameworks enable the representation of complex concepts with fewer nodes than shallow architectures. With regard

to object classification, these networks have recently been shown to equal the performance of neurons in the primate inferior temporal cortex [8], even under difficult conditions such as pose, scale, and occlusions.

AlexNet [5] a large, deep convolutional neural network, is trained on 1.2 million high-resolution images and 1000 classes in the ILSVRC 2010. It achieved 63.5% and 83% top-1 and top-5 accuracies respectively, and has 60 million parameters. VGG [6] is a very deep convolutional neural network with 16-19 layers and 144 million parameters. VGG outperformed AlexNet by achieving 74.5% and 92.66% top-1 and top-5 accuracies respectively. However, GoogLeNet [7] achieved better accuracy (top-5 accuracy 93.33%) compared to other architectures and it has $12\times$ fewer parameters compared to [5]. It has been shown that network depth is more important than the number of nodes in each layer [9], with modern architectures [6],[7] containing more than 20 layers[6], requiring the solution of over 100M parameters. As the classification task becomes more difficult, the number of parameters increases exponentially.

As datasets become larger, a natural question to ask is if conventional CNNs can keep up with the 50+K classes thought to be perceptible by a typical human. Although CNNs have been trained for upwards of 10K classes, the number of weights in the fully connected layers grow exponentially, demanding a daunting number of training samples, and consuming huge computational resources. The CNNs [5], [6] and [7] are built to classify 1000 classes in ILSVRC. Their parameters explode when scaled to classify 21K classes or even more. In fact, these architectures use a single large network to classify the large problem. There is a significant need to improve existing frameworks by increasing parallelism and reducing number of computation/parameters in order to build efficient classifiers which can be extended to classify virtually unlimited number of classes.

Techniques such as domain adaptation [10] and hashing [11] have been used to tackle problems with larger classes. Yan et al. [12] leveraged the hierarchical structure of categories by embedding CNNs into category hierarchy. In this thesis, a multi-layer hierarchical framework is introduced where an abstract higher level network initially determines which subnetwork a sample should be directed to. Lower level networks take on the task of finding discriminating features amongst similar classes in the subnetwork. Each sub-network is called a class assignment classifier. Outputs from these class assignment classifiers feed a probabilistic classifier to predict the final class for a test sample.

The size and type of architecture of a deep network can have a profound impact on both network accuracy and training resources. The subnetworks generated in hierarchical frameworks exhibit different properties. These subnetworks have different statistical characteristics and levels of confusion. Considering the importance of appropriate network configuration for classification tasks, an adaptive network selection approach would be required to optimize the family of low level networks or class assignment classifiers. Adaptive network configurations should examine the confusion among the classes and decide an appropriate network configuration to optimize these individually.

1.2. Contributions

1. A framework that automatically analyzes and configures a family of smaller deep networks as a replacement to a singular, larger network.
2. Resulting smaller networks are not only highly scalable, parallel and more practical to train, but also achieve higher classification accuracy.
3. A method to adaptively select the configuration of the hierarchical family of classifiers.
4. Demonstrate the advantages of using transfer learning as an initialization for the subnetworks.

Chapter 2 Background

From the early work of Hubel and Wiesel [13], it has been learnt that the visual cortex is made up of a complex arrangement of cells that are sensitive to sub regions of the visual field called receptive fields. Receptive fields act as local filters over the input visual information and are combined to form complex cells in visual fields V1 through V4 to exploit the spatial correlation present in the visual space. Simple cells have maximum response to the minimal features in receptive field such as edges whereas complex cells with large receptive fields respond to high level features that are more abstract representations of low level features. These responses are further processed in other parts of brain like the inferior temporal cortex (IT cortex) where the object representation is most appropriate for the tasks like object recognition, localization, image and video understanding etc. Being the most efficient visual processing system it has inspired major visual models as early in 1980 by Kuniyiko Fukushima [14] and later LeNet [15] by Yann LeCunn in 1998.

2.1. Convolutional Neural Networks

Convolutional neural networks are a variant of feed-forward artificial neural networks that are functionally inspired from the visual mechanism observed in living organisms. In a convolutional neural network individual neurons are tiled in order to provide response similar to visual field.

A convolutional neural network consists of one or more of convolutional layers, optionally followed by down sampling and then by one or more fully connected layers as in a feed-forward artificial neural networks. Fig 1. Shows an example of convolutional neural network consisting of an Input Image followed by two convolutions and pooling layers, fully connected layer connecting to the output.

Fig 1. Example of a convolutional neural network

2.1.1 Convolution layer

The convolution layer is the core building block of convolutional neural networks and its output is the convolution response of the given input with respect to a spatial filter. In fact, these filters are learnable parameters which respond to a specific pattern observed at some spatial position in the input and extend to the depth of the input volume. Fig. 2. shows an instance of convolution operation over the height and depth of an image.

Fig 2. Convolution Operation over the height and width of an Image.

In a typical convolutional layer several filters are applied or convolved over the depth of the input volume and their responses are collected as an output volume which feed the later stages of the network. By convolving each filter over the width and height of the input

volume, a spatial response is obtained. Stacking these spatial responses of all the filters together form the output volume. Fig. 3. shows convolution output volume $3 \times 3 \times 2$ obtained from an image volume of $5 \times 5 \times 3$ when convolved with two filters W1 and W2 of size $3 \times 3 \times 3$.

Fig 3. Convolution output volume $3 \times 3 \times 2$ obtained from an image volume of $5 \times 5 \times 3 (+1 \text{ pad})$ when convolved with two filters W1 and W2 of size $3 \times 3 \times 3$. [16]

The convolution output y ($y \in \mathbb{R}^{H_{out} \times W_{out} \times D_{out}}$) of an input volume x ($x \in \mathbb{R}^{H_{in} \times W_{in} \times D_{in}}$) with bank of N -multidimensional filters f ($f \in \mathbb{R}^{H_f \times W_f \times D_f \times N}$) and biases b is given by 2.1.1.1

$$\begin{aligned}
 y_{h_{out}, w_{out}, d_{out}} &= b_N \\
 &+ \sum_{h_f=1}^{H_f} \sum_{w_f=1}^{W_f} \sum_{d_f=1}^{D_f} f_{h_f, w_f, d_f} \\
 &\times x_{h_{out}+h_f-1, w_{out}+w_f-1, d_{out}+d_f}
 \end{aligned} \tag{2.1.1.1}$$

In practice, convolution operations are performed with zero padding and selected stride value. The boundaries of the input volume are appended with zeroes of fixed number of

rows and columns on height and width through the depth of image. This operation is done to preserve the image dimension after the convolution layer. Stride is another hyper parameter which determines the step size over the height and width of the input image. Stride value is selected such that the dimensions of output volume are integral values.

Output volume of a convolution layer $(H_{out}, W_{out}, D_{out})$ for a given input volume (H_{in}, W_{in}, D_{in}) is determined by (2.1.1.2) , (2.1.1.3), (2.1.1.4). Let us assume that corresponding hyper parameters filter size, depth, pad and stride are denoted as (H_f, W_f) , N , S and P respectively.

$$H_{out} = \frac{H_{in} - H_f + 2 \times P}{S} + 1 \quad (2.1.1.2)$$

$$W_{out} = \frac{W_{in} - W_f + 2 \times P}{S} + 1 \quad (2.1.1.3)$$

$$D_{out} = N \quad (2.1.1.4)$$

Original Image

After Convolution

Fig 4. Demonstrating output (Right) of a convolution operation on an Image (Left).

2.1.2 Pooling

After convolutional layer, there may be a pooling layer where the output volume of the convolution layer is subsampled/down sampled. The pooling layer takes small rectangular block from the convolutional layer and subsamples it to produce a single output. It can be done in several ways like max-pooling, average pooling and L2 pooling etc.

Let the size of pooling block be (H_{pf}, W_{pf}) and stride be S . The output of pooling layer $(H_{pool}, W_{pool}, D_{pool})$ for a convolution layer's output volume $(H_{out}, W_{out}, D_{out})$ is given by (2.1.2.1), (2.1.2.2) and (2.1.2.3).

$$H_{pool} = \frac{H_{out} - H_{pf}}{S} + 1 \quad (2.1.2.1)$$

$$W_{pool} = \frac{W_{out} - W_{pf}}{S} + 1 \quad (2.1.2.2)$$

$$D_{pool} = D_{out} \quad (2.1.2.3)$$

Fig 5. Illustration of a pooling operation on a 2-d convolution map sampled to a lower size.

2.1.3 Building Block

Fig 6. Building block of convolutional neural network containing a convolution layer followed by pooling layer [17].

Fig. 6. shows the building block of a convolutional neural network. It contains a convolutional neural layer followed by an optional activation function and a pooling layer. This block is repeated over several layers changing the hyper parameters like filter size, pad size, stride etc.

2.1.4 Fully-Connected Layer

A fully-connected layer implements same function as a regular neural network. The output from the previous stage convolutional layers is flattened to a single vector and connected to few layers of a neural network.

Fig 7. Output of the forward propagation showing CNN architecture and outputs [16].

The output of fully connected layer is the final prediction of the convolutional neural network.

