ON THE ANALYSIS OF DEEP CONVOLUTIONAL NEURAL NETWORKS APPLIED TO BUILDING DETECTION IN SATELLITE IMAGES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BATUHAN KARAGÖZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2015

Approval of the thesis:

ON THE ANALYSIS OF DEEP CONVOLUTIONAL NEURAL NETWORKS APPLIED TO BUILDING DETECTION IN SATELLITE IMAGES

submitted by **BATUHAN KARAGÖZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Fatoş Yarman Vural _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Prof. Dr. Fatoş T. Yarman Vural
Computer Engineering Department, METU _____

Prof. Dr. Enis Çetin
Electrical and Electronics Eng. Dept., Bilkent Univ. _____

Prof. Dr. Faruk Polat
Computer Engineering Department, METU _____

Assist. Prof. Dr. Sinan Kalkan
Computer Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: BATUHAN KARAGÖZ

Signature :

ABSTRACT

ON THE ANALYSIS OF DEEP CONVOLUTIONAL NEURAL NETWORKS APPLIED TO BUILDING DETECTION IN SATELLITE IMAGES

Karagöz, Batuhan

M.S., Department of Computer Engineering

Supervisor : Prof. Dr. Fatoş Yarman Vural

July 2015, 59 pages

Deep Learning has gained much interest recently, probably induced by the requirements to learn more complex and abstract concepts. As concepts to be learned become more abstract, their regions in the raw input space also become highly variational. In many cases, shallow architectures fail to learn highly variational functions. One area of interest where concepts to be learned are complex is remote sensing. In this thesis, performance and suitability of deep architectures for recognition of building patches in satellite images are analyzed and discussed. Main architecture that is the subject of interest in this thesis is Deep Convolutional Neural Networks. Deep Convolutional Neural Networks has proven to be state of the art machine learning systems in several pattern recognition tasks such as bank check reading, handwriting recognition and face detection. We focus on a particular CNN architecture and trained a Deep Convolutional Neural Network with fully supervised stochastic gradient descent. We obtained a classification accuracy of 90 percent on average which is promising for deep learning implementations on the Remote Sensing Domain. Several measurements on the penultimate layer activations has been employed to reveal insights about what the models learn. Despite seemingly high accuracy results, these measurements put forward that the architecture we pick is unable to learn high level features.

Keywords: Deep Learning, Deep Convolutional Neural Networks, Remote Sensing

ÖZ

UYDU GÖRÜNTÜLERİNDE BİNA TANIMAYA UYGULANAN DERİN EVRİŞİMSEL SİNİR AĞLARININ ÇÖZÜMLEMESİ ÜZERİNE

Karagöz, Batuhan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. Fatoş Yarman Vural

Temmuz 2015 , 59 sayfa

Derin öğrenme daha karmaşık ve soyut kavramların öğrenilmesi için, son dönemde büyük ilgi kazanmıştır. Öğrenilmek istenen kavramlar soyutlaştıkça, bu kavramların ham girdi uzayında yerleştiği bölgenin topolojisi de değişken hale gelmektedir. Bir çok durumda, sığ mimariler değişken fonksiyonları öğrenememektedir. Uzaktan algılama bu tür soyut ve karmaşık kavramların öğrenilmesinin gerekli olduğu alanlardan biridir. Bu tezde, derin mimarilerin uydu görüntülerine uygulandıklarındaki performans ve uygunluğu analiz edilmiştir. Bu tezin temel analiz öğesi olan mimari Derin Evrişimsel Sinir Ağlarıdır. Derin Evrişimsel Sinir ağlarının başarısı bir çok örüntü tanıma probleminde kanıtlanmıştır. Bu işlevlerden bazıları çek okuma, elyazısı tanımlama ve yüz tanımlama olarak gösterilebilir. Bu tezde belirli bir Derin Evrişimsel Sinir Ağı mimarisi odak noktası olarak seçilmiş ve denetlemeli öğrenmeyle eğitilmiştir. Bunun sonucunda %90 ortalama kesinlik oranı elde edildi. Bu rakam uzaktan algılamada Derin Evrişimsel Sinir Ağlarının kullanımı konusunda umut vermektedir. Kullandığımız Derin Evrişimsel Sinir Ağı modellerinin ne tür kavramları öğrendiğini daha iyi anlayabilmek amacıyla sonöncesi katman çıktıları üzerinde çeşitli ölçümler yapıldı. Elde edilen yüksek kesinlik oranlarına rağmen, bu ölçümler seçtiğimiz mimarinin yüksek seviye öznitelikleri öğrenemediğini ortaya koymaktadır.

Anahtar Kelimeler: Derin Öğrenme, Derin Evrişimsel Sinir Ağları, Uzaktan Algılama

To Pelin, my only and forever love

ACKNOWLEDGMENTS

I am most grateful to my advisor, Prof. Dr. Fatoş Tünay Yarman Vural for the many things she has become for me during my study. She did not only create and shape the skeleton of this work, she also supported the development of it in each an every way. Much more than making this study possible, she is the one who makes me who I want to be in the life possible. More than anything, she was the one who believes in me in any circumstances and makes everything she can during the toughest period of my life.

I sincerely appreciate the feedback offered by my thesis committee members Prof. Dr. Göktürk Üçoluk, Prof. Dr. Faruk Polat, Assist. Prof. Dr. Sinan Kalkan and Assist. Prof. Dr. Onur Pekcan.

I would like to thank to my family who give their best effort in the making of this thesis without expecting nothing in return as their love is the most sincere and unbounded. I give my gratitude to my dearest mother, Meral and my dearest father, Mümin whose efforts are always for helping me find happiness. I wish to thank to my precious sister, Hande for being my sister and for the joy and happiness she brings to our family.

I would like to thank to Dr. Kemal Göksan who is not only my therapist and doctor but also my spiritual mentor during this journey and hopefully for the rest of my life.

I would like to thank to my colleagues Orhan and Itır for their much welcomed advises. I also wish to thank to my labmates Emre, Burak, Arman, Hazal, Barış, Güneş and Sarper for their warm and intimate attitudes towards me.

I would like to show my gratitude towards my significant other Pelin with my deepest love. She is the one who make this thesis more than a bunch of papers as she is the one who makes everything in my life more than a bunch of tasteless

experiences.

I acknowledge the support of TÜBİTAK (The Scientific and Technological Research Council of Turkey) BİDEB 2210 graduate student fellowship during my M.Sc. education. I also acknowledge the support of HAVELSAN for sharing their satellite image database.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Deep Learning

For a long time, mankind has been in the pursue of creating intelligent machines. Until now, we, as scientists and engineers, achieved success in designing the fragments of intelligence to a certain extent, especially in the domains where we can define enough discriminative features and where we can represent subject of interests in concrete mathematical notion. Consider a decision making problem where the observations somehow, when made up into a feature vector representation, create linearly separable clusters of data for each decision. Such problems have the most simplest decision models which apply linear thresholds. From this point of view, every decision problem could be simplified into learning linear threshold functions if we would obtain a good representation. However, as concepts to be learned become more complex, their regions in the raw input space also become highly variational. In many cases, discriminative features are substantially unapparent to exert into the architecture, nor underlying factors-or latent variables- are easy to determine. How can we functionally, or more realistically speaking by using logic gates, define a concept such as *human*? This is the fundamental question of a branch of Machine Learning paradigm called *Representation Learning*. It suggests that representations learned by machines, themselves, yield better performances than that of the ones obtained with handcrafted representations. To discriminate *human* from other classes; instead of defining concept of *human* embedded into the machine, it is more suitable to design the machine such that it would be able to learn the representation of

human itself. There exists, however, the difficulty of extracting high-level, abstract features from highly variational raw input data. Obviously, there are certain covert features which separate *human* from other objects, such as head, arms, fingers, etc. However, these features themselves are still abstract, but not as much abstract as *human*. When we continue this process of creating feature hierarchies deeper by deeper, at some point we hopefully reach concepts like edges end corners which are mathematically well defined. It is this idea that offers a solution of the abstract feature learning problem and constitutes the heart of *Deep Learning* paradigm: By learning feature hierarchies, starting from very fundamental domain features, letters of a language, edges and contours in images, a machine can capture more and more abstract concepts.

As far as the current use of the phrase goes, *Deep Learning* refers to designing modern learning architectures which are formed by multiple, typically 3 to 10, layers of computational elements. A mathematically well defined measure of the *depth* of an architecture is the maximum length of sequentially wired computational elements. A more meaningful, but maybe less rigorous, definition to depth is proposed as the number of times we update the representation. When we discuss *Artificial Neural Networks*, these two definitions coincide, since each layer of a deep neural network creates a different representation of the raw data. Theoretical results pointed out in the Background Chapter already suggest that it is possible to obtain an exponential gain of computational units by increasing the depth. The key factor in this exponential gain is so-called *feature re-use*. It is a term referring to defining a set of features whose combinations can be used to construct more sophisticated features. Feature re-use is an inherent property of Neural Networks: Features of the same layer are constructed from the same set of lower layer features.

When we rely on constructing a machine based on feature re-use, the assumption is that the domain which the machine operates have a hierarchical structure by itself. This a strong yet very reasonable prior. The world around us has a hierarchical structure itself, starting from small scale particles, elements, molecules etc., ending up in very complex beings such as humans or computers. Moreover, there are several studies confirming that human brain has also a hierarchical

processing chain [24].

It is well-known that learning feature hierarchies can provide high computational gain by feature reuse. Another aspect of deep architectures which makes them appealing is that they do not rely on hand-crafted features/representations. Increasing the number of training examples and computational speed has a very crucial effect on learning deep architectures. Then, we can train larger and deeper networks and thus can learn wider dictionaries and deeper abstractions. Hence, deep learning methods scale very well with both statistical and computational costs. On the other hand, these methods are questionable from some aspects. Deep architectures suffer from the absence of a mathematical workbench in which we can analyse the learning and inference procedures and make rigorous assessments about quality of learning. Hyperparameters selection is another major cavity of deep learning methods. There is a little research about effects of hyperparameters on learning, especially their mutual effects. Most of the successful architectures have been proposed with either cross-validated or **deus ex machina** hyperparameters.

In summary, *Deep Learning* is an approach to AI which involves learning to disassemble a highly abstract concept into its compounds, revealing the underlying factors creating the observed data and being able to make generalizations from previously observed data to represent new data. The ultimate goal is designing architectures which are able to learn a dictionary of features/concepts large enough to represent the whole world, a task at which human brain become mastered.

1.2 Current State of Deep Learning and Convolutional Neural Networks

As far as learning feature hierarchies goes, neural networks are probably the best tools due to not only their flexibility and representational power but also their very nature to be able to capture hierarchies of representations. Hence, most of the modern deep architectures are more or less variations of *Multi Layer*

Perceptrons. Most popular of these architectures are Deep Belief Networks, Stacked Auto Encoders and Convolutional Neural Networks.

Deep Belief Networks were introduced in 2006 as a semi supervised deep architecture [8]. Hinton et al. proposed that *Restricted Boltzman Machines* can be trained and stacked onto each other to form a deep generative model. After unsupervised weight learning from Restricted Boltzman Machines, whole architecture is finetuned by supervised learning. Their study has been referred as a breakthrough since it opened the way through unsupervised learning of abstract representations. *Stacked Auto Encoders* have a similar training procedure with Deep Belief Networks [4]. The difference is that building blocks of DBNs are Restricted Boltzman Machines which are generative probabilistic models, whereas Stacked Auto Encoders are composed of Auto Encoders which are reconstructive models. Two models (DBNs and SAEs) has been proved to be asymptotically equivalent.