2.2. Activation Functions

Convolutional layer or pooling layers are followed by an activation function to increase the non-linear properties of the output of the overall network without effecting the filtered responses of the convolutional layer. The most common activation functions are sigmoid, tanh, and Rectified Linear Unit (ReLU). Sigmoid activation function has a non-linearity defined by:

$$f(x) = \frac{1}{(1 + e^{-x})} \quad (2.2.1)$$

Sigmoid activation function limits the output between 0 and 1. The non-linearity of tanh is defined as shown in (2.2.2) and limits the output between -1 and 1:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.2.2)$$

The Rectified Linear Unit computes the function and thresholds at zero:

$$f(x) = \max(0, x) \quad (2.2.3)$$

Sigmoid

(a)

Tanh

(b)

ReLU

(c)

Fig 8. Activation functions (in order of a,b,c) Sigmoid, Tanh and ReLU.

2.2.1 Softmax

Softmax is very useful function predominantly used in the output layer of artificial neural network or support vector machines. Softmax is widely used because it converts the values obtained after several activations/raw value into posterior probability that is essential in the classification task. This provides the measure of certainty. The Softmax function for a given input x_i is given by:

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^N e^{x_i}} \quad \forall i \in 1, 2, \dots, N. \quad (2.2.1.1)$$

N represents the number of units in the output layer, like number of neurons in the output layer of a neural network.

2.3. Loss Functions

Let a function (a model/hypothesis) $f(x)$ predicts value of y for the given data x such that $y = f(x)$. Let $t \in Y$ be the ground truth/true label of input x . Then a loss function $l : Y \times Y \rightarrow \mathbb{R}_0^+$ evaluates $f(x)$. If the prediction y is equivalent to corresponding t , then the loss would be 0 making the prediction accurate. In this case,

$$l(t, y) = 0 \quad \because t = y. \quad (2.3.1)$$

If not (2.3.2), the loss function would be some positive value which measures how bad the hypothesis is. Thus loss function can be understood as error measure or how deviated is the prediction from the true labels. For simple classification tasks, fundamental loss function is 0-1 loss, i.e.,

$$l(t, y) = \begin{cases} 0, & \text{if } t = y \\ 1, & \text{otherwise} \end{cases} \quad (2.3.2)$$

There are several other measures of the error/loss and they are further discussed in later sections.

2.3.1 Squared Loss

Squared loss for given input x , hypothesis $y = f(x)$ and the true label t , is defined by loss function in 2.3.1.1:

$$l(t, y) = (t - y)^2. \quad (2.3.1.1)$$

Fig 9. Plot of a square loss function.

Squared loss is well applicable for regression problems. Since this function explodes with larger errors, it suffers with outliers in the input data (isolated points that are located very far from the targeted values). In general, many applications that include squared loss are coupled with filtering the outliers first or else this loss function may not be most suitable one to be optimized.

2.3.2 Absolute Loss

For a given input x , hypothesis $y = f(x)$ and the true label t , the absolute loss function is defined as:

$$l(t, y) = |t - y|. \quad (2.3.2.1)$$

In other words, it is the absolute difference between the true label and the prediction of the input x . Absolute loss is applicable to regression problems like the squared loss, but

also it avoids the problem of outliers because it weights the outliers linearly instead of quadratically.

Fig 10. Plot of an absolute loss function.

2.3.3 Hinge Loss

For a given input x , hypothesis $y = f(x)$ and the true label t , the hinge loss function is defined as:

$$l(t, y) = \max(0, 1 - t \cdot y). \quad (2.3.3.1)$$

Hinge loss works well in cases like support vector machines where the more error the more penalty is. Hinge loss doesn't work very well in regression due to its one-sided error.

Fig 11. Plot of hinge loss function.

2.3.4 Negative Log Likelihood Loss

Log likelihood $\mathcal{L}(\theta, \mathcal{D})$ of a classifier for a given dataset $\mathcal{D}$, hypothesis $f(x)$ (classification function associated with model $P(Y|x, \theta)$) parameterized by θ and input x with its corresponding true label t is given by:

$$\mathcal{L}(\theta, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log P(Y = t^i | x^i, \theta) \quad (2.3.4.1)$$

In general, we try to maximize the log likelihood but higher the likelihood doesn't mean more the number of correct predictions. But, they are considered to be close when a classifier is randomly initialized. Since we generally minimize a loss function we alternatively minimize the negative of the log likelihood function. In this way negative log likelihood loss (NLL loss) is defined as:

$$\mathcal{NLL}(\theta, \mathcal{D}) = - \sum_{i=0}^{|\mathcal{D}|} \log P(Y = t^i | x^i, \theta) \quad (2.3.4.2)$$

NLL Loss is easily differentiable and is widely used in modern optimization techniques.

2.4. Optimization

In any supervised machine learning algorithm, developing a hypothesis and formulating its appropriate loss function with known true labels is followed by learning the parameters involved. The key idea is to optimize the loss function in small steps over multiple iterations. Doing this, parameters that are involved in formulating the loss function are learnt/optimized. These optimized parameters when used in the hypothesis is expected to provide accurate predictions. Widely used optimization techniques are gradient descent

algorithm (GD), Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS), Quasi-Newton method etc. In further sections, variants of gradient descent are discussed.

2.4.1 Gradient Descent

Gradient descent is an optimization algorithm which tries to find the minimum of a function. It is also called steepest descent. By taking steps proportional to the negative of the gradient of the function at the given value, gradient descent tries to reach the minimum of the function. Alternatively, if steps were taken in the direction proportional to the positive of the gradient it is supposed to lead to the maximum of the function.

Fig 12. Illustration of gradient descent algorithm where steps are taken from x_0 to x_4 in the direction of negative of gradients at each position. Here x_0 to x_4 are data points on the contour plot of a loss function whose minimum is at the center.

Let $l(X)$ be a multivariate loss function which is defined and differentiable near point x_1 , then $l(X)$ decreases while going from x_1 in the direction of negative gradient $-\nabla l(x_1)$ of $l(X)$ at x_1 . In this way moving from x_1 to $x_2, x_3, \dots, x_n$ in the direction of $-\nabla l(x_i) \forall i \in 1, 2, 3, \dots, n$ is expected to lead to the minimum of the loss function.

Parameter update based on gradient descent for a weight W_{ij} in the network can be represented as:

$$\Delta W_{ij}(t) = \eta \frac{\partial l}{\partial W_{ij}} \quad (2.4.1.1)$$

$$W_{ij}(t) = W_{ij}(t) - \Delta W_{ij}(t) \quad (2.4.1.2)$$

Gradient descent is relatively slow, technically its asymptotic rate of convergence makes it slower compared to other methods. In poor conditions, gradient descent keeps oscillating in zigzag manner. It doesn't always guarantee the global minimum and involves the risk of ending at a local minimum. On the other hand, gradient descent is widely used because it can be applied to optimize any problem in any number of dimensions.

2.4.2 Stochastic Gradient Descent

Stochastic gradient descent is variant of gradient descent algorithm. In GD all samples are used to make a single update for a parameter in a particular iteration, where as in SGD a single data point is used. This is done by considering a single data point of the training set and evaluating its gradient to update a parameter in an iteration. Technically, SGD is defined as GD for minimizing an objective function that is expressed in terms of sum of differentiable functions.

Stochastic gradient descent is a very popular algorithm used in machine learning algorithms including support vector machines, logistic regression and neural networks (coupled with back propagation which is discussed later).

2.4.3 Batch Gradient Descent

Batch gradient descent considers the whole dataset to compute gradient to update a parameter. This is done by considering the sum of the gradients at every point to update a parameter in an iteration. The effective use batch GD is mini-batch gradient descent.

2.4.4 Mini-batch Gradient Descent

Mini-batch gradient descent evaluates the cumulative gradients over a batch of samples to make an update. In this method, updates are made once after every batch is evaluated. This helps in improving accuracy as well as stabilizing the classifier. Whenever there is a considerable increase in the accuracy after consecutive iteration it is recommended to increase the number of samples considered per batch. When number of samples in batch are equal to the total number of samples, then this is equivalent to the batch gradient descent algorithm. In general, once when all samples of training set are considered it is called as an epoch. It is not necessary that an epoch is equivalent to an iteration. Training is generally performed over several epochs to evaluate the performance of the classifier.

2.4.5 Momentum

In some cases, cost function can have multiple local minima along with global minima. It is possible that in such cases, the network could be stuck at a local minimum since gradient descent is a greedy algorithm. This is more probable with the randomly initialized weights and low learning rates.

Fig 13. Illustration of a situation where a network stuck at some local minima.[19]

Looking at the previous update of weights ($\Delta W_{ij}(t - 1)$) to decide the current update is one way to avoid converging at local minima. [18] Proposes (2.4.5.1) as a solution with hyper parameter α , momentum.

$$\Delta W_{ij}(t) = \eta \frac{\partial l}{\partial W_{ij}} + \alpha \Delta W_{ij}(t - 1) \quad (2.4.5.1)$$

Fig 14. (a) & (b) are illustrations of how momentum help moving over local minimum. [20]

Fig. 14. (a) shows how high momentum helps the network to move over local minima. In Fig.14. (b), even if network is stuck at local minima, they can move away from local minima with an optimal momentum.

2.5. Regularization

In some situations, neural networks end up with very low performance on unforeseen test/validation data even though the network had very high performance on the training data. This situation is called over fitting. Fig 15. Illustrates a scenario where non-linear mapping would result in good generalization (left) and a worst possible generalization (right). In general, the right mapping will perform best on training data. On the validation set, the mapping on the right will perform worse failing to generalize the unforeseen data. This kind of undesirable situation is called over fitting.