Convolutional Neural Networks were first introduced by Fukushima [6] and later improved by LeCunn et al. exploiting the backpropogation algorithm for learning [17]. They are the first instances of successful deep architectures. They have been used since mid eighties and achieved commercial success in specific areas. LeCunn's LeNet architectures have shown excellent performances at particular tasks such as document letter and number detection and face recognition. Until 2006, there was not much of a success in designing proper tools for deep learning other than Convolutional Neural Networks designed for very specific tasks. Massive amount of work has been done since and there are many novel architectures each with its own merits. Especially for, unsupervised feature learning, greedy-layer wise training strategy has shown great potential [19], [31]. Deep unsupervised learning relies less on large labelled datasets and thus produces exceptional results in transfer learning, i.e., when there are slight differences between training and test distributions. An example is being able to capture information on previously absent categories. Some studies put forward that features learned from a specific dataset can even be used in supervised settings with totally different datasets [19]. On the other hand, Deep supervised learning architectures, particularly Deep Convolutional Neural Networks, show outstand-

ing performances when it comes to supervised learning tasks with large number of labelled data.

1.3 Motivation

As stated in the previous section, Convolutional Neural Nets(CNNs) have been applied to specific domains such as document recognition, successfully. During the last three or four years, several papers have been published presenting CNN architectures with outstanding performances on more complex vision based classification problems. *Multi Column Deep Convolutional Neural Networks* proposed by Ciresan et al. demonstrate state-of-the-art performance on NORB [18] and CIFAR-10 [15] datasets [5]. Most notable performance result is obtained by Krizhevsky et al. on the ImageNet 2012 classification benchmark, with their fully supervised ten layer CNN model. They managed to bring the state-of-the-art error to 15.3% from 26.1% [16]. Since then, many competition winning performances are obtained by expansions of so called *AlexNet*s (derived from the firstname of the main author of [16]). Using an even deeper network error rate of ImageNet 2012 benchmark has been brought to 6.5% [26]. These studies demonstrates the potential of Convolutional Neural Networks as powerful, benchmark image classification architectures. Utilization of high powered GPUs is what makes these tremendous achievements obtained by Convolutional Neural Networks possible. Machine Learning libraries using CUDA Development Toolkit by Nvidia are the backbones of these studies. Development of these libraries which perform highly parallelized matrix operations open up the possibility of using Convolutional Neural Networks in previously unfeasible settings. In this regard, satellite images are challenging subjects with very large datasets both labeled and unlabeled. In this study, we seek to explore the capabilities of CNN architectures when they are used in object recognition in satellite images. Since this study can be regarded as a starting point of a new research area, we desire to focus on a relatively simple dataset that is explained in chapter 4. We aim to measure the performance on this dataset and extract some insights about what might be the cause of the shortcomings.

1.4 Contribution

In this thesis, we mainly analyze the statistical properties of feature mappings learned by Deep Convolutional Neural Networks that are trained on a specially created dataset. Having similar design choices such as activation function, error function, pooling and padding styles(each of these choices shall be explained in the Chapter 2 with possible options), we trained several CNNs with different hyperparameters such as convolution and pooling sizes, number of layers etc. To employ our experiments on a satellite image recognition task that is relatively simple, we prepare a dataset of satellite image patches all of which are labeled either as 'building patch' or 'non-building patch' in the hope that our architecture learns the concept of building parts. For comparative purposes we applied each CNN to CIFAR-10 dataset, too, and we analyze those networks as well.

In Chapter 4, we present accuracy results and precision-recall values for each CNN we trained. These results simply put forward the effects of hyperparameter selection, especially the effect of depth on the performance. Experiments suggest that a depth 3 architecture (2 convolution + 2 pooling + 1 softmax layer) is no worse than depth 4 or 5 architectures on our satellite image dataset. By comparison, same depth 3 architecture performs significantly worse than again the same depth 4 architecture on CIFAR-10 dataset.

Several statistical properties of representations described by layers of each network have been calculated. Mathematical definitions of these properties are presented in Chapter 3, calculation results are presented in Chapter 4 and their implications are discussed in Chapter 5. Although too complex in mathematical notion, more intuitive meanings of these properties could be as follows:

1. Redundancy: We measure the redundancy of a penultimate layer feature set using an approximation to the most linearly correlated subset to the network output.

2. Sparsity of representations in terms of percentage of total weight have been calculated to demonstrate that representations are highly sparse.

3. Similarity between activation patterns for two class samples have been calculated to show how much class specific each representation.

1.5 Content of the Document

In the second chapter, Convolutional Neural Networks are overviewed, starting from the building blocks, continuing with standard procedures and ending with modern techniques applied in CNNs.

In the third chapter, methodology used in our experiments explained in detail. General properties of Convolutional Neural Network that we train can be found in Chapter 2. In Chapter 3, we give all the steps of training and testing algorithms for our network models. Hyperparameters, filter sizes, number of layers, learning rate etc., and their choice rationales are also explained. In chapter 3, we also give the rigorous definitions of measures shortly discussed in the previous section.

Fourth chapter is devoted to explanation of dataset acquisition and the experiment results.

Finally, we give a brief summary of our study and discuss the continuation of this thesis at Conclusion.

CHAPTER 2

BACKGROUND ON CONVOLUTIONAL NEURAL NETWORKS

In this chapter, Convolutional Neural Networks are examined.

2.1 Fully Connected Neural Networks

2.1.1 Artificial Neurons as Computation Elements

Basic computational unit in a neural network is an artificial neuron. What an artificial neuron is intended to do is as follows: Let us denote the vector form of its input as

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)\,.$$

Its output is a binary variable where value 1 corresponds to *firing* or, as of machine learning jargon, *activation* of the unit.

$$y = \begin{cases} 1 & if\ \boldsymbol{w}\boldsymbol{x}^T > b \\ 0 & else. \end{cases}$$

Hence it is a threshold unit with each feature x_i assigned a weight w_i and the threshold value for activation is the parameter b which is referred as *bias* of the neuron. To simplify the notation, a constant input $x_0 = 1$ is fed to neuron and therefore input and weight vectors take the forms

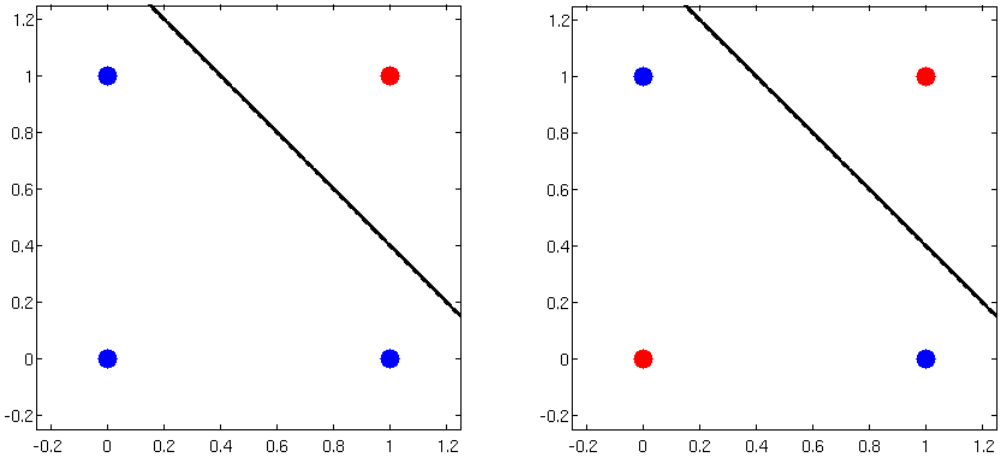

Figure 2.1: AND function is linearly seperable(left) whereas XOR function is not (right)

$$\boldsymbol{x} = (x_0, x_1, x_2, \ldots, x_n)\,.$$

$$\boldsymbol{w} = (w_0, w_1, w_2, \ldots, w_n)\,.$$

where $w_0 = -b$. Now, if $\Phi(.)$ is the step function, output can be reorganized as

$$y = \Phi\left(\boldsymbol{w}\boldsymbol{x}^T\right).$$

As far as classification problems go, this is equal to seperating the vector space $\mathbb{R}^n$ with a hyper plane with equation $\boldsymbol{w}\boldsymbol{x}^T = 0$.

If we put together multiple, say m, artificial neurons functionality, what we would obtain is a seperation of the vector space $\mathbb{R}^n$ into 2^m -if any two hyperplane corresponding to two neurons is different- clusters. Clearly there is nothing interesting about such a machine, since image classification tasks are far away from being linearly seperable. For demonstration purposes, even XOR function can easily be seen to be not linearly seperable in Figure 1.

2.1.2 Multilayer Perceptron

We can consider architecture presented in the previous section as a one layer neural network, since each computational element (a neuron) is directly fed by raw input. It is already discussed that this machine has no peculiarly important property. Nonetheless, if we also feed the output produced by this first *layer* to another *layer* of neurons, then such a machine may produce outputs of our interest. This class of machines are known as *Multi Layer Perceptron* and they can compute arbitrary functions as discussed shortly. In rigorous definition of a multilayer perceptron, first layer of neurons are called as *hidden units* and activation of hidden unit i are denoted as h_i. Second layer of neurons are called as *output units* and similarly, activation of hidden unit i are denoted as y_i. Following these notations, formula for h_i in terms of inputs is

$$h_i = \Phi\left(\boldsymbol{w}_i^{(1)}\boldsymbol{x}^T\right) = \Phi\left(\sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

and similarly formula for output y_i in terms of hidden unit outputs is

$$y_i = \Phi\left(\boldsymbol{w}_i^{(2)}\boldsymbol{h}^T\right) = \Phi\left(\sum_{j=1}^{n} w_{ij}^{(2)} h_j\right).$$

For a two class classification problem, output of the network, the vector y, happens to be a single binary variable in which a zero or one value corresponds to two different classes. In this case, we can sum up the whole network with one equation:

$$y = \Phi\left(W^{(2)}\Phi_{elm}(W^{(1)}\boldsymbol{x}^T)\right)$$

where $\{W^l\}_{ij} = w_{ij}^l$ and $\Phi_{elm}(.)$ is the elementwise application of the function $\Phi(.)$. Elementwise application of the function $\Phi(.)$ to a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ is given as

$$\Phi_{elm}(\boldsymbol{v}) = (\Phi(v_1), \Phi(v_2), \ldots, \Phi(v_n)).$$

Here we introduce a nonlinearity, using step function, between two linear projections of feature vector $\boldsymbol{x}$ identified by weight matrices $W^{(1)}$ and $W^{(2)}$. By

introducing this nonlinearity, we obtain a powerful model which can represent more than linear decision surfaces. In fact, there is a well known result about the representational power of multilayer perceptrons with step activation functions due to McCulloch and Pitts [21]:

Theorem 1. *McCulloch and Pitts: every basic Boolean function can be implemented using a linear threshold activation function, and given the appropriate inputs and bias weights.*

This theorem states two things:
1. Multilayer perceptrons with step activation functions are sufficiently capable of representing every function a digital computer can compute.
2. Multilayer perceptrons with step activation functions and with only *one* hidden layer is no less powerful than deeper neural networks in terms of representational power.
It is possible to enhance the representational capabilites of Multilayer Perceptrons even more, so to handle continuous inputs. In this regard, Multilayer Perceptrons are *Universal Approximators* provided that step activation function is replaced with a continuous one [25]. Most common choice for a continuous activation function is the *logistic sigmoid* function defined as:

$$\sigma(a) = \frac{1}{1 + e^{-a}}.$$

In other words, we can approximate any function with arbitrary precision in the following form:

$$f(\boldsymbol{x}) \approx \left(W^{(2)}\sigma_{elm}(W^{(1)}\boldsymbol{x}^T)\right)$$

where $\sigma_{elm}(\boldsymbol{v})$ is the elementwise application of the logistic sigmoid function to vector $\boldsymbol{v}$.