Fig 15. Illustration of properly fitted/well generalized data vs Overfitted data. [21]

Fig. 16. Shows an example of over fitting where validation cost increases from point a to b (a & b are different iterations).

Fig 16. Illustration of cost function during over fitting, where training cost is very less compared to validation error. [21]

In this section, four widely used regularization techniques are discussed, which make neural networks or other classifiers better at generalizing training data rather than memorizing/ over fitting the train data.

2.5.1 L1 Regularization

L1 regularization restricts over fitting by restricting the weights from not becoming too large. A penalty term, also referred to as a regularizing term is added to the loss function shown in (2.5.1.1). This tends to make networks prefer smaller weights. l_0 represents the unregularized loss function and l is the regularized loss function. λ is the regularization hyper parameter and n is number of weights (W).

$$l = l_0 + \frac{\lambda}{n} \sum |W| \quad (2.5.1.1)$$

2.5.2 L2 Regularization

L2 regularization also reduces over fitting by restricting the weights from becoming too large just like L1 regularization. However, instead of making the penalty proportional to the sum of absolute weights, it makes the penalty proportional to the sum of squares.

$$l = l_0 + \frac{\lambda}{2n} \sum |W^2| \quad (2.5.2.1)$$

The L1 regularization term and L2 regularization term aren't the same and it isn't expected that they get exactly the same behavior. In both cases, they try to shrink the weights, but the way they do is different. In L1, weights shrink in proportion to a constant value where as in L2, the weights shrink in proportion to weights. So when a particular weight has a large magnitude, L1 regularization shrinks the weight much less than L2 regularization does [22]. By contrast, when $|W|$ is small, L1 regularization shrinks the weight much more than L2 regularization. The net result is that L1 regularization tends to concentrate the weight of the network in a relatively small number of high-importance weights (connections), while the other weights are driven toward zero.

2.5.3 Max Norm constraints

Sometimes, L1 and L2 regularizations fail to limit the weight. Larger value of weights often induce poor generalization of the data. To avoid such situations, max norm constraints enforce an absolute upper bound on the weights. In general, this is implemented as a regular weight update with regularization by clamping the weights above an upper bound. This regularization allows usage of higher learning rates without exploding the network weights as their updates are always bounded.

2.5.4 Dropout

Dropout is completely different technique as compared to L1 and L2 regularization schemes. Instead of restricting weights, dropout modifies the network by dropping hidden nodes with a certain probability, leaving the input and output untouched. In convolutional neural networks, dropout is most often employed in the fully connected layers.

Fig 17. Illustration of dropped connections. [22]

Table 1. Comparison of performance of different regularization techniques on MNIST data.

Regularization Method	Test Classification Error (%)
L2	1.62
L2+L1	1.60
Max norm constraints	1.35
Dropout + L2	1.25
Dropout + Max norm constraints	1.05

With dropout, the network is trained multiple times. Each training is modified by randomly dropping some of the hidden nodes and forward propagating with the new network and estimating a new loss and gradients. Those hidden nodes that are not dropped are forced

to be the same across all training threads. For each training thread, a different subset of hidden nodes are randomly dropped, and only those nodes not dropped contribute to the loss and gradients for updating the weights and biases.

2.5.5 Data Augmentation

Neural networks are data driven. They require more training examples. Obtaining more training data is a great idea to avoid over fitting. One such idea is data augmentation. Data is artificially expanded to generate more training data. Rotating by a small angle in multiple directions, scaling, shifting and affine transformations etc. are different data augmentation techniques that artificially expand the training data by producing synthetic data which is visually similar to the original data. Popular augmentation techniques are horizontal flipping, random cropping and color jittering. Combinations of different augmentations are often used, e.g., rotation followed by random cropping.

Using PCA to alter the intensities of RGB channels in training images is an another technique proposed by Krizhevsky et al. [5]. In this method, PCA is performed over RGB pixel values throughout the training images and each image is added multiples of principal components with magnitudes proportional to eigenvalues times a random variable picked from a Gaussian with zero mean and standard deviation of 0.1.

2.6. Back Propagation

In order to update weights in a neural network, appropriate gradients and weight updates have to be calculated based on the loss incurred due to loss function and it has to be proportionally distributed among the different hidden layers based on the activation functions they have. The partial derivate of the loss function with respect to all the weights in the network are calculated and updated in the process of back propagation [18].

Algorithm 1: Back Propagation:

Initialize random weights

for epoch=1:epochs

for a=1:n (n is number of training examples)

 % Calculate the outputs using forward propagation:

$$y_i^{(a)} \quad \forall i = 1 \text{ to } M \text{ (M is number of layers)}$$

 % Calculate the output error values:

$$\delta^{(M,a)} = y_M^{(a)} - t^{(a)}$$

 % Calculate the other gradient values:

$$\delta^{(i,a)} = \delta^{(i+1,a)} \times W^{(i+1)} \times f'(\sigma^{(i,a)}) \quad \forall i = M-1 \text{ to } 2$$

 % Update cost derivative:

$$\frac{\partial l}{\partial W^{(i)}} = \frac{\partial l}{\partial W^{(i)}} + \delta^{(i,a)} \times \sigma^{(i-1,a)}$$

 % Update all weights:

$$W^{(i)} = W^{(i)} - \eta \frac{\partial l}{\partial W_{ij}^{(i)}}$$

end

end

Fig 18. Algorithm 1 explains the back propagation.

With regard to Algorithm 1:

- $y_i^{(a)}$ represents the output of a^{th} training sample at i^{th} layer of the network which contains M layers.
- $\delta^{(M,a)}$ represents output error of a^{th} training sample obtained by finding the difference between output of final layer ($y_M^{(a)}$) and corresponding ground truth label ($t^{(a)}$).
- $\delta^{(i,a)}$ represents gradient corresponding to a^{th} training sample at i^{th} layer.
- $W^{(i)}$ represents weight vector of the i^{th} layer and $f'(\sigma^{(i,a)})$ is the derivative of the activation function f . $\sigma^{(i,a)}$ represents the weight sum of the a^{th} training sample at i^{th} layer; and
- $\frac{\partial l}{\partial W^{(i)}}$ is rate of the change of loss function w.r.t weight vector of the i^{th} layer. η is the learning rate.

Chapter 3 Deep Learning Models

Deep learning models have surpassed the performance of traditional computer vision and machine learning algorithms [23],[24] for object recognition tasks. The introduction of deep convolutional models in 2012 has revolutionized the computer vision and pattern recognition communities. Krizhevsky et al., 2012 [5] trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ILSVRC-2010 dataset to one of 1000 different classes and achieved top-1 and top-5 error of 37.5% and 17.0% on the test sets. The results were considerably better than the previous state of the art results. Later in 2013, Clarifai [2], reduced the top-5 error to 11.2%. The ILSVRC 2014 challenge saw a considerable decrease in the top-5 error to 7.35%, 7.32% and 6.67% by the entries MSRA [25] , VGG[6] and GoogLeNet [7]. Recently, with the introduction of new activations like PReLU's [3] top-5 error has surpassed the human level performance (5.1%, Russakovsky et al. [2]) by achieving 4.94% error on the test set and MSRA ResNet-152 [26] achieved 3.57%.

Fig 19. Shows the progress of classification performance (top-5 error %) on Imagenet dataset over past few years. Dashed line indicates Human performance (5.1%).

Recent improvements in classification accuracy can be attributed to advances in deep architectures with more layers and effective use of regularization methods [25], [27],[28],[29],[30]. Zeiler & Fergus [31] improved classification results by introducing random crops on training samples and improved parameter tuning methodologies. Simonyan and Zisserman [6] investigated the usage of network depth and C. Szegedy, et al. used banks of smaller convolutional filters [7] to simultaneously improve accuracy and lesson the number of parameters. X. Zhang, et al. [25] computes the feature maps from the entire image only once, and then pools features in arbitrary regions (sub-images) to generate fixed-length representations.

Other advances include the use of new non-linear activations [3],[27],[32],[33],[34],[35]. Rectifier Linear Units (ReLU) are key to recent success in the deep architectures. X. Zhang, et al. [3] use variants of ReLU which adaptively learns the parameters of the activation. In [3], an initialization method is introduced which helps in convergence of very deep models.

3.1. AlexNet

AlexNet contains eight weight layers, which consists of 5-convolutional layers followed by 3-fully connected layers. The output of the last fully connected layer feeds the softmax probability layer which yields distributions over 1000 classes of the ImageNet dataset. The input image size is 224x224 RGB image. The first convolutional layer filters the input image with 96 filters of 11x11x3 filter size with a stride of 4. The second convolutional layer has 256 filters of size 5x5x96 and the output is sent to local response normalization (LRN) layer followed by max-pooling by a 3x3 window and stride 2. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or

normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

Table 2. AlexNet Configuration [5].

AlexNet	
5 weight layers	
Input (224×224 RGB Image)	
Conv 11×11×3 96, Stride 4	
Conv 5×5×48 128	Conv 5×5×48 128
Maxpool [3,3 stride 2]	
Conv 3×3×256 394	Conv 3×3×256 394
Maxpool [3,3 stride 2]	
Conv 3×3×192 192	Conv 3×3×192 192
Conv 3×3×192 128	Conv 3×3×192 128
Maxpool[3,3 stride 2]	
FC-4096	
Relu, dropout	
FC-4096	
Relu, dropout	
FC-1000	
Softmax

Fig 20. Illustration of AlexNet CNN configuration.

Table 3. Case Study on AlexNet Parameters & Memory [16].

Layer, Input Size	Memory	Parameters
Input: [224×224×3]	224×224×3=150K	0
Conv-11 96: [55×55×48]×2	55×55×96=290K	11×11×3×48×2=104,544
Conv-5 256: [55×55×128]×2	55×55×128×2=770K	5×5×48×128×2=307,200
Pool-2: [27×27×128]×2	27×27×128×2=187K	0
Conv-3 384: [27×27×192]×4	27×27×384×2=560K	3×3×256×384×2=1,769,472
Pool-2: [13×13×192]×2	13×13×192×2=67K	0
Conv-3 384: [13×13×192]×2	13×13×192×2=67K	3×3×192×192×2=663,552
Conv-3 256: [13×13×128]×2	13×13×128×2=43K	3×3×192×128×2=442,368
Pool-2: [6×6×128]×2	6×6×128×2=9K	0
FC-1: [1×1×4096]	4096	6×6×256×4096=37,748,736
FC-2: [1×1×4096]	4096	4096×4096=16,777,216
FC-3: [1×1×1000]	1000	4096×1000=4,096,000
TOTAL	2.2M	61.9M

3.2. VGG

VGGNet was a runner-up in the 2014 ILSVRC. It has shown that network depth is critical for good performance. VGGNet has several configurations listed in TABLE IV. Perhaps the best network in terms of computational complexity vs. accuracy contains 16 weight layers containing 3×3 convolution layers and 2×2 pooling layers. It is more expensive to evaluate and uses more memory and parameters than AlexNet (five variations of VGGNet shown in TABLE III). The incorporation of 1×1 conv. layers (configuration C, TABLE IV) is a way to

increase the nonlinearity of the decision function without affecting the receptive fields of the convolutional layers.

Table 4. Parameters of various VGG Net Configurations.

Network	A	A-LRN	B	C	D	E
Parameters	133M	133M	133M	134M	138M	144M

Table 5. Multiple VGG Net Configurations [6].

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
Input (224×224 RGB Image)					
Conv 3 64	Conv 3 64 LRN	Conv 3 64 Conv 3 64	Conv 3 64 Conv 3 64	Conv 3 64 Conv 3 64	Conv 3 64 Conv 3 64
Maxpool					
Conv 3 128	Conv 3 128	Conv 3 128 Conv 3 128	Conv 3 128 Conv 3 128	Conv 3 128 Conv 3 128	Conv 3 128 Conv 3 128
Maxpool					
Conv 3 256	Conv 3 256	Conv 3 256	Conv 3 256	Conv 3 256	Conv 3 256
Conv 3 256	Conv 3 256	Conv 3 256	Conv 3 256 Conv 1 256	Conv 3 256	Conv 3 256 Conv 3 256 Conv 3 256
Maxpool					
Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512
Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512 Conv 1 512	Conv 3 512	Conv 3 512 Conv 3 512 Conv 3 512
Maxpool					
Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512
Conv 3 512	Conv 3 512	Conv 3 512	Conv 3 512 Conv 1 512	Conv 3 512	Conv 3 512 Conv 3 512 Conv 3 512
Maxpool					
FC-4096					
FC-4096					
FC-1000					
Softmax					

Table 6. Case Study on VGG-D Model.

Layer, Input Size	Memory	Parameters
Input: [224×224×3]	224×224×3=150K	0
Conv-3 64: [224×224×64]	224×224×64=3.2M	3×3×3×64=1,728
Conv-3 64: [224×224×64]	224×224×64=3.2M	3×3×64×64=36,864
Pool-2: [112×112×64]	112×112×64=800K	0
Conv-3 128: [112×112×128]	112×112×128=1.6M	3×3×64×128=73,728
Conv-3 128: [112×112×128]	112×112×128=1.6M	3×3×128×128=147,456
Pool-2: [56×56×128]	56×56×128=400K	0
Conv-3 256: [56×56×256]	56×56×256=800K	3×3×128×256=294,912
Conv-3 256: [56×56×256]	56×56×256=800K	3×3×256×256=598,284
Conv-3 256: [56×56×256]	56×56×256=800K	3×3×256×256=598,284
Pool-2: [28×28×256]×2	28×28×256=200K	0
Conv-3 256: [28×28×512]	28×28×512=400K	3×3×256×512=1,179,648
Conv-3 256: [28×28×512]	28×28×512=400K	3×3×512×512=2,359,296
Conv-3 256: [28×28×512]	28×28×512=400K	3×3×512×512=2,359,296
Pool-2: [14×14×512]	14×14×512=100K	0
Conv-3 256: [14×14×512]	14×14×512=100K	3×3×512×512=2,359,296
Conv-3 256: [14×14×512]	14×14×512=100K	3×3×512×512=2,359,296
Conv-3 256: [14×14×512]	14×14×512=100K	3×3×512×512=2,359,296
Pool-2: [7×7×512]	7×7×512=25K	0
FC-1: [1×1×4096]	4096	7×7×512×4096=102,760,448
FC-2: [1×1×4096]	4096	4096×4096=16,777,216
FC-3: [1×1×1000]	1000	4096×1000=4,096,000
TOTAL	24M*4bytes	138M

Chapter 4 Datasets

In this thesis, experiments are performed on CalTech101 [36], CalTech256 [37], CIFAR100 [38] & ImageNet datasets. Datasets were processed through multi-layer perceptron (MLP) as well as convolutional neural networks (CNNs). MLP pre-processing often involves extracting hand crafted features (such as SIFT, HOG, LBP features) from images while CNN has no pre-processing as it will learn these features as part of the back propagation process. Both methods often involve mean subtracting images, and generally resize images to a fixed input size. For Transfer learning images are resized to 224x224x3. For CalTech101 and CalTech256 and the training and test splits are obtained using a 6-fold cross validation for all the datasets. Later sections describe each dataset.

4.1. CalTech 101 & 256

CalTech 101 has images of objects belonging to 102 classes, with 40 to 800 images per class with image size of 300×200×3 pixels. Most categories have about 50 images and were collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato.

Fig 21. Average of images of 100 different classes in CalTech 101 dataset.

The CalTech256 dataset has 257 classes, with 80 to 827 images per class with image size of 300×200×3 pixels.

Links to datasets:

http://www.vision.caltech.edu/Image_Datasets/Caltech101/

http://www.vision.caltech.edu/Image_Datasets/Caltech256/

4.2. CIFAR-100

The CIFAR100 dataset has 100 classes with 500 images for training and 100 images for testing respectively per class and has an image size of 32×32×3 pixels. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs) [38].

Table 7. Shows list of superclasses and classes in CIFAR-100 dataset [38].

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
Fish	aquarium fish, flatfish, ray, shark, trout
Flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
Insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
People	baby, boy, girl, man, woman
Reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
Trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Link to dataset: <https://www.cs.toronto.edu/~kriz/cifar.html>

4.3. ImageNet: ILSVRC-2012

The dataset includes images of 1000 classes, and is split into three sets: training (1.3M images), validation (50K images), and testing (100K images with held-out class labels). Experiments are performed on the training and validation sets to evaluate the performance.

Fig 22. Collage of images from different object categories in ImageNet:ILSVRC2102 dataset [1].

Chapter 5 Hierarchy Models

There are numerous works describing hierarchical decomposition of classification problems [39]. One of the earliest attempts of a CNN hierarchical approach [40] used transfer learning from sub-groups with many samples to sub-groups with few. Deng et al. [41] used a hierarchy of label relations, and further improvements were made by [42] and [12] using two and many categories respectively.

Confusion matrices can be used to determine hierarchical clusters [43], [44]. Podolak [45] increased robustness by allowing classes to fork in more than one hierarchical branch. Salakhutdinov et al. [46] combined structured hierarchical Bayesian models with deep learning to generate a framework that can learn new concepts with a minimal number of training samples.

CNN hierarchical improvements were demonstrated by [7],[47], and a category hierarchy CNN based classifier was demonstrated in [12] that builds a two stage classifier to separate easy and difficult classes. Unfortunately the memory footprint and time constraints were a major challenge.

In this thesis a novel method is proposed to alleviate the computational complexity involved in training larger networks for datasets with higher number of discrete classes or concepts. Our approach uses a high-level classifier to initially determine which sub-class a sample belongs to, then passes that sample into the corresponding sub-class network to make a final class assignment. Our method automatically determines the optimal number of sub-classes, then trains each sub-class in an independent fashion. The first stage of determining the number of sub-classes is called Hierarchy Clustering. In this stage by exploiting the rich information from the class-to-class confusion matrix (generated using a simplified

conventional neural network mapping to all classes or concepts) to extract hidden correlations amongst classes. During training, a Hierarchy Classifier predicts which sub-network a sample belongs. This sample is then passed into one of C Smaller Class Assignment Classifiers, each which is only concerned with a subset of classes to make a final classification estimate.