Despite their representational power, there is not much of a use for one hidden layer MLPs in practical terms. Result of Theorem 1 is no more than a mere statement of representational power. In fact, some very well defined boolean function classes have been proposed to be expressible by only exponential amount of logic units (AND and OR performing computational units) with architectures having

a number of layers less then a certain number. One such family of boolean functions are the well known parity functions. Following theorem states that they are impractible to be computed by a shallow architecture:

Theorem 2. *d-parity bit circuits with depth 2 have $O(2^d)$ number of logic gates. d-parity bit function is*

$$p_d(b_1, \ldots, b_d) = \begin{cases} 1 & if \ \sum_{i=1}^{d} b_i \ is \ even \\ 0 & otherwise. \end{cases}$$

where $(b_1, \ldots, b_d) \in \{0,1\}^d$. [28]

When we extend our building blocks from a set of logic gates to artificial neurons of subsection 1.1, there are still strong results suggesting arbitrarily deep architectures for some families of functions. Of particular interest is the following theorem, which applies to monotone weighted threshold circuits (i.e., multi-layer neural networks with linear threshold units and positive weights) when trying to represent a function compactly representable with a depth k circuit:

Theorem 3. *A monotone weighted threshold circuit of depth $k-1$ computing a function $f_k \in F_{k,N}$ has size at least 2^{cN} for some constant $c > 0$ and $N > N_0$.*

The class of functions $F_{k,N}$ is defined as follows. It contains functions with N^{2k-2} inputs, defined by a depth k circuit that is a tree. Input variables are fed to the leaves of the tree without negations, and the function value is produced at the root. Starting from first level there are alternating layers of OR and AND gates. The fan-in at the top and bottom level is N and at all other levels it is N^2. [11]

The above results do not prove that other classes of functions (such as those we want to learn to perform AI tasks) require deep architectures, nor that these demonstrated limitations apply to other types of circuits. However, these theoretical results raise the question with a high level of scientific suspicion: are the depth 1, 2 and 3 architectures (typically found in most machine learning algorithms) too shallow to represent efficiently more complicated functions which can be found in image related tasks, even there exists simply defined family of func-

tions whose compact representations require arbitrarily bigger depths? Another critical implication of Theorem 3 is that there is no such thing called universal depth. Each function appears to have a different minimum depth where it is compactly representable. Motivated by these ideas, architectures with more hidden layers have been created, thereby resulting *Deep Fully Connected Neural Networks*. Above formulas can easily be rewritten recursively, if we use upper indices to indicate the layer that a variable belongs to. Henceforth, we will follow the common nomenclature which makes a distinction between the *activation* of a unit and its *output*. Activation of the ith unit of lth layer is, then:

$$a_i^l = \sum_{j=1}^{M^l} w_{ij}^l x_j^{l-1},$$

and its output is in fact an input fed to the next layer, thus we will denote it with the symbol x_i^l:

$$x_i^l = \Phi\left(a_i^l\right).$$

By defining $\boldsymbol{x}^l$ as the column vector of scalars x_i^ls, $\boldsymbol{a}^l$ similarly and matrix W^l as weight matrix where

$$\{W^l\}_{ij} = w_{ij}^l$$

we can rewrite the above equalities as follows:

$$\boldsymbol{a^l} = W^l \boldsymbol{x^{l-1}},$$

$$\boldsymbol{x^l} = \Phi_{elm}\left(\boldsymbol{a^l}\right)$$

and finally, output of the network is:

$$\boldsymbol{y} = f\left(W^{L+1}(\boldsymbol{x^L})^T\right)$$

assuming that there are L number of hidden layers in the network.

We apply the function $f(.)$ to obtain a value in the range $[0, 1]$, so that we can treat the output of the network as the class probability estimates for the sample.

Using the logistic, we can map the whole real values into the interval $[0, 1]$:

$$p(\mathcal{C}_1|\boldsymbol{x}) = f(a) = \frac{1}{1+e^{-a}} = \frac{p(\mathcal{C}_1, \boldsymbol{x})}{p(\mathcal{C}_1, \boldsymbol{x}) + p(\mathcal{C}_2, \boldsymbol{x})}.$$

This will help us in defining a plausible and differentiable error function.

2.1.3 Parameter Optimization

Using the logistic sigmoid function, we built a probabilistic discriminative model from perceptron units. Therefore, it is convenient to seek for a maximum likelihood solution in the parameter space. For a given input set $\boldsymbol{x_1}, \ldots, \boldsymbol{x}_N$, let us define the training set $\boldsymbol{t} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}$ and the actual class labels c_i where y_i is the output of the network for the input vector $\boldsymbol{x}_i$. Then, the joint likelihood function of the training set $\boldsymbol{t}$ is

$$p(\boldsymbol{t}|\boldsymbol{W}) = \prod_{i=1}^{N} y_i^{c_i}\{1-y_i\}^{1-c_i}.$$

Maximizing this probability is equivalent to minimizing the negative loglikehood. This will give the *cross entropy* error function:

$$E(\boldsymbol{t}; \boldsymbol{W}) = -\ln\, p(\boldsymbol{t}|\boldsymbol{W}) = -\sum_{i=1}^{N}\{c_i \ln\, y_i + (1-c_i)\ln\,(1-y_i)\}.$$

There is no closed form solution of the equation $\nabla E_{\boldsymbol{W}}(\boldsymbol{t}; \boldsymbol{W}) = \boldsymbol{0}$ for neural networks with *cross entropy* error function. Hence, parameters are optimized through gradient descent learning procedure which has the simple update rule:

$$\Delta W^{(k+1)} = -\alpha\, \nabla E_{\boldsymbol{W}}(\boldsymbol{t}; \boldsymbol{W}^{(k)}) = -\alpha \sum_{i=1}^{N} \nabla E_{\boldsymbol{W}}(\boldsymbol{x_i}; \boldsymbol{W}^{(k)})$$

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \Delta W^{(k+1)}$$

where the upper indices of the parameter vector $\boldsymbol{W}$ correspond to its value in each iteration. This is called *batch* gradient descent, since the parameter vector $\boldsymbol{W}$ is updated with the cumulative errors of all the training samples. We will

make three modifications to this simple rule:

1. Regularization: Without a penalty on magnitudes of weights, it is possible that procedure does not converge even to a local minima. This is because parameter space is not bounded. To overcome this, we adopt the standard regularization penalty called *weight decay*. We will add a quadratic term into our error function, and it becomes:

$$E_R(\boldsymbol{t}; \boldsymbol{W}) = E(\boldsymbol{t}; \boldsymbol{W}) + \frac{1}{2}\lambda \sum_{w_{ij} \in \boldsymbol{W}} w_{ij}^2$$

Gradient of the updated error function is

$$\nabla_{\boldsymbol{W}} E_R(\boldsymbol{x}; \boldsymbol{W}) = \nabla_{\boldsymbol{W}} E(\boldsymbol{x}; \boldsymbol{W}) + \lambda \boldsymbol{W}.$$

Corresponding update rule is then

$$\Delta W^{(k+1)} = -\alpha \sum_{i=1}^{N} \nabla E_{\boldsymbol{W}}(\boldsymbol{x_i}; \boldsymbol{W}^{(k)}) - \alpha\lambda \boldsymbol{W}^{(k)}.$$

2. Momentum: By adding a term proportional to the previous update value into the update rule, we keep the gradient move towards a relatively steady direction. This is called a *momentum* term. Using momentum helps in both espacing local minimas and fast convergence. It makes the update term:

$$\Delta W^{(k+1)} = -\alpha \sum_{i=1}^{N} \nabla E_{\boldsymbol{W}}(\boldsymbol{x_i}; \boldsymbol{W}^{(k)}) - \alpha\lambda \boldsymbol{W}^{(k)} + \gamma \Delta W^{(k)}$$

3. Stochastic Gradient Descent : Applying batch gradient descent on large datasets is impractical in terms of memory usage. Instead, common method is to use a single sample or a *minibatch* for the gradient calculation. However, the samples should be chosen randomly. Otherwise, the gradient may travel between local minimas corresponding to single sample updates. Resulting and the final

update rule is

$$\Delta W^{(k+1)} = -\alpha \, \nabla E_{\boldsymbol{W}}(\boldsymbol{x}^{(k)}; \boldsymbol{W}^{(k)}) - \alpha\lambda \boldsymbol{W}^{(k)} + \gamma \Delta W^{(k)}$$

where $\boldsymbol{x}^{(k)}$ values are chosen from the training set randomly.

2.1.4 Calculating the Gradient: Backpropogation

Gradient of the error function with respect to network weights is calculated by *Backpropogation* algorithm which is popularized by Rumelhart et al. [23]. The algorithm involves backpropogating through layers starting from output layer, an error term which is due to partial derivatives of error function with respect to weights. Gradient formulas are as follows, assuming that there are L number of hidden layers:

$$\nabla_{W^{L+1}} E(\boldsymbol{x}; \boldsymbol{W}) = (y - c)\boldsymbol{x}^{\boldsymbol{L}}.$$

and

$$\nabla_{\boldsymbol{a}^{L+1}} E(\boldsymbol{x}; \boldsymbol{W}) = (y - c)$$

where y is the network output and c is the actual class label for the input $\boldsymbol{x}$. An error term is defined for each layer as

$$\boldsymbol{\delta}^{\boldsymbol{l}} = \nabla_{\boldsymbol{a}^{l}} E(\boldsymbol{x}; \boldsymbol{W}), \ \forall l$$

which is the gradient of the error function with respect to activation values of the particular layer l. This error terms are calculated by starting from the output layer and proceeding through backward on the network. In other words, error is *backpropogated* through network:

$$\boldsymbol{\delta}^{\boldsymbol{l}} = \left((W^{l+1})^T \boldsymbol{\delta}^{\boldsymbol{l}}\right) \bullet \Phi'_{elm}(\boldsymbol{a}^{\boldsymbol{l}}).$$

After calculating the error terms, gradient of the error function with respect to weight vectors between layers can be calculated using the following formula:

$$\nabla_{W^l} E(\boldsymbol{x}; \boldsymbol{W}) = \nabla_{\boldsymbol{a^l}} E(\boldsymbol{x}; \boldsymbol{W}) \nabla_{W^l} \boldsymbol{a^l} = \boldsymbol{\delta^l} (\boldsymbol{x^{l-1}})^T$$

Backpropogation procedure for a single training sample is given in the Algorithm 1.