5.1. Hierarchy Clustering

To tackle problems with a large number of classes, a hierarchical approach for clustering similar classes into sub-groups is used. This requires the training of a handful of much simpler neural networks where the number of overall parameters has been reduced. The intuition behind using hierarchical clustering is the presence of coarse categories or super classes which contain a higher number of finer classes. To categorize the given set of classes into super classes, spectral clustering of the confusion matrix is used to generate a given number of clusters. The main challenge with the hierarchical clustering scheme is the selection of an optimum merge or split breakpoints, which if done improperly, can lead to low quality clusters. To address this challenge, a multi-phase technique that is based on the analysis of the confusion matrix of the classifier in the parent stage is proposed.

Linkage statistics are used for getting the correlation indicators among classes in a hierarchical configuration. The distance matrix D , which is estimated from the confusion matrix C , measures the dissimilarity among different classes. If a stage p has K_p clusters of classes, D has dimensions $K_p \times K_p$, where an element $D_p(C_i, C_j)$ represents the dissimilarity between cluster C_i and cluster C_j . An unweight pair group method based on the arithmetic mean is used for determining the linkages between individual clusters. $D_p(C_i, C_i) = 0 \quad \forall i \in K$, represents the dissimilarity of a cluster with itself. A top-down divisive strategy is used to find non-overlapping classes that starts by including all classes in a single cluster. The parent

cluster is subdivided into smaller class clusters until a termination criterion is met. The dissimilarity between clusters helps in dynamically determining the split points with an upper limit on the number of sub-clusters. As a result, this technique automatically adapts to the internal characteristics of the data.

Algorithm 2: Hierarchy class clustering:

Hierarchy relationships between classes are derived using the confusion matrix C_p that measures linkage distances d between classes. To form clusters with overlapping classes, threshold class posterior probabilities DCN for classes originally not in cluster.

Input: Confusion matrix C_p at classification stage p

Output: Overlapping class labels Q

Initialize: Upper limit on non-overlapping cluster size θ and overlapping factor γ

1) Compute distance matrix D from C_p

2) Compute linkage statistics:

$$d(r, s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} \text{dist}(x_{ri}, x_{sj}),$$

Where x_{ri} and x_{sj} are dissimilar groups with n_r and n_s elements, respectively

3) Compute cumulative linkage values $Cum(d)$

4) **for** descending values k in $Cum(d)$

$\alpha = \text{no. of classes with } d < k$

if $\alpha > \theta$ **then**

group classes α as new cluster Q

repeat until all classes are assigned clusters

end

5) Compute column normalized confusion matrix (DCN)

6) **for** each cluster Q_i

if $DCN_p(C_i, C_j) \geq (\gamma \cdot K_{p-1})^{-1} \forall C_i \notin Q_i, C_j \in Q_i$ **then**

append class j to cluster Q_i

end

Fig 23. Algorithm 2, Hierarchy class clustering algorithm.

Small non-overlapping class groups are obtained by grouping similar classes together. However, in a non-overlapping setting, a sample that is misclassified at a parent level, has no chance of getting predicted correctly at the lower levels. Therefore, the small clusters are overlapped using the posterior probabilities to achieve higher generalization accuracy. The confusion matrix of the parent cluster is column normalized (DCN_p) to obtain the class

posterior probabilities. An element $DCN_p(C_i, C_j)$ represents the likelihood that a sample is of true class C_i given that it was predicted as class C_j . Let Q_i be the collection of classes in cluster i , then the condition that certain classes are similar to this cluster can be given as,

$$DCN_p(C_i, C_j) \geq (\gamma \cdot K_{p-1})^{-1} \forall C_i \notin Q_i, C_j \in Q_i \quad (2.1.11)$$

A parametric threshold of $(\gamma \cdot K_{p-1})^{-1}$ is used, where γ is an overlapping hyper-parameter that determines the probability for including a class in cluster Q_i . The value of γ depends on the number of classes in the original problem and the number of clusters in the parent stage.

(a)

(b)

Fig 24. Illustration of Hierarchy clustering on Toy data having 11 classes. (a) Shows a dendrogram with dissimilarity among the classes. (b) Shows the 3 non-overlapping clusters formed with similar classes grouped together (left) and the overlapping cluster (right).

5.2. Hierarchy Classifier

Let S be the training set with N classes, where C clusters are formed after Hierarchical clustering such that C clusters have $n_1, n_2 \dots n_c$ number of classes $\forall n_1 + n_2 + \dots + n_c = N$. The classes associated with n_1 are labelled with class 1 and n_2 as class 2 and similarly $n_3 \dots n_c$ as class 3 ... class c . In this way, the training set S is classified into C outputs instead of N classes. An unforeseen test sample when passed through the network shown in Fig. 12 enters the hierarchy classifier. The hierarchy classifier directs the test sample in to one of the C networks. Once the test sample passes through a network in a class assignment classifier, the final class prediction is obtained.

Fig 25. Illustration of Hierarchical deep network framework.

5.3. Class Assignment Classifier

The class assignment classifier consists of several simple neural networks (C neural networks in Fig 25) predicting smaller number of classes at each neural network. The class assignment classifier outputs N classes, i.e., all classes of the dataset are classified at this stage of hierarchical deep network. In order to address misclassification at hierarchy classifier, overlapping clusters allow a test sample to be passed to more than one assignment classifiers. Let $p_1, p_2, \dots, p_c$ be the predictions of the hierarchy classifier for the corresponding C outputs and $q_1, q_2, \dots, q_{n_1}$ be the predictions of Network 1 for the corresponding n_1 outputs. When overlapped clusters are used, the top k predictions are used, each of which is a product of predictions from the hierarchy classifier and class assignment classifier. This product, referred to as confidence ($\mathbb{C}$), is the predicted final classification output:

$$Confidence(\mathbb{C}_i) = p_j * q_k \forall i \in 1 \dots N, j \in 1 \dots C, k \in 1 \dots n_j \quad (2.1.12)$$

Fig 26. Flow of classification using hierarchal deep networks.

Fig 27. Illustration of hierarchical Models.

Hierarchical Models do not allow different configurations for different sub-networks. Therefore, an adaptive network selection is required which considers both the number of classes as well as statistical properties in a cluster to select an appropriate network configuration. Several experiments are performed to predict the best possible configuration for a given cluster. Convolutional Neural Networks (CNN) are used as a base configuration and the adaptive network model determines a configuration that will be used on an individual cluster as the class assignment classifier. In addition to CNNs, the model may select a Multi Layer Perceptron (MLP) neural network as appropriate. Later sections will discuss how the adaptive network selection and transfer learning approaches are deployed.

5.4. Adaptive Hierarchical Models

Both network depth and number of nodes per layer are important in optimizing classification problems. Since hierarchical clusters exhibit unique statistical properties, selecting an appropriate network would typically improve the overall classification accuracy. Attributes such as the number of classes as well as confusion matrix linkage and class-to-class correlation statistics allow an adaptive network framework to predict between CNN and MLP architectures as well as both the number of layers and nodes per layer for a given classification problem. A regressive model is trained on the cluster properties and their corresponding best network is selected as the network architecture for each given cluster.

To train this regressive model, multiple network configurations were trained on varied clusters, each cluster with unique statistical properties. The model learns to correlate cluster attributes to the corresponding best network architecture.

Fig 28. Illustration of network selection manager that selects the best network configuration.

Fig 29. Illustration of Adaptive Hierarchical Model.

5.4.1 Transfer Learning

CNNs simultaneously learn feature extractors along with a feature classifier for unsurpassed classification performance. It has been shown that using previously learned feature extractor and classifier weights enable both higher classification accuracies and faster learning convergence when applied to new classification problems. For example, the popular architectures of AlexNet, VGG, and GoogLeNet pre-trained on ImageNet offer an excellent weight initialization for new visual classification problems. The first few layers of CNNs describe high level abstract features which apply equally to most visualization problems, negating the need for further fine-tuning. Initializing all weights from a previously learned network, then keeping the first few layers of transferred parameters constant while learning

the rest of the weights typically offers increased performance. Further, because of the desirable initialization, only a few training epochs are generally needed to converge to a solution for most hierarchy models. Experiments demonstrate the impact transfer learning has on the adaptive architectural selection.

In this thesis, the optimized parameters obtained from VGG trained on ImageNet are used for 20 epochs for the transfer learning experiments.

5.4.2 Joint Model

In earlier sections, either convolutional neural networks or multi-layer perceptron neural networks in an adaptive mode are used. Now the adaptive model allows the use of convolutional neural networks and multi-layer perceptron neural networks simultaneously to select the best network configuration for a particular subnetwork.

Fig 30. Illustration of network selection manager that chooses between a convolutional neural network or a multi-layer perceptron neural network.

Chapter 6 Experiments & Results

As part of this thesis, experiments on datasets were processed through multi-layer perceptron (MLP) as well as convolutional neural networks (CNNs). MLP processing used HOG input features. CNN processing used mean subtracted images resized to $64 \times 64 \times 3$ for CalTech101 and CalTech256. The training and test splits are obtained using a 6-fold cross validation for all the datasets. For CIFAR-100 training and testing splits are used as provided in [38] and images mean subtracted keeping the size at $32 \times 32 \times 3$. ILSVRC 2012 provides training and validation sets for ImageNet dataset. These images are preprocessed with mean subtraction and random cropping to a size of $224 \times 224 \times 3$.

6.1. Hierarchical Model Experiments

All hierarchical model experiments are performed using model shown in Fig. 31. Experiments are performed by choosing classifiers between a CNN and MLP.

Fig 31. Illustration of Hierarchical Model.

6.1.1 CalTech 101 Results

In Table 8, top line indicates performance of a single large MLP neural network with two hidden layer of size [200 150]. MLP neural network used with two hidden layers of size [25 10] used in each class assignment classifier for the rest of the lines. Hierarchical Clustering is controlled by varying the parameter Gamma (γ). HC indicates Hierarchy Classifier accuracy and FC indicates Final Classification accuracy. C represents number of clusters formed using hierarchical clustering.

Table 8. Shows performance of Hierarchical Model using an MLP on CalTech101 dataset.

C	Method	Gamma(γ)	HC (%)	FC (%)
1	NA	NA	NA	45.6
44	Non-overlap	NA	69.43	61.39
44	Overlap	3	69.05	61.56
44	Overlap	5	62.73	60.13
44	Overlap	8	52.05	58.61

MLP processing on the CalTech101 dataset increases the final accuracy by approximately 15.79% using a non-overlapping hierarchical architecture. Similar observations were found with overlapping hierarchical architecture, but performance decreases with increasing overlap factor. This was attributed to the increase in confusion in the hierarchical stage. It should also be noted that the memory requirements increase as the overlap factor increases due to larger mini-networks.

Fig 32. Example of a Dendrogram with 102 Classes of CalTech101 dataset generated using confusion matrix obtained from a single CNN (Better viewed in color).

The dendrogram in Fig.32, represents the class grouping formed using the linkage statistics for CalTech 101 dataset. The colors in the graph depict groups of classes as determined by the algorithm described in Fig. 23. While analyzing the groups, it is observed that similar classes were grouped together which proves the efficacy of the hierarchical clustering.

Table 9. Estimation of memory and computations in a single MLP on CalTech101 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	1764	0
FC-1:[1764×250]	250	1764×250=441,000
FC-2:[250×100]	100	250×100=25,000
FC-3:[100×102]	102	100×102=10,200
TOTAL	2216	476,200

HoG features (1×1764) of input images ($64 \times 64 \times 3$) are extracted and used in CalTech101 experiments with an MLP classifier.

Table 10. Estimation of memory and computations in a Hierarchy Classifier on CalTech101 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	1764	0
FC-1:[1764×25]	25	$1764 \times 25 = 44,100$
FC-2:[25×10]	10	$25 \times 10 = 250$
FC-3:[10×44]	44	$10 \times 44 = 440$
TOTAL	1843	44,790

Table 11. Estimation of worst case memory and computations in a Class assignment classifier on CalTech101 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	0	0
FC-1:[1764×25]	25	$1764 \times 25 = 44,100$
FC-2:[25×10]	10	$25 \times 10 = 250$
FC-3:[10×12]	12	$10 \times 12 = 120$
TOTAL	47	44,470

*In worst case, maximum number of classes in a class assignment classifier is selected .

Table 12. Comparison of worst case memory and computations between single MLP and hierarchical model on CalTech101 dataset during testing of an image.

Model	Memory	Computations
Single MLP	2216	476,200
Hierarchical Model	1890	89,260

It is observed that hierarchical models require comparatively less memory and computations while testing an unforeseen sample using MLP classifiers. It is also inferred that the automatic clustering of large classification problems into a hierarchy of smaller classification networks offers a solution whereby the smaller networks have less computations and are faster while offering increased classification accuracy. Current increase in accuracy is dependent on the number of class assignment classifiers allowed.

Table 13. Shows performance of Hierarchical Model using a CNN on CalTech101 dataset.

C	Method	Gamma(γ)	HC (%)	FC (%)
1	NA	NA	NA	55.84
44	Non-overlap	NA	62.42	51.57
44	Overlap	3	50.33	50.72

In Table 13, top line indicates performance of a single large Convolutional neural network. Similar CNNs are used in each mini-network.

When CNNs are used to evaluate the CalTech101 dataset, the final accuracy decreased by 4.27% using a non-overlapping hierarchical architecture. It is hypothesized the reason for this decline is due to

- 1) The identical architecture of all the mini-networks
- 2) When a cluster has fewer classes, the number of training samples for that network are also less, making them insufficient for CNNs.

CalTech 101 dataset have significant variation in number of samples per class. The accuracies would be improved if the number of samples were identical across all classes.

6.1.2 CalTech 256 Results

In Table 14, the top line indicates performance of a single large MLP neural network with two hidden layers of size [200 , 150]. The MLP neural network has two hidden layers of size [25 , 10] for each class assignment classifier.

Table 14. Shows performance of Hierarchical Model using an MLP on CalTech256 dataset.

C	Method	Gamma(γ)	HC (%)	FC (%)
1	NA	NA	NA	18.61
104	Non-overlap	NA	23.07	21.56
104	Overlap	3	24.49	21.96
104	Overlap	5	22.55	20.61

It was observed that the performance is increased by 2.95% in case of CalTech256 dataset when MLP neural network was used to evaluate the performance of the non-overlapping and overlapping hierarchical architectures.

Table 15. Shows performance of Hierarchical Model using a CNN on CalTech256 dataset.

C	Method	Gamma(γ)	HC (%)	FC (%)
1	NA	NA	NA	36.21
104	Non-overlap	NA	29.87	28.34
104	Overlap	3	30.65	25.62
104	Overlap	5	27.38	21.47

When convolutional neural networks are used to evaluate the CalTech256 dataset, the final accuracy decreased by 7.87 % using a non-overlapping hierarchical architecture. CalTech 256 dataset also has significant variation in number of samples per class. The accuracies would be improved if the number of samples were identical across all classes.

Fig 33. Example of a dendrogram with 257 Classes of CalTech257 dataset generated using confusion matrix obtained from a single CNN (Better viewed in color).

Table 16. Estimation of memory and computations in a single MLP on CalTech256 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	1764	0
FC-1:[1764×250]	250	1764×250=441,000
FC-2:[250×100]	100	250×100=25,000
FC-3:[100×257]	257	100×257=25,700
TOTAL	2371	491,700

Table 17. Estimation of memory and computations in a Hierarchy Classifier on CalTech256 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	1764	0
FC-1:[1764×25]	25	1764×25=44,100
FC-2:[25×10]	10	25×10=250
FC-3:[10×104]	104	10×104=1,040
TOTAL	1903	45,390

Table 18. Estimation of worst case memory and computations in a Class assignment classifier on CalTech256 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	0	0
FC-1:[1764×25]	25	1764×25=44,100
FC-2:[25×10]	10	25×10=250
FC-3:[10×21]	21	10×21=210
TOTAL	56	44,560

*In worst case, maximum number of classes in a class assignment classifier is selected (i.e,21).

Table 19. Comparison of worst case memory and computations between single MLP and hierarchical model on CalTech256 dataset during testing of an image.

Model	Memory	Computations
Single MLP	2371	491,700
Hierarchical Model	1959	89,950

It is observed that hierarchical models require comparatively less memory and computations while testing an unforeseen sample using MLP classifiers.

6.1.3 CIFAR-100 Results

In Table 20, top line indicates performance of a single large MLP neural network with two hidden layers of size [200 , 150]. The MLP neural network used with two hidden layers of size [25 , 10] used in each class assignment classifier.

Table 20. Shows performance of Hierarchical Model using an MLP on CIFAR-100 dataset.

C	Method	Gamma (γ)	HC (%)	FC (%)
1	NA	NA	NA	22.83
30	Non-overlap	NA	28.37	24.96
30	Overlap	3	27.89	24.82
30	Overlap	5	26.34	23.25

It was observed that the performance is increased by 2.13 % in case of CIFAR-100 dataset when MLP neural network was used to evaluate the performance of the non-overlapping and overlapping hierarchical architectures.

Table 21. Shows performance of Hierarchical Model using a CNN on CIFAR-100 dataset.

C	Method	Gamma(γ)	HC (%)	FC (%)
1	NA	NA	NA	42.29
30	Non-overlap	NA	46.91	37.56
30	Overlap	3	46.48	40.74

When convolutional neural networks are used to evaluate the CIFAR-100 dataset, the final accuracy decreased by 0.47% using a non-overlapping hierarchical architecture. The drop in accuracies is relatively less compared to CalTech101 and CalTech256.

Fig 34. Example of a Dendrogram with 100 Classes of CIFAR-100 dataset generated using confusion matrix obtained from a single CNN (Better viewed in color).

Table 22. Estimation of memory and computations in a single MLP on CIFAR100 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×576]	576	0
FC-1:[576×250]	250	$576 \times 250 = 144,000$
FC-2:[250×100]	100	$250 \times 100 = 25,000$
FC-3:[100×100]	100	$100 \times 100 = 10,000$
TOTAL	1026	179,000

HoG features (1×576) of input images ($32 \times 32 \times 3$) are extracted and used in CIFAR-100 experiments with an MLP classifier.

Table 23. Estimation of memory and computations in a Hierarchy Classifier on CIFAR-100 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×576]	576	0
FC-1:[576×25]	25	$576 \times 25 = 14,400$
FC-2:[25×10]	10	$25 \times 10 = 250$
FC-3:[10×36]	36	$10 \times 36 = 360$
TOTAL	647	15,010

Table 24. Estimation of worst case memory and computations in a Class assignment classifier on CIFAR-100 dataset during testing of an image.