Algorithm 1 Backpropogation Algorithm with Forward Pass

for $l = 1, 2, \ldots, L$ **do**
  $\boldsymbol{x^l} \leftarrow \Phi_{elm} \left(W^l \boldsymbol{x^{l-1}} \right)$
end for
$y \leftarrow f \left(W^{L+1} (\boldsymbol{x^L})^T \right)$
$\delta^{L+1} \leftarrow y - c$
for $i = L, \ldots, 1$ **do**
  $\boldsymbol{\delta^l} \leftarrow \left((W^{l+1})^T \boldsymbol{\delta^{l+1}} \right) \bullet \Phi'_{elm} (W^l \boldsymbol{x^{l-1}})$
  $\nabla_{W^l} E(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{\delta^l} (\boldsymbol{x^{l-1}})^T$
end for

2.1.5 Limiting Factors of Fully-Connected Networks

Despite their high expressive power and simple gradient descent learning algorithm, fully connected neural networks suffer from several practical issues. Especially for vision tasks where size of the raw input vector is very large and concepts to be learned are complex ,or if we put it in another term, functions to be learned are highly variational, there seems to be two edges of architectural decisions:

1- Resource Problem: Above theoretical results already suggest that representing a concept such as BUILDING will require exponential number of hidden units with a shallow architecture. This exponentiality gives rise to impracticalities in both training and inference. As number of neurons, hence number of tunable elements, is exponential in input dimension, a good approximation can be obtained if there are also exponential number of training data available.

Even with the correct weights of neurons, inference will take exponential time for sequential computing, or exponential number of threads for parallel computing. Either way, exponential amount of resource is required.

2- Vanishing Gradient Problem: Even with increased depth, problems arise. As error is propogated through lower layers during back-propogation phase, partial derivatives become smaller and smaller, since the derivative of sigmoid and tanh functions tend to be very close to zero. Sepp Hochreiter's diploma thesis of 1991 [10] formally identified the reason for this failure in the "vanishing gradient problem." As errors propagate from layer to layer, they shrink exponentially with the number of layers. Since the lower layer derivatives are very close to zero, their weights stay very close to their initial values. Learning occurs only in upmost two or three layers. Hence, we end up with an effectively shallow architecture.

Unsupervised techniques have been developed to overcome this phenomena. These methods involves greedy layer wise training using input reconstruction. Study of Hinton et al. [8] can be seen as a breakthrough on training deep architectures. They proposes a method of training Deep Belief Networks layer by layer. In each layer, Posterior probability of layer just above is used as input prior and parameters are then learned by Contrastive Divergence. Bengio et al. [4] applies same strategy by stacking auto-encoders. At the very last they all require supervised learning of upper layer weights through back propogation. It is notable that back propogation will function as a mere fine-tuning algorithm for lower layers. Hence unsupervised layer wise training is nothing but an inialization of lower layer weights. Nonetheless, experimental results suggest that this method of inialization is a very good one indeed, since it has ability to construct a hierachy of features, thereby transforming raw input to high level abstractions. Theoretical justifiability of learning unsupervised feature hierarchies is still an open question.

Even with unsupervised techniques, fully connected deep networks are still an overkill in terms of tunable parameters if one considers vision tasks. As we shall see shortly, it can be made very strong and solid prior assumptions on the

computer vision domain. Recently, layerwise unsupervised training methods for convolutional neural networks are also been proposed. Notable ones are, Deconvolutional neural networks [30] and Convolutional Deep Belief Nets [19]. We will mention these studies in the convolutional neural network section. These methods are applicable to vision tasks with small labeled data.

Idea of deep learning, though not with this name, is around nearly for three decades. Yet, it has been recent that deep architectures came up to practical use. Until recently, the scene had been dominated by local template matching methods such as *Support Vector Machines*. The reasons for deep architectures being unfavorable are clearly discussed. Convolutional Neural Networks are the most successful exceptions to this fact. They are discussed in the following chapter.

2.2 Convolutional Neural Networks

2.2.1 Motivation

Fully connected neural networks suffer from curse of dimensionality and computational inefficiencies. However, the main drawbacks of fully connected neural networks results from their inability to be configurable according to domain properties. As stated earlier, several solid properties of images exist and they can lead to strong and instrumental priors on network structure.

Firstly, an object class is translation invariant in the sense that a translation of an object image which belongs to a particular class yields an image of same class. Thus, translation is label conserving on the image domain. Secondly, many of the objects of interest in the computer vision domain such as buildings, faces or handwritten symbols have also features invariant to local distortions.

These two ideas have even been used to enlarge training sets for particular applications including the one in this thesis. From a theoretical perspective, a fully connected neural network is able to learn to make inference invariant to shifting and local distortions. However, learning such a task would probably

result in multiple units with similar weight patterns scattered around different locations in the input. There would be multiple detector units which are trained seperately for the same feature. Soon it will be discussed that convolutional neural networks resolve this inefficiency by *weight sharing* among neurons of the same layer. Secondly fully connected neural networks also ignore the topological structure of the inputs. Images exhibit very strong local dependencies. Spatial correlations are observed among pixels which are locally connected. These correlations give rise to local features such as edges, corners which then constitute less local and more complex features such as eyes on faces, side walls or loft patterns on buildings. This is why many layered neural network architectures are well suited for object recognition tasks in images.

2.2.2 Main Architectural Properties of CNNs

Convolutional Neural Networks are variants of Multi Layer Perceptrons which are inspired from biology. From Hubel and Wiesel's early work on the cat's visual cortex [12] , we know there exists a complex arrangement of cells within the visual cortex. These cells are sensitive to small sub-regions of the input space, called a receptive field, and are tiled in such a way to cover the entire visual field. These filters are local in input space and are thus better suited to exploit the strong spatially local correlation present in natural images.

Additionally, two basic cell types have been identified: simple cells (S) and complex cells (C). Simple cells (S) respond maximally to specific edge-like stimulus patterns within their receptive field. Complex cells (C) have larger receptive fields and are locally invariant to the exact position of the stimulus.

The visual cortex being the most powerful "vision" system in existence, it seems natural to emulate its behavior. Many such neurally inspired models can be found in the literature.

Motivated by the goal of overcoming above mentioned deficiencies of fully connected networks to capture the properties shift invariance, local distortion invariance and high spatial correlation; Hubel and Wiesel's seminal study in-

spired three main architectural ideas : local receptive fields, shared weights, sub-sampling(or pooling). Fukushima was the first to combine these ideas into a network model called neocognitron [6]. Even if some parts of neocognitron -i.e. average pooling, unsupervised training via WTA-based unsupervised learning rules, or by pre-wiring, is now somewhat obsolete, Modern convolutional architectures are still constructed on the above mentioned three main ideas.

1-Local receptive fields: In images, one may assume that low level features are local in the sense that they are formed by spatially nearby input units- which is pixels for the first layer and lower level feature detections for upper layers. Findings of Hubel and Wiesel on cats visual system reveals that there exist simple cells which response to simple visual stimulus such as edges. These cells seem to be fed by a group of sensory neurons which is called their *receptive fields*. The neuron fires then, when its receptive field captures a particular pattern. When applied to neural networks, a neuron with a local receptive field acts as a local feature detector.

2-Shared weights: The features we try to detect is mostly in the input space. In other word, elementary features that are useful in one part of the image are very likely to be useful across entire image. This knowledge is exploited in convolutional neural networks by assigning same weights and receptive field shape to neuron groups. A group of such neurons spans a whole input map with their receptive fields and consequently composes a *feature map* themselves. The feature mapping from input map to the output map can be defined by one convolution operator. By such a design, we gain two important advantages. First one is the ability to detect translations of features, in other words property of being translation invariant. Second one is the most important factor for the success of convolutional neural networks. It is the computational gain obtained by the drastic decrease in the number of parameters.

3-Sub-sampling: As we proceed through more abstract representations,and finally to decision point which is the concept of building in the case of this thesis, of objects, positions of low level features become less and less informative. On the contrary, it is harmful to design the network to be sensitive to local shift-

ings of those features, since it greatly reduces the abstraction of the object and generalization performance. Once a convolution is applied and a feature map is obtained, exact positionings of those features becomes somewhat redundant and deceptive. For example, once we know that the input image contains the endpoint of a roughly horizontal segment in the upper left area, a corner in the upper right and the endpoint of a roughly vertical segment in the lower portion of the image, we can label it to be an instance of digit 7. To detect complex abstractions, we need much more feature maps in the upper layers. As convolutional feature maps have roughly the same size with their inputs, usually we end up with a much larger representation of the input from previous layer. To reduce this redundancy and create invariance to local distortions, sub-sampling layers follow convolutional layers in Convolutional Neural Networks. What a subsampling layer perform is reducing the resolution of the feature maps fed through it. Usually units of sub-sampling layers have 2x2 or 3x3 non-overlapping receptive fields. There are applications where receptive fields overlap, too. When such units perform averaging over its receptive field inputs, layer simply applies blurring to the feature map. Max pooling is preferred in comtemporary systems.

2.2.3 Technical Definition

Convolution of a two dimensional signal $X(N \times N)$ with a two dimensional filter $W(m \times m)$ is defined as follows:

$$Y_{ij} = (W * X)_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{m} W_{ij} X_{u+i,v+j}$$

where convolution map, or output map in our discussion, is of size $N - m + 1 \times N - m + 1$. However, Output map Y could arranged to be of same size with X applying zero padding during convolution operation. We shall assume that each convolution operation is applied with zero padding so to keep input and output map of same size.

Notice that receptive field of the convolutional layer unit Y_{ij} is the set

$$\{x_{u+i,v+j} \mid u, v \in \{1, \ldots, m\}\}.$$

If we denote the elementwise application of the activation function $\Phi(.)$ to the two dimensional map y as $\Phi_{elm}(y)$, a feature map corresponding to a particular filter can be written in as

$$Y^{l,o} = \Phi_{elm}(W^{l,o,i} * X^{l,i})$$

where indices l, i and o index the layer, the input map and the output map, respectively.

Subsampling layers are simpler and do not involve in learning themselves since they are defined only by receptive fields which are hyperparameters. They simply take some $k \times k$ region and output a single value, which is the maximum in that region. For instance, if their input layer is a $N \times N$ layer, they will then output a $N/k \times N/k$ layer, as each $k \times k$ block is reduced to just a single value in a winner takes it all manner. However, receptive fields of units in a pooling layer may overlap and indeed it is demonstrated to be useful in preventing overfitting in the study of Krizhevsky et al. [16].