Layer	Memory	Computations
Input:[1×1764]	0	0
FC-1:[576×25]	25	$576 \times 25 = 14,400$
FC-2:[25×10]	10	$25 \times 10 = 250$
FC-3:[10×18]	18	$10 \times 18 = 180$
TOTAL	53	14,830

*In worst case, maximum number of classes in a class assignment classifier is selected.

Table 25. Comparison of worst case memory and computations between single MLP and hierarchical model on CIFAR-100 dataset during testing of an image.

Model	Memory	Computations
Single MLP	1026	179,000
Hierarchical Model	700	29,840

6.2. Spectral Clustering

Spectral clustering [48] on low dimensional representations of the data may offer clues to natural cluster boundaries at a coarser level. To categorize the given set of classes into super classes, spectral clustering of the confusion matrix is used to generate a given number of clusters. The main challenge in this clustering scheme is the selection of an optimum merge or split breakpoints, which if done improperly, can lead to low quality clusters.

Fig 35. Illustration of hierarchical model generated using spectral clustering.

Table 26. shows performance of hierarchical model generated using spectral clustering on Caltech101 dataset. Top line indicates performance of a single large MLP neural network with two hidden layer of size [200 , 150]. MLP neural network used with two hidden layers of size [25 , 10] used in each mini-network for the rest of the lines. HC

indicates Hierarchy Classifier accuracy and FC indicates Final Classification accuracy. C represents the number of clusters formed using spectral clustering.

Table 26. Shows performance of Hierarchical Model using an MLP on CalTech101 dataset.

C	HC (%)	FC (%)
1	NA	45.6
5	80.3	35.2
10	70.1	50.2
15	65.2	51.0
17	63.2	53.7
20	62.9	49.0
25	59.9	50.8

It is observed that while using spectral density clustering the number of networks affect the final accuracy. An inappropriate choice of networks will lead to low quality clusters. It is observed that lower number networks improve hierarchy classifier accuracy but fail to improve final classifier accuracy. Choosing an optimal number of networks using dendrograms have already demonstrated a significant improvement in performance.

6.3. Adaptive Hierarchical Models

Several experiments were performed to demonstrate the application of the adaptive network selection manager. Multiple architectures were carefully pre-defined to feed the network selection manager which selects the best possible network for different class assignment classifiers. ImageNet100 and ImageNet200 are subsets of ImageNet with 100 and 200 classes randomly selected from the 1000 classes.

With regards to Table 27, the baseline network configurations which each use a single large CNN is referred to as *CNN1* from pre-defined configurations and in Table 28, *CNNTL1* from adaptive transfer learning configurations. The pre-defined CNNs, *CNN1* through *CNN5*, each contain convolution layers followed by pooling and the fully connected layers with a dropout ratio of 0.5. These networks are trained for 40 epochs with a learning rate 0.01 and momentum 0.9.

Table 27. Pre-Defined Convolutional Neural Network Configurations for Caltech101, Caltech256.

Pre-defined CNN Configurations					
Net	CNN1	CNN2	CNN3	CNN4	CNN5
Depth	6	6	6	8	10
Input: 64×64 RGB image					
Conv	5×5 32	3×3 32	7×7 32	5×5 32	5×5 32
ReLU,Maxpool 2					
Conv	5×5 32	3×3 32	5×5 32	3×3 32	3×3 32 3×3 32
ReLU,Maxpool 2					
Conv	5×5 64	3×3 64	3×3 64	3×3 32 3×3 64	3×3 32 3×3 64
ReLU,Maxpool 2					
Conv	5×5 64	3×3 64	3×3 64	3×3 64 3×3 64	3×3 64 3×3 128
ReLU,Maxpool 2					
FC	512	512	256	512	512
ReLU,Dropout					
FC	128	128	128	128	256 128
ReLU,Dropout					
FC	Num Classes	Num Classes	Num Classes	Num Classes	Num Classes

The adaptive transfer learning configurations, *CNNTL1* through *CNNTL5*, each contain eight weight layers of which the first five are convolutional layers and rest are fully connected layers with the same dropout ratio of 0.5. *CNNTL1* is identical to VGG-f Model [6].

Table 28. Pre-Trained configurations for ImageNet datasets.

Adaptive Transfer learning Configurations					
Net	CNNTL1	CNNTL2	CNNTL3	CNNTL4	CNNTL5
Depth	8	8	8	8	8
Input: 224×224 RGB image					
Conv	11×11 64 Stride:4	11×11 64 Stride:4	11×11 64 Stride:4	11×11 64 Stride:4	11×11 64 Stride:4
ReLU,Maxpool 2					
Conv	5×5 256	5×5 256	5×5 256	5×5 256	5×5 256
ReLU					
Conv	3×3 256	3×3 256	3×3 256	3×3 256	3×3 256
ReLU					
Conv	3×3 256 3×3 256	3×3 256 3×3 256	3×3 256 3×3 256	3×3 256 3×3 256	3×3 256 3×3 256
ReLU,Maxpool 2					
FC	4096	4096	4096	4096	4096
ReLU,Dropout					
FC	4096	4096	4096	4096	4096
ReLU,Dropout					
FC	Num Classes	Num Classes	Num Classes	Num Classes	Num Classes

The hierarchical model represents the framework used in section 6.1 whose hierarchical classifier and class assignment classifiers consists of *CNN1* and *CNNTL1* for models without and with transfer learning. The adaptive network selection model uses the adaptive network

section manager that chooses between *CNN1* through *CNN5* and adaptive transfer learning models let network section manager to choose between *CNNTL1* through *CNNTL5*. In this thesis, we use the optimized parameters obtained from VGG [6] trained on ImageNet for 20 epochs for the transfer learning experiments. Convolutional neural network configurations for CalTech 101 and CalTech 256 are shown in Table 27. CIFAR 100 has the same configurations except the top layer is excluded. Configurations for ImageNet are shown in Table 28 where layers in blue are the ones learned via transfer learning.

Conv 5×5|32 Stride: 4 indicates convolutional operation with 32 filters of size of 5×5 with a stride factor 4. ReLU, Maxpool|2 allows convolutional output to non-linear activation function followed by maxpooling with stride 2. FC is the number of nodes in the fully connected layer.

6.3.1 CalTech 101 Results

Table 29. Shows performance of a single CNN vs hierarchical model, adaptive network selection, adaptive transfer learning mode on CalTech101.

Table 29. Comparison of performance between different models on CalTech 101 dataset.

Model	Groups	Accuracy (%)
CNN1	NA	55.84
Hierarchical Model	44	51.57
Adaptive Network Selection	44	53.25
Adaptive Transfer Learning	44	54.85
Joint Model	44	55.12

It is observed that the non-adaptive hierarchical models and adaptive hierarchical models do not improve the performance over a single, large CNN. However, adaptive hierarchical models show improvement over non-adaptive hierarchical models.

6.3.2 CalTech 256 Results

Table 30. Shows performance of a single CNN vs hierarchical model, adaptive network selection, adaptive transfer learning mode on CalTech256.

Table 30. Comparison of performance between different models on CalTech 256 dataset.

Model	Groups	Accuracy (%)
CNN1	NA	36.21
Hierarchical Model	104	28.34
Adaptive Network Selection	104	29.54
Adaptive Transfer Learning	104	36.65
Joint Model	104	36.54

It is observed that the non-adaptive hierarchical models and adaptive hierarchical models do not improve the performance over a single, large CNN. Adaptive transfer learning model shows improvements compared to other models.

6.3.3 CIFAR 100 Results

Table 31. Shows performance of a single CNN vs. hierarchical model, adaptive network selection on CIFAR 100.

Table 31. Comparison of performance between different models on CIFAR 100 dataset.

Model	Groups	Accuracy (%)
CNN1	NA	42.29
Hierarchical Model	30	37.56
Adaptive Network Selection	30	44.67
Joint Model	30	44.67

It is observed that the adaptive hierarchical models improve the performance over a single, large CNN. Adaptive transfer learning experiments were not performed on CIFAR-100 due to the smaller size of the images in the dataset. It is assumed that these variations in performance are due to the data-driven nature of CNN's. Both the CalTech datasets have significant variation in number of samples per class, but the results presented in this thesis were obtained on the entire dataset. Since CIFAR-100 has sufficient samples per class, class assignment classifiers can optimize network weights. In contrast, CalTech-101 and CalTech-256 have many class assignment classifiers with few training samples.

The accuracies would be higher if the number of examples were identical across all classes. To validate this assumption, the performance of the non-adaptive hierarchical model as well as adaptive hierarchical models were evaluated on subsets of the ImageNet dataset.

To ensure the correctness of the taxonomy generated by our hierarchical models, Table 32 compares the performance of hierarchical models generated from hierarchical clustering with the taxonomy provided by the CIFAR-100 creators. The CIFAR Hierarchical Model and CIFAR Adaptive Network selection are hierarchical models generated using the CIFAR-100 taxonomy. Our taxonomy improves CNN1 accuracy by 5.62% where as CIFAR-

100 taxonomy improves CNN1 accuracy by 1.28%. It was observed that our taxonomy generated more groups or coarser categories.

Table 32. The performance of a single cnn vs. hierarchical models generated from hierarchical clustering and CIFAR taxonomy.