A convolutional layer in a neural network is followed by an immediate subsampling layer. Since sub-sampling operations inside network is determined before training of the network, it is convenient to denote them with a single function $pool(.)$ on feature maps. Now, we may define a complete model with the following equations:

X^0 : An array of 2D input maps(R,G,B maps in our study).

y : A random variable corresponding to class likelihood $p(C_1|X^0)$.

$$A^{l,o} = \sum_{i=1}^{M_{l-1}} W^{l,o,i} * X^{l-1,i}$$

$$X^{l,o} = pool(\Phi_{elm}(A^{l,o}))$$

$$y = \sigma(\sum_{i=1}^{M_L} W^{L+1,1,i} * X^{L,i}).$$

Let us derive the backpropogation formulas:

$$\nabla_{W^{l,i,o}} E = X^{l-1,i} * \boldsymbol{\delta^{l,o}}$$

$$\boldsymbol{\delta^{l,i}} = \big(up(\boldsymbol{\delta^{l+1,o}}) * rot180(W^{l+1,o,i})\big) \circ \Phi'_{elm}(A^{l,i})$$

where $\boldsymbol{\delta^{l,i}}$ is the gradient of the error function with respect to $A^{l,i}(\nabla_{A^{l,i}} E)$, $up(.)$ operation backpropogates the error to the receptive field element that output the maximum value during the forward propogation phase and '$\circ$' operation is the *Hadamard* product.

Backpropogation algorithm for Convolutional Neural Networks follows these calculations in a fashion very similar to the Algorithm 1.

2.3 Current State of Convolutional Neural Networks

For many years, convolutional neural networks could not find an application domain other than hand-written digit classication and face detection. Recently however, there has been a growing interest in Convolutional Neural Networks. On one side, sophisticated unsupervised architectures have been proposed presumably inspired by reconstructive techniques which cause a breakthrough in the field [8] [4]. Lee et al. [19] employed Hinton's greedy unsupervised training strategy for deep belief networks to convolutional layers with a novel technique called probabilistic max pooling, thereby constructing a generative architecture called Convolutional Deep Belief Networks. Zeiler et al. [29] followed Bengio's

strategy instead and proposed an auto-encoding convolutional neural network which is called Deconvolutional Neural Network. Both studies focus on generating representative features rather than raw performances. On the other side, already existing fully supervised convolutional neural networks has been somehow renovated to deliver outstanding performances on several image classification benchmarks. Ciresan et al. [5] obtain state-of-the-art accuracy on both NORB and CIFAR-10 datasets and break record on ICDAR Chinese handwriting recognition benchmark by an architecture consists of multiple convolutional neural networks operating in parallel and then sum up to obtain an averaged score for each class. It is very remarkable that their convolutional neural network architecture is only a massive extension of early examples such as LeNet-7. What makes this breakthrough possible is the enormous improvement of processing units as they state that todays computer are 60000 times faster than the ones in early 90s and powerful GPUs can speed up a serial code by a factor of 100. Yet, the most remarkable success of fully supervised convolutional neural networks came from ImageNet LSVRC-2012 contest winner architecture of Krizhevsky et al. [16] with an error rate of 16.4%, compared to the 2nd place result of 26.1%. Several factors contributing to their performance can be listed as follows:

(i) An architecture with 60 million parameters: The network consists of five convolutional layers some of them followed by pooling layers and three fully connected layers. It is depicted in figure 2.

(ii) a very powerful GPU utilization with two GPUs and a very fast impelementation of convolution and other network operations.

(iii) A training set of over 1 billion images which is generated by boosting the original training data of roughly 1.2 million images: Clearly, 1 million images are not enough to learn 60 million parameters without overfitting. Hence, they boost the dataset by two label preserving image transformations. First, from each image of size 256x256, they generate 1024 images of size 224x224 by using same size sliding window with stride 1. Second, they alter intensities of images by adding a three dimensional RGB intensity vector unique for every image to each pixel in that image. The vector is chosen randomly by using a procedure

involving PCA analysis on RGB covariance matrix of the whole training set.

(iv) Modern techniques to prevent overfitting: Other than training set boosting, they prevent overfitting by using ReLU neurons, Dropout, Local Response Normalization and Overlapping Pooling.

2.4 Conclusion

We define the Convolutional Neural Networks starting from their building blocks. We present first a more broader architecture, Fully Connected Neural Networks. Their mathematical definition and representation power. Then, their training procedure based in Stochastic Gradient Descent and Backpropogation algorithms is explained. We define the Convolutional Neural Networks on top of the Fully Connected Neural Network architecture.

CHAPTER 3

ARCHITECTURE AND METHODOLOGY FOR BUILDING PART RECOGNITION IN SATELLITE IMAGES

3.1 Architecture

Convolutional Neural Network Architecture is explained in Chapter 2, comprehensively. The inference models of a Convolutional Neural Network have many hyperparemeters to be tuned before training procedure. It is possible to encapsulate these model hyperparameters into compact layer definitions as popular CNN libraries do(Caffe, cuda-ConvNet) [16]. Note that each layer of a neural network define a mapping from a representation of data to another one. These mappings are obtained by applying the layer function given in the following equation:

$$X^{l,o} = pool(\Phi_{elm}(\sum_{i=1}^{M_{l-1}} W^{l,o,i} * X^{l-1,i}))$$

where l, o and i are indices of the layer, the output map and the input map, respectively. Then, $W^{l,o,i}$ is a convolutional filter associated to the layer, output map and input map indexed by l, o and i, $\Phi_{elm}(.)$ is the elementwise application of the activation function, $pool(.)$ is the subsampling(pooling) function. Defining a layer is equivalent to defining this mapping function. The mapping function is composed of filters $W^{l,o,i}$, activation function $\Phi(.)$ and the pooling function

pool(.). We examine these components and present our design choices in terms of them.

3.1.1 Convolutional Filters

What a layer performs is mostly discriminated by convolutional filters it learned. In fact, it is possible to make qualitative evaluation of layer filters by examining their visualization. This is quite straightforward to employ in first convolutional layer filters, since they can be visualized as three band images. On the other hand, upper layers are hard to visualize and a great deal of work is devoted to this task in the field [19] [31].

Filter size and the number of output maps are the hyperparameters on convolutional filters:

1. Filter size is directly related to the abstraction level of a layer when it uses the previous layer's representation. Layers with small filter sizes capture simpler spatial relations whereas layers with large filter sizes may detect complex features. In an extreme case, a 1×1 filter size does not detect any spatial relation about the texture of an image at all; on the other hand, a maximum size filter is actually a fully connected unit and it lacks the image domain priors mentioned in chapter 2, such as translation invariance and locality of the features. One should consider the issue of selecting filter size from both computational cost and overfitting-underfitting viewpoint. During hyperparameter selection process, we pick the 5 as the optimal value for the filter size. A justification can be found in Chapter 4.

2. Number of Output Maps is the only layer specific hyperparameters in our models. CUDA-ConvNet library performs convolution operation most efficient when the number of output maps are powers of 2. We determine the number of output maps accordingly. Bengio suggests using an *overcomplete* first hidden layer, meaning that it is more convenient to have more units in the first layer than the input [2]. Aside from that, state-of-the-art CNN architectures usually shrink the layers through higher layers, i.e. pyramid-like architectures are preferred

[17] [16]. We follow the same approach and design an expanding first layer and shrinking layers thereafter. This approach is summarized in the Table 3.3.

3.1.2 Convolution Style

There is not much of a modification over the convolution operation defined in the Section 2. We apply zero padding with standard convolution operation as follows: From input map $x(N \times N)$, we define

$$x' = \begin{bmatrix} 0_{m' \times m'} & 0_{m' \times N} & 0_{m' \times m'} \\ 0_{m' \times N} & x & 0_{m' \times N} \\ 0_{m' \times m'} & 0_{m' \times N} & 0_{m' \times m'} \end{bmatrix}$$

where $m' = \frac{m-1}{2}$, and the $0_{a \times b}$ is the a by b matrix with all zero entries. Since the size of x' is $N + m - 1 \times N + m - 1$, resulting output map of operation $y = W * x$ will have the size $N \times N$ which is equal to the dimensions of input map x. This convolution style is applied in all the models we trained.

3.1.3 Activation Function $\Phi(.)$

There are several activation function choices presented in the literature. Most common choice of activation functions in the history of Neural Networks are the sigmoid and hyperbolic tangent functions which are effectively equivalent in the sense that there is a linear transformation of parameters which we can transform a network using one of them to another network using the other. Recently, sigmoid and hyperbolic tangent functions are observed to be inefficient in terms of learning rate and also classification performance. Different activation functions used in neural network research are given in Table 3.1 with their derivatives.

As stated in the Chapter 2, ReLU activation is presented to be very useful in fast convergence and enforcing sparsity. Moreover, ReLU nonlinearity is observed to be utterly important for overcoming the vanishing gradient problem. Our experiments agree with its importance as learning rate saturates at very

Table3.1: Unit Activation Functions

Name	Function	Derivative
sigmoid	$\Phi(x) = \frac{1}{1+e^{-x}}$	$\Phi'(x) = \Phi(x)\,(1 - \Phi(x))$
hyperbolic tangent	$\Phi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$\Phi'(x) = 1 - \Phi^2(x)$
ReLU	$\Phi(x) = max(0, x)$	$\Phi'(x) = \begin{matrix} 1 & if\ x > 0 \\ 0 & else. \end{matrix}$
Softplus	$\Phi(x) = ln(1 + e^{x})$	$\Phi'(x) = \frac{1}{1+e^{-x}}$
Absolute Rectifier	$\Phi(x) = \lvert x \rvert$	$\Phi'(x) = sign(x)$

low accuracies when training the network with sigmoidal activation functions. Results of these controlled experiments could be found at Chapter 4.

3.1.4 Subsampling

Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap. In other words, a pooling layer can be considered as a grid of pooling units spaced s units apart, each summarizing a neighborhood of size $z \times z$ centered at the location of the pooling unit. If we set $s = z$, we apply the traditional local pooling process as commonly employed in Convolutional Neural Networks. If we set $s < z$, we perform an overlapping pooling process. In this study, we use overlapping pooling with stride $s = 2$ and pooling dimensions(receptive fields of pooling layer neurons) $z \times z = 3 \times 3$. The rationale behind this choice is borrowed from Krizhevsky et al. [16]. Their empirical studies suggest that overlapping pooling helps preventing overfitting. When this is schema used, feature maps obtained from the preceding convolutional layer are halved by each dimension.

Subsampling reduces the dimension of convolution and drastically reduces the computational costs. However, a big subsampling ratio may cause a very big information loss during the process and consequently reduce the discriminative power and results in underfitting. Our experiments suggest that severe underfitting occurs when subsampling ratio is bigger than 3. We prefer subsampling ratio 2 (pooling size 3 with stride 2) over ratio 3 since the models with subsampling ratio 2 and 3 yield similar results while models with subsampling ratio 2 are trained faster due to the cuda-convnet built-in restrictions.

Table3.2: Fixed Hyperparameters

Name	Value
Activation Function	ReLU
Convolution Filter Dimension	5×5
Pooling Window Dimension	3
Pooling Stride	2

Table3.3: Compound Layers

Alias	Num. of Output Maps	Num. of Units
conv1	32	$8n^2$
conv2	32	$2n^2$
conv3	64	n^2
fc	-	64
softmax	32	2

3.1.5 Defining The Network Models

We examine the components of a CNN layer in the previous section. Now, we may define the models analyzed in this thesis. As stated earlier, we investigate the effects of hyperparameters on testing and training error rates. After this fine-tuning, some of the hyperparameters are fixed. These hyperparameters are summarized in the Table 3.2.

Models that we compare share the hyperparameters given in Table 3.2. Using those hyperparameters, we create CNN models adding compound layers (convolution followed by pooling) starting from the base model with 2 compound layers. Therefore, we first define the layers of the most extensive model and then refer to these layers when defining the models. After fixing the above hyperparameters, it is sufficient to specify the number of input and output maps to define a compound layer. Number of input maps are determined by the preceding layer; hence, we specify the number of output maps on each layer. In the most extensive model, there are 3 compound layers (conv1,conv2,conv3) and one fully connected layer (fc) followed by the output layer(softmax). The number of output maps and the total number of units in each layer is given in Table 3.3 in terms of input image dimensions ($3 \times n \times n$).

Note that we specify the number of units in terms of N. This in fact leaves one

Table3.4: Neural Network Architectures of the Study

Hidden Layers	Input Dataset	Num. of Parameters
conv1+conv2	Building Patches	36192
conv1+conv2+conv3	Building Patches	87392
conv1+conv2+conv3+fc	Building Patches	349536
conv1+conv2+conv3	CIFAR-10	84320
conv1+conv2+conv3+fc	CIFAR-10	152856

last model hyperparameter to be specified. Input dimensions are decided by the choice of the dataset. As stated previously, we train our models for both our Building Patch dataset and CIFAR-10 dataset. Under the above considerations, we study the following 6 architectures:

3.2 Training Procedure

Stochastic gradient descent update rule given in the following equations is used with some additions to train the CNN models:

$$\Delta W^{(k+1)} = -\alpha \ \nabla E_{\boldsymbol{W}}(\boldsymbol{x}^{(k)}; \boldsymbol{W}^{(k)}) - \alpha\lambda \boldsymbol{W}^{(k)} + \gamma \Delta W^{(k)}$$

$$W^{l(k+1)} = W^{l(k+1)} + \Delta W^{l(k+1)}$$

where $\boldsymbol{x}^{(k)}$ values are chosen from the training set randomly. The additions are explained in the following subsections.

3.2.1 Using Minibatches

Updating the parameters by the gradient over a single sample is substantially inefficient in terms of GPU usage. Instead, we can exploit the GPU's very fast matrix calculation capabilities by performing the parameter update over minibatches. CUDA-Convnet does it when updating over minibatches of sample size 128. Therefore, the gradient term $\nabla E_{\boldsymbol{W}}(\boldsymbol{x}^{(k)}; \boldsymbol{W}^{l(k)})$ corresponds to the gradient over the kth minibatch $\boldsymbol{x}^{(k)}$.

Table3.5: Initialization Parameters

Layer	Non-Bias Weights	Biases
conv1	$\mathcal{N}(0, 0.0001)$	Set to 0
conv2	$\mathcal{N}(0, 0.01)$	Set to 0
conv3	$\mathcal{N}(0, 0.01)$	Set to 0
fc	$\mathcal{N}(0, 0.01)$	Set to 0
softmax	$\mathcal{N}(0, 0.01)$	Set to 0

3.2.2 Learning Curve and Early Stopping

One of the most straightforward and effective ways for preventing overfitting is *early stopping*. In early stopping, instead of performing updates until a previously determined validation error is reached, algorithm stops when validation error stops improving. We use early stopping as the termination criteria of the stochastic gradient descent algorithm.

3.2.3 Weight Initialization

Weight initialization is one of the most crucial steps of gradient descent algorithm. A good initialization prevents the algoritm from getting stuck at a local minima. Particularly for Neural Networks, initialization has been studied extensively. Nevertheless, training with large datasets is usually enough to take weight vector out of suboptimal solution regions. Therefore, we initialize the weights for each layer according to the distributions given in the table 3.5. Random initialization serves the purpose of symmetry breaking.

3.2.4 Update Rule Parameters

We apply different update rule parameters to different layers. Specifically, weight decay parameter of the output layer(softmax) is much greater than the other ones. It should also be noted that weight decay is not applied to bias paremeters. Learning hyperparameters for each layer is given in the Table 3.6. According to

Table3.6: Learning Rate, Weight Decay and Momentum

Layer	α^l	λ^l	γ^l
conv1	0.001	0.004	0.9
conv2	0.001	0.004	0.9
conv3	0.001	0.004	0.9
fc	0.001	0.004	0.9
softmax	0.001	3	0.9

this table for example, update rule for softmax layer is given as follows:

$$\Delta W^{l(k+1)} = -0.001 \ \nabla E_{\boldsymbol{W}}(\boldsymbol{x}^{(k)}; \boldsymbol{W}^{l(k)}) - 0.003\boldsymbol{W}^{l(k)} + 0.9\Delta W^{l(k)}$$

$$W^{l(k+1)} = W^{l(k+1)} + \Delta W^{l(k+1)}$$

where l is the index of the softmax layer.

3.3 Measurements and Visualization

3.3.1 Precision, Recall and Accuracy

Precision, recall and accuracy are measured using the following formulas:

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN}, Accuracy = \frac{TP + TN}{10000}$$

where TP is the number of correctly labeled positive samples, FP is the number of correctly labeled negative samples and FN is the number of mislabeled positive samples. Accuracy is calculated as the number of correctly labelled samples divided by the number of total samples used in the test of the particular CNN model.

3.3.2 Measuring Redundancy

A convolutional Neural Network detects spatial features and as network transformation proceeds it captures more and more complex features. In the best scenario, we expect these features to be uncorrelated and relevant as much as possible. In this regard, we might consider the CNN classification function as a feature space transformation followed by a linear discriminant function. Let us denote the transformation function with $\mathbf{\Phi}$ and the linear desicion surface normal vector with $\boldsymbol{w}$. Then, likelihood estimation for the class C_1 is

$$p(\mathcal{C}_1|\boldsymbol{x}) = \sigma(\boldsymbol{w} \cdot \mathbf{\Phi}(\boldsymbol{x}))$$

where $\sigma(.)$ is the logistic sigmoid function.

Now, the class likelihood estimation $p(\mathcal{C}_1|\boldsymbol{x})$ is a function of the sum $y = \sum_{i=1}^{dim(\boldsymbol{z})} z_i$ where $z_i = w_i \mathbf{\Phi}_i(\boldsymbol{x}))$. If the correlation between a particular z_k and y is very high, we might conclude that features other than the x_k is redundant. Correlation, in this case can be found by calculating the correlation coefficient between y and x_k if we treat each x_k as a set of observations(dataset samples). Similarly we might look at the k number of features whose sum is highly correlated with y. We approximated this with the following algorithm:

1-Let X be the representation matrix, meaning that each row is a sample representation calculated by the network mappings through the penultimate layer. Let Y be the corresponding matrix of y values.

2- Calculate the correlation coefficient between Y and the each column of $w_i X$.
3- Sort X_is and obtained a sorted list of X_i's.

4- For every j from 1 to N(num of samples)

calculate $S_j = corrcoeff(\sum_{i=1}^{j} w_i X_i', Y)$.

By examining the j vs. S_j graph, we draw striking conclusion that penultimate layer feautures of the 3 and 4 layered models are mostly redundant. These conclusions shall be given in Chapter 5.

3.3.3 Sparsity

There are numerous sparsity measure for different tasks. We use Gini Index [13] whose definition is as follows.

Assuming that the vector $\boldsymbol{v}$ is sorted in ascending order, Gini Index is:

$$GI(\boldsymbol{v}) = 1 - 2 \sum_{i=1}^{dim(\boldsymbol{v})} \frac{|v_i|}{\|\boldsymbol{v}\|_1} \left(\frac{dim(\boldsymbol{v}) - i + 1/2}{dim(\boldsymbol{v})} \right)$$

Gini Index is normalized and a value close to 1 corresponds to high sparsity and vice versa. Index can be 1 only if all the entries of the vector $\boldsymbol{v}$ is zero. It can be 0 only if all the entries are equal and nonzero.

We calculate the GINI index of representations(after the feature mapping of the model is applied) of each sample and take the average of it for 2,3 and 4 layered models. Results are given in the Subsection 4.4.2.

3.3.4 Dimensional Seperation between Classses

A feature can be said to be *class specific*, if it is discriminative feature of a particular class. It is desirable to learn a representation where each feature acts on its own, meaning that features are independent factors of the raw observations. In such a case, one expect that higher layer features are more class specific than the lower layers. We measure the class specificity of a feature set using the representation of a dataset as follows:

2-Divide the dataset into 4 subsets P_1, P_2, N_1, N_2 where first two consist solely of positive samples and the last two consist solely of negative samples. Let us denote the representation matrices with the same names.

3- Define the following operation on a matrix M:

$$H(M) = \frac{1}{n} \mathbf{1}_{1\times n} M^T$$

, where $\mathbf{1}_{1\times n}$ is the n dimensional row vector whose entries are all 1.

4- Calculate the followings:

$$\|H(P_1) - H(P_2)\|,\ \|H(N_1) - H(N_2)\|,\ \|H(P_1 \cup P_2) - H(N_1 \cup N_2)\|$$

Basically, operation $H(.)$ sums the number of activations(nonzero values) of each feature into a normalized vector form. We call this vector form the *activation histogram* of the dataset M Then we calculate the similarity between 3 datasets.

3.4 Summary

In this chapter, we give the parameters of the models that we employ. Our models share some hyperparameters which are convolution filter size, convolution style, pooling window size and stride and activation function. After fixing this hyperparameters, we basically define three models with 2,3 and 4 number of compound (one convolutional and one pooling layer followed by an elementwise ReLU nonlinearity) layers. Measurements that we make conclusions using with are explained in detail as well. These measurements are percentage of redundant features, sparsity based on GINI index and class specificness of a representation.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Datasets

4.1.1 Specifically Designed Building Dataset

HASAT is a large scale remote sensing project which is conducted by METU, HAVELSAN and ASELSAN *. It involves analysis of remote sensing imagery and automatic detection/classification of certain objects/regions. There are several categories of target objects/regions which vary from very large and unstructured targets such as forest areas to structured and smaller objects such as aircrafts.

Projects database consists of a total of 246 satellite images which cover very large areas each. These images are obtained from different satellites (GEOEYE-1, IKONOS, WORLDVIEW). Source images used in our experiments has been collected by satellite GeoEye-1 with a resolution of 0.50 meter square per pixel and in four wavelengths of spectrum which are red, green, blue and near-infrared bands. Database also includes ground truth shape files. Shape files are ,in fact, set of polygons which depicting covering areas of target objects/regions drawn by field experts. These polygons are used for data acquisition in this thesis. An example image tile and several ground truth polygons are shown in Figure 4.1.

For observational quality of "classification by feature existence" phenomena, we create our dataset from a particular region type and with labels according

Figure 4.1: HASAT Database Example
Top:Original image, Bottom:Ground truths, pinks are building, greens are aircrafts.

to a particular object. Our dataset consists of 64x64 three band RGB image patches each gathered from Airport regions. Labeling is done by using "building" ground truth polygons in the following way: If one of the center pixels of a patch coincides with a building ground truth polygon, this patch is taken and labeled to be a positive sample. Those patches whose center pixels do not coincide with any building ground truth polygon, yet one of its pixels coincides with a building ground truth polygon are considered to be ambigious and completely discarded. This is a sound decision for sampling training patches, since labeling of those sort of patches depends on the interpretation of net's classification results by other modules of a hypothetical building detection system. While their true labels are ambigious, it would be unjust to analyze the nets performance on these sort of patches, too. Patches with no intersection with any building ground truth polygon are taken too and considered to be negative examples.