Model	Groups	Accuracy (%)
CNN1	NA	42.29
CIFAR Hierarchical Model	20	41.82
CIFAR Adaptive Network selection	20	42.83
Hierarchical Model	30	37.56
Adaptive Network Selection	30	44.67

6.3.4 ImageNet Results

Table 33 and 34 shows the performance of adaptive hierarchical models on the ImageNet100 and ImageNet200 datasets.

Table 33. Comparison of performance between different models on ImageNet 100 dataset.

Model	Groups	Accuracy (%)
CNN1	NA	63.24
Hierarchical Model	22	62.81
Adaptive Network Selection	22	66.45

Although hierarchical models do not improve performance over a single CNN, our adaptive hierarchical models increase performance by 5.07% and 4.69% relatively for ImageNet100 and ImageNet200 datasets respectively.

Fig 36. Example of a dendrogram with 100 Classes of ImageNet 100 dataset generated using confusion matrix obtained from a single CNN (Better viewed in color).

Table 34. Comparison of performance between different models on ImageNet 200 dataset.

Model	Groups	Accuracy (%)
CNN1	NA	58.47
Hierarchical Model	63	58.14
Adaptive Network Selection	63	61.21

Fig 37. Example of a dendrogram with 200 Classes of ImageNet 200 dataset generated using confusion matrix obtained from a single CNN (Better viewed in color).

Table 35. demonstrates similar results on the full ImageNet dataset with 1.3% increase in accuracy relatively compared to a single CNN.

Table 35. Comparison of performance between different models on ImageNet dataset.

Model	Groups	Accuracy (%)
CNN1	NA	55.32
Hierarchical Model	89	50.27
Adaptive Network Selection	89	56.06

Fig 38. Example of a dendrogram with 1000 Classes of ImageNet dataset generated using confusion matrix obtained from a single CNN

It is observed that some classes in ImageNet possess extreme similarities. This led to formation of clusters with many classes or fine categories. This hinders the performance of the hierarchy classifier, hurting the overall accuracy. We hypothesize that the hierarchy classifier's performance would be improved if the resulted clusters had an even distribution of classes across them.

6.4. Discussion

Hierarchical models offer significant benefits to datasets that have classes with higher visual similarities. Datasets with few classes could gain more performance using hierarchical models with MLP neural network or a simple linear classifier as base classifiers. On the other hand, datasets with large number of classes would benefit from using adaptive hierarchical models. These models fine tune individual subclasses to offer significant performance gains. The performances of hierarchical models demonstrated in this these are aimed to improve the

baseline results but are not intended to compete with the state-of-the-art results. This thesis proposes that as this new framework is applied to existing techniques, increased processing efficiencies and classification accuracies will result. Further, it is hypothesized that these frameworks are not limited to only visual recognitions, but can also be used in other domains wherever there is high similarity or correlation between different classes in the dataset.

Chapter 7 Conclusion

An automatic hierarchical clustering method is introduced which reduces parameters while simultaneously increasing classification accuracy. This new approach borrows concepts from traditional divisive clustering techniques as well as confusion matrix dissimilarity linkage tree decomposition, to create an iterative method which methodically identifies cluster boundaries in a natural fashion. Hierarchical cluster boundary formation was tested on both MLP and CNN classifier frameworks, and shows significant benefit to the former, but not the latter. It is hypothesized that other classification frameworks such as SVM and Bayes classifiers can also benefit from the hierarchical framework. What is most intriguing is that the proposed strategy allows for virtually unlimited number of classes in any particular classification problem.

The proposed adaptive network selection framework, consisting of hierarchical models based on adaptive transfer learning, outperform single CNN models. The class assignment classifier network configuration is based on class confusion and composition statistics. As the complexity of classification problems increase, hierarchical models will offer significant benefits for large scale classification problems.

Future work will demonstrate adaptive hierarchical clustering over multiple stages on the full ImageNet-22K dataset. Use of ensembles to improve hierarchical classifier accuracy, data augmentations on imbalanced cluster to eradicate biases in hierarchical classifier predictions, sharing initial layers among the class assignment classifiers etc. will be used to improve hierarchical framework performance.

Bibliography

- [1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 248–255.
- [2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and others, “Imagenet large scale visual recognition challenge, 2014,” *arXiv Prepr. arXiv1409.0575*.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.
- [4] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *CoRR*, vol. abs/1602.0, 2016.
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.
- [6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Iclr*, pp. 1–14, 2015.
- [7] C. Szegedy, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *2015 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 1–9, 2015.
- [8] D. L. Yamins, H. Hong, C. Cadieu, and J. J. DiCarlo, “Hierarchical modular optimization of convolutional networks achieves representations similar to macaque IT and human ventral stream,” in *Advances in neural information processing systems*, 2013, pp. 3093–3101.

- [9] A. Mohamed, G. E. Dahl, and G. Hinton, “Acoustic modeling using deep belief networks,” *Audio, Speech, Lang. Process. IEEE Trans.*, vol. 20, no. 1, pp. 14–22, 2012.
- [10] J. Hoffman, S. Guadarrama, E. S. Tzeng, R. Hu, J. Donahue, R. Girshick, T. Darrell, and K. Saenko, “LSDA: Large scale detection through adaptation,” in *Advances in Neural Information Processing Systems*, 2014, pp. 3536–3544.
- [11] T. Dean, M. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, “Fast, accurate detection of 100,000 object classes on a single machine,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1814–1821.
- [12] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu, “HD-CNN: Hierarchical Deep Convolutional Neural Network for Large Scale Visual Recognition,” pp. 1–13, 2014.
- [13] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *J. Physiol.*, vol. 195, no. 1, pp. 215–243, 1968.
- [14] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202.
- [15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient Based Learning Applied to Document Recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [16] F.-F. . . A. K. . Li, “Convolutional Neural Networks for Visual Recognition.”
- [17] Li Deng, “Tutorial Slides on Deep Learning,” *Deep Learning Technology Center, Microsoft Research, Redmond, WA. USA.*, 2014. .
- [18] R. J. W. D. E. Rumelhart, G. E. Hinton, “Learning internal representations by error

- propagation,” no. DTIC Document, 1985.
- [19] A. Y. Ng, “Machine Learning,” *Stanford OpenClassroom*. .
 - [20] A. J. papagelis and D. S. Kim., “Multilayer Perceptron Backpropagation.” .
 - [21] “10 Misconceptions about neural networks.” [Online]. Available: <http://www.turingfinance.com/misconceptions-about-neural-networks/>.
 - [22] “Improving the way neural networks learn.” [Online]. Available: <http://neuralnetworksanddeeplearning.com/chap3.html>.
 - [23] A. Berg, J. Deng, and L. Fei-Fei, “Large scale visual recognition challenge 2010.” 2010.
 - [24] J. Sánchez and F. Perronnin, “High-dimensional signature compression for large-scale image classification,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 1665–1672.
 - [25] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 37, no. 9, pp. 1904–1916, 2015.
 - [26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv Prepr. arXiv1512.03385*, 2015.
 - [27] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” *arXiv Prepr. arXiv1302.4389*, 2013.
 - [28] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv Prepr. arXiv1207.0580*, 2012.
 - [29] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *Proceedings of the 30th International Conference on*

- Machine Learning (ICML-13)*, 2013, pp. 1058–1066.
- [30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
 - [31] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer vision--ECCV 2014*, Springer, 2014, pp. 818–833.
 - [32] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. ICML*, 2013, vol. 30, p. 1.
 - [33] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.
 - [34] M. Lin, Q. Chen, and S. Yan, “Network in network,” *arXiv Prepr. arXiv1312.4400*, 2013.
 - [35] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, “Compete to compute,” in *Advances in Neural Information Processing Systems*, 2013, pp. 2310–2318.
 - [36] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories,” *Comput. Vis. Image Underst.*, vol. 106, no. 1, pp. 59–70, 2007.
 - [37] G. H. Griffin and A. D. Perona, “P. The Caltech-256,” 2012.
 - [38] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images.” Citeseer, 2009.
 - [39] A.-M. Tousch, S. Herbin, and J.-Y. Audibert, “Semantic hierarchies for image annotation: A survey,” *Pattern Recognit.*, vol. 45, no. 1, pp. 333–345, 2012.

- [40] N. Srivastava and R. R. Salakhutdinov, “Discriminative Transfer Learning with Tree-based Priors,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2094–2102.
- [41] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam, “Computer Vision -- ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I,” D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 48–64.
- [42] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, “Error-Driven Incremental Learning in Deep Convolutional Neural Network for Large-Scale Image Classification,” in *Proceedings of the 22Nd ACM International Conference on Multimedia*, 2014, pp. 177–186.
- [43] S. Godbole, “Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multi-way classifiers,” *Prog. Report, IIT Bombay*, no. MARCH 2002, p. 17, 2002.
- [44] Y. Xiong, “Building text hierarchical structure by using confusion matrix,” in *2012 5th International Conference on BioMedical Engineering and Informatics*, 2012.
- [45] I. T. Podolak, “Hierarchical classifier with overlapping class groups,” *Expert Syst. Appl.*, vol. 34, no. 1, pp. 673–682, 2008.
- [46] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba, “Learning with hierarchical-deep models,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 35, no. 8, pp. 1958–1971, 2013.
- [47] A. G. Howard, “Some improvements on deep convolutional neural network based image classification,” *arXiv Prepr. arXiv1312.5402*, 2013.

- [48] U. von Luxburg, “A tutorial on spectral clustering,” *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.