As mentioned above, source images are captured by GEOEYE satellite. Then, they are transformed into three band RGB images and interpolated to the 1 meter square resolution since it is the default resolution for building detection system in the project HASAT.

By this way total of around 20 million patches are gathered from airport regions of 30 GEOEYE images. Avarage ratio of negative samples over positive ones is around 20 making classification chance accuracy roughly equal to 0,95. To obtain, more comparable results for learning the concept *a building part*, we reduce this ratio to 1.5 and making chance accuracy equal to 0,60. We also downsample from original set while keeping sample from each airport region roughly equal.

4.1.2 CIFAR-10 Dataset

CIFAR-10 is a labeled subset of the 80 million tiny images dataset by Torralba et al. [27]. It is created and published by Krizhevsky et al. [15] at 2009 and has been a benchmark data for Convolutional Neural Networks, since then. It contains 60000 32×32 three channel images in 10 classes, 50000 images in each. The dataset is divided into 6 batches of 10000 images. One batch contains 1000

Table4.1: Hyperparameter Testing Models and Their Validation Errors

	filter size	pooling size	pooling stride	pooling ratio	act. function	val. error
model 1	**5**	**3**	**2**	**2**	**relu**	0.10
model 2	**6**	3	2	2	relu	0.10
model 3	**7**	3	2	2	relu	0.10
model 4	5	**3**	**3**	**3**	relu	0.10
model 5	5	**4**	**4**	**4**	relu	0.40
model 6	5	3	2	2	**sigmoid**	0.29
model 7	5	3	2	2	**tanh**	0.27

images from each class and it is the test batch, while others are training batches. The best result, currently known to the author of this thesis is obtained by Lin et al. by their 'Network in Network' model [20].

Note that we use CIFAR-10 dataset for only validation and verification purposes.

4.2 Experiment Setup

4.2.1 The Workbench: Cuda-Convnet

For the entire training and testing processes, we use "cuda-convnet" framework created by Alex Krizhevsky. Cuda-convnet is a very fast C++/CUDA (with a python front-end) implementation of neural networks using the back-propagation algorithm. It supports convolutional layers, pooling layers, fully-connected layers, locally-connected layers, and others.

4.2.2 Hyperparameter Tuning

Models with different hyperparameters are trained by one-leave-out method. These hyperparameter testing models are given in the Table 4.1.

Validation error curves of some of the hyperparameter testing models for the same training sets are drawn in Figure 4.2. As it can be seen, the CNN model with pooling size 4 is completely unable to learn from samples. It is classification accuracy is at the chance level. Since, the information loss is too high, this is pretty much expected. Validation errors of the models with sigmoid and tanh activation functions are reasonably high compared to their relu counterpart. it can be seen inFigure 4.2 that learning progress of the model with sigmoid activation function saturates after 6th epoch. Learning curve of the other models converge to same validation error around epoch 16. We examine same trend in each one-leave-out training case with slight differences between exact validation error values. Subsampling with stride 2 is preferred over stride 3 to keep the feature map sizes as powers of 2. This has an observable effect on the speed of the training on Cuda-Convnet library. It is easily observable on the figure that the model with stride 2 converges much faster than the model with stride 3. We observe the same effect on other one-leave-out cases as well. The models with filter size 7 and 6 are strongly competent with the models with filter size 5. However, the number of parameters in models with filter size 7 and 6 are almost twice the number of parameters in models with filter size 5, since the number of parameters is proportional to the square of the filter size. Therefore, first type of models converges at the half speed of the second type of models in fact.

4.2.3 Training Procedure

Patches from same image tend to have very high spectral correlation.Previous experiments that we perform also clearly show that when test and training sets contain within image patches, models learn spectral features of each image, producing error rates near zero. For that reason we prepare 11 batches of building patches such that patches from a single satellite image are put into the same batch. Each batch contains 10000 patches which we call from now on as *samples*. We choose samples randomly from each airport region in numbers proportional to the region sizes. Within each batch, we keep the ratio of positive samples at 0.6.

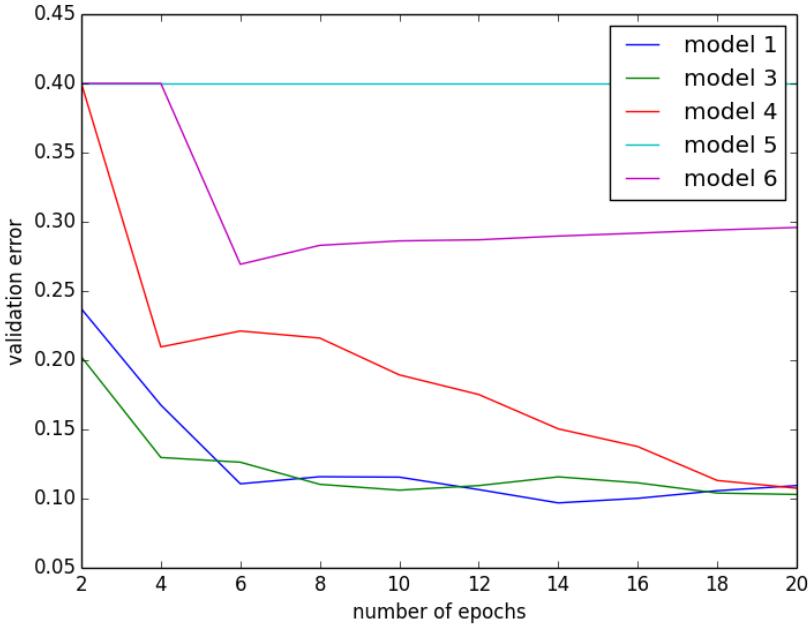

Figure 4.2: Learning Curves of Models with Different Hyperparameters (See Table 4.1 for specifications)

11 test scenarios are prepared in a one-leave-out fashion for each batch. In each scenario, 10 training batches are randomly shuffled to form 10 homogeneous batches so that parameters do not oscillate between different regions which may correspond to features of particular source images or negative and positive samples. One batch is used as validation set. Then, 11 instances of each CNN model are trained corresponding to the 11 test scenario.

Since, CIFAR-10 dataset has one test batch, we trained single instance of each CNN model for CIFAR-10 dataset.

4.3 Classification Results

6 CNN models defined in the section 3.1.5 are trained using the training procedure given in the section 3.2. As we state previously, 11 instances of 3 models corresponding to 2,3 and 4 hidden layer architectures accepting 'Building Patch' Dataset are trained for each batch of the 'Bulding Patch' Dataset. One instances of 3 models on CIFAR-10 dataset is trained. Table 4.2 gives the precision(PR),

recall(RC) and accuracy(ACC) of each test scenarios. Precision and recall for CIFAR-10 instances are not calculated. Average over 11 test scenarios are given as well:

As it can be seen, avarage accuracy is 30 percent better than chance level(60 percent). It is clear that there are lots of rooms for improvement considering the near perfect detection results obtained for buildings in remote sensing domain. It is open to question, nonetheless, whether the convolutional neural network architecture is suitable for obtaining near perfect accuracy.

Most notable statistic is that adding the third and fourth layer does not improve the accuracy. Learning capacity of the 4 hidden layered net theoretically encompasses the 2 and 3 hidden layered ones. Despite that, an imrovement over simpler models are not observed. We see in the section 4.4.1 that third convolution layer and fully connected layer consist of mostly redundant features after training. Why does this happen is open to interpretation. For CIFAR-10 under these hyperparameter assumptions, it is strongly affirmable that 3 is the optimal hidden layer count.

Table4.2: Classification Accuracies

	2 Layers			3 Layers			4 Layers		
	PR	RC	ACC	PR	RC	ACC	PR	RC	ACC
1	0.94	0.96	0.96	0.94	0.98	0.97	0.90	0.98	0.95
2	0.95	0.96	0.97	0.86	0.97	0.93	0.95	0.97	0.97
3	0.91	0.29	0.71	0.93	0.44	0.76	0.90	0.44	0.76
4	0.85	0.93	0.91	0.82	0.93	0.89	0.84	0.90	0.89
5	0.89	0.90	0.91	0.94	0.75	0.88	0.92	0.85	0.91
6	0.91	0.82	0.90	0.92	0.85	0.91	0.84	0.95	0.91
7	0.96	0.59	0.83	0.94	0.86	0.92	0.96	0.71	0.87
8	0.93	0.77	0.89	0.94	0.80	0.90	0.92	0.76	0.88
9	0.81	0.98	0.90	0.80	0.99	0.89	0.79	0.99	0.89
10	0.94	0.81	0.90	0.97	0.75	0.89	0.92	0.83	0.90
11	0.94	0.70	0.86	0.94	0.69	0.86	0.92	0.74	0.87
Avg	0.91	0.79	**0.88**	0.91	0.82	**0.89**	0.90	0.83	**0.89**
CIFAR-10	-	-	-	-	-	0.80	-	-	0.75

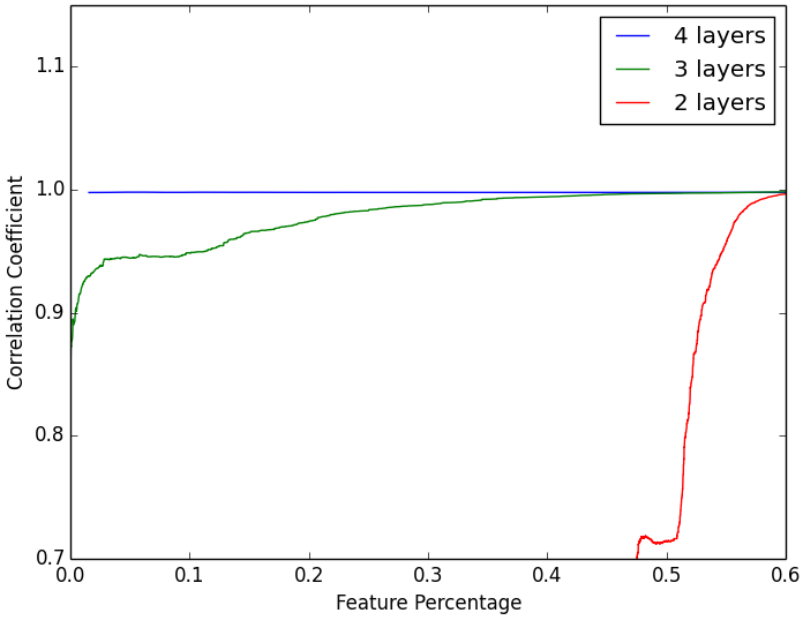

Figure 4.3: Correlation Coefficient between Network Output and Strongest k Features vs k(percentage)

4.4 Measurements

4.4.1 Measuring the Redundancy

Using the algorithm given in section 3.3.2, in the Figure 4.3, we draw the graph of $S_k = corrcoeff(\sum_{i=1}^{k} w_i X'_i, Y)$ vs k. In this way, we measure how close we can linearly approximate network output Y using k features which are correlated with network output most. To be able te compare three models trained by 'Building Patches' dataset, we give the number of features as percentage over total number of features at the penultimate layers of corresponding network models. First observation is that, very small percentage of the penultimate layer features of the 4 layered model is useful. In fact, we notice that, only single feature at the penultimate layer of this model is enough to classify the data with the same accuracy. Considering this fact, we conclude that 4th layer is completely useless in terms of learning high level abstractions. Even the 3 layered model has very high level of redundancy as graph shows that using very small percentage of the features, we can obtain 0.9 correlation with the network

output. 2 layered model has some redundancy as well. It can be seen from the graph that 40 percent of the features of each model is completely redundant. Nonetheless, this is practically normal; since, number of units at the layers are picked to learn possibly overcomplete representations.

4.4.2 Sparsity of the Representations

Following table shows the avarage GINI index of representations of each sample. It can be observed that sparsity of the models are very high.

Table4.3: Sparsity(GINI Index)

	Building Patches			CIFAR-10	
	2 layers	3 layers	4 layers	3 layers	4 layers
GINI Index	0.989	0.928	0.812		

4.4.3 Class Specifity

As it is expected, accuracy histogram distances between two sets of different classes are much higher than the accuracy histogram distances between two sets of the same classes. Figure 4.4 shows this for 3 layered Building Patch model.

Note: Each cell corresponds to a triple in the form of

$$\begin{array}{|c|}\hline \|H(p_1) - H(p_2)\| \\ \|H(n_1) - H(n_2)\| \\ \|H(p_1 \cup p_2) - H(n_1 \cup n_2)\| \\ \hline \end{array}$$

4.5 Summary

In this chapter, we explain the 'Building Patch' dataset that we create and CIFAR-10 dataset that we use for validation and verification purposes. Then, hyperparameter selection rationales are explained in the light of training experiments that we employ. We present performance results of CNN models of our main interest and discuss their implications. Percentage of redundant features,

Table4.4: Sparsity Histogram Distances

	1	2	3	4	5	6	7	8	9	10	11	12
	0.85	0.92	0.89	0.95	0.85	0.87	0.96	0.84	0.86	0.92	0.88	0.92
1	0.81	0.85	0.67	0.82	0.69	0.68	0.74	0.83	0.68	0.73	0.69	0.85
	15.23	**15.26**	**15.31**	**15.39**	**15.30**	**15.17**	**15.46**	**15.24**	**15.49**	**15.35**	**15.45**	**14.37**
	0.90	0.85	0.87	0.86	0.91	0.83	0.91	0.90	0.95	0.92	1.00	0.92
2	0.85	0.65	0.72	0.78	0.70	0.72	0.72	0.64	0.68	0.69	0.62	0.77
	13.62	**13.70**	**13.63**	**13.63**	**13.45**	**13.59**	**13.81**	**13.56**	**13.44**	**13.71**	**13.60**	**17.53**
	0.98	0.79	0.92	0.80	0.86	1.04	0.80	0.78	0.79	0.93	0.86	0.74
3	0.57	1.07	0.60	0.66	0.79	0.62	0.60	0.60	0.67	0.64	0.57	0.63
	14.94	**15.34**	**15.29**	**15.08**	**15.25**	**15.13**	**14.89**	**15.20**	**15.07**	**15.10**	**15.36**	**9.91**
	0.89	0.90	0.92	0.76	0.89	0.95	0.88	0.89	0.87	0.84	0.88	0.82
4	0.76	0.77	0.66	0.70	0.76	0.69	0.68	0.80	0.77	0.69	0.72	0.70
	13.30	**13.25**	**13.09**	**13.15**	**13.42**	**13.36**	**13.40**	**13.32**	**13.27**	**13.41**	**13.49**	**12.25**
	0.89	0.99	0.99	0.89	0.94	0.88	0.97	0.89	0.84	0.84	0.84	0.77
5	0.83	0.64	0.67	0.67	0.66	0.70	0.82	0.68	0.71	0.76	0.71	0.61
	14.69	**14.40**	**14.53**	**14.67**	**14.64**	**14.48**	**14.57**	**14.48**	**14.30**	**14.78**	**14.41**	**16.82**
	0.87	0.86	0.88	0.83	0.91	0.86	0.86	1.00	0.87	0.86	0.86	0.76
6	0.70	0.65	0.64	0.61	0.67	0.70	0.68	0.69	0.79	0.77	0.61	0.90
	14.36	**14.42**	**14.44**	**14.33**	**14.49**	**14.15**	**14.08**	**14.38**	**14.48**	**14.21**	**14.41**	**21.01**
	0.92	0.91	0.86	0.90	0.86	0.88	0.92	0.94	0.94	0.88	0.79	0.78
7	0.88	0.79	0.64	0.78	0.71	0.72	0.77	0.70	0.65	0.59	0.63	0.64
	14.08	**13.99**	**14.04**	**13.96**	**14.36**	**14.07**	**14.22**	**14.11**	**14.03**	**14.02**	**13.97**	**9.96**
	1.09	0.84	0.90	0.84	0.87	0.84	0.80	0.95	0.92	0.85	0.85	0.82
8	0.77	0.67	0.66	0.69	0.67	0.65	0.78	0.66	0.76	0.74	0.76	0.61
	13.23	**13.38**	**13.37**	**13.51**	**13.33**	**13.36**	**13.35**	**13.30**	**13.35**	**13.38**	**13.55**	**17.68**
	0.84	0.85	0.87	0.90	0.87	0.87	0.86	0.85	0.93	0.86	0.82	0.82
9	0.72	0.70	0.76	0.72	0.66	0.88	0.68	0.83	0.63	0.82	0.89	0.64
	13.04	**12.98**	**13.08**	**12.94**	**12.86**	**13.03**	**12.80**	**12.89**	**12.98**	**12.81**	**12.88**	**14.44**
	0.89	1.05	1.06	0.85	0.95	0.92	0.82	0.80	0.83	0.91	0.82	0.82
10	0.66	0.75	0.71	0.71	0.73	0.67	0.61	0.62	0.71	0.85	0.75	0.73
	16.32	**16.66**	**16.61**	**16.47**	**16.55**	**16.42**	**16.57**	**16.33**	**16.53**	**16.49**	**16.26**	**13.84**
	0.79	0.89	0.79	0.83	0.81	0.98	0.82	0.83	0.86	0.89	0.85	0.91
11	0.63	0.69	0.60	0.62	0.62	0.64	0.70	0.63	0.65	0.75	0.69	0.74
	15.27	**15.34**	**15.16**	**15.12**	**15.34**	**15.20**	**15.15**	**15.20**	**15.06**	**15.30**	**15.07**	**10.53**

sparsity values based on GINI index and class specificness of representations are given in detail and discussed as well.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this study, we explore the possibility of employing the Deep Learning idea to a dataset created from satellite images. Our main goal is to provide a first report on the feasibility of using Deep Architectures for satellite image classification tasks. Convolutional Neural Network Classification model is chosen as the most successful representative of so-called Deep Architectures in supervised settings. Considering the size of satellite images, CNN architectures are natural choices, since they are both scalable and practical and most of the recent success on object recognition tasks is earned by them. Since we have a very large dataset, we are able to exploit the learning power of Convolutional Neural Networks. It is important to note that Convolutional Neural Network's suitability to parallel computation and fast GPU-based machine learning libraries made it possible to use CNNs to their full extent. In this regard, we can argue that GPU based implementation is very crucial to construct a CNN based detection model with state-of-the-art performance on satellite images if it is even possible. Using the cuda-convnet library, training time of a single CNN model was around 90 minutes. Without parallel computation, this time may increase to several days even to weeks.

One of the biggest problems of an architecture like CNN is hyperparameter selection. In fact, we may say that the generic CNN model is merely a composition of some ideas. On top of those ideas, one has to choose many hyperparameters

including number of layers, filter sizes, subsampling sizes, activation function etc. We test each of these hyperparameters by fixing the other ones and comparing accuracy results of candidate values decided by rule of thumbs. On the other hand, stochastic gradient descent hyperparameter are used without a validation. One may argue that hyperparameters have a really big impact on the performance of neural networks. However, our work aims to provide a first observation on using the CNN model on remote sensing. We basically aim to present the potential of the CNN models and identify their shortcomings and strengths through several measures.

Performance results suggest that it is possible to obtain a considerable accuracy rate even with only 2 convolutional layers. It is easy to conclude that such a model can only capture mildly complex geometrical shapes. The striking result was that we could not beat the record of 2 layered models with 3 and 4 layered models, while this is not the case for CIFAR-10 dataset. These results suggest that improvements over our results are highly susceptible to dataset preparation.

Three measurements have been presented in the study. Each of them uses the dataset projected onto the representation space. Using the informativeness measure we are able to gain more insight on the apparent shortcomings of 3 and 4 layered CNN models. The finding was that 3rd and 4th layer features are mostly redundant reducing the model to practically to 2 hidden layer. Again this is not the case for CIFAR-10. Representations are very sparse according to GINI index. There is a trend towards less sparsity when adding more layers; nonetheless, this was due to the fact that low layers in CNNs are designed to detect a relatively small feature in the whole image. It is practically normal that a feature is detected at small percentage of the image and consequently output maps of such features become very sparse. Using the activation histogram between positive sample sets, negative sample sets and mixed sets, we observe that classes were separated mostly by feature activation rather than feature magnitude.

Conclusion is that Convolutional Neural Network models designed for classification of " building part" concept are able to perfom % 30 percen higher than the chance level even when they are trained using relatively simple hyperparameter

selection processes.

5.2 Future Work

This thesis is considered as a first step on the analysis of the realizability of a robust, large scale real-time satellite imagery object detection system based on Deep Architectures. The Dataset which we perform our experiments on consists of same size patches each belonging to one of the two classes which we strictly define. Furthermore, it does not reflect the real ratio of positive samples over negative samples. We obtain % 90 percent accuracy results over this dataset. From the current state, there are several serious steps to be made through the ultimate goal. First of all, classification performance is required to be improved and there is a room for % 10 percent improvement. Possible ways to achieve this subgoal are to use more advanced techniques to prevent overfitting, to use the whole images for unsupervised learning and providing more classes. It is probable that using labeled samples from several classes help in learning a larger variety of high level features. Second, a larger scale segmentation and CNN hybrid classification system is required to detect class objects with various sizes. Third, system must be able to modify itself to recognize new classes and even reveal hidden classes.

REFERENCES

[1] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artif. Intell.*, 210:78–122, May 2014.

[2] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[3] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[4] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montréal, and M. Québec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[5] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[6] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[7] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, pages 315–323, 2011.

[8] G. E. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006.

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[10] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[11] J. Håstad. Almost optimal lower bounds for small depth circuits. In *RANDOMNESS AND COMPUTATION*, pages 6–20. JAI Press, 1989.

[12] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[13] N. Hurley and S. Rickard. Comparing measures of sparsity. *Information Theory, IEEE Transactions on*, 55(10):4723–4741, Oct 2009.

[14] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153, Sept 2009.

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

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

[18] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society.

[19] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[20] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[21] W. S. McCulloch and i. j. v. n. d. t. u. p. a. p. l. Pitt, Walters year=1943.

[22] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In J. Fürnkranz and T. Joachims, editors, *ICML*, pages 807–814. Omnipress, 2010.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[24] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio. A quantitative theory of immediate visual recognition. *PROG BRAIN RES*, pages 33–56, 2007.

[25] H. Siegelman and E. D. Sontag. Neural nets are universal computing devices. Technical Report SYCON-91-08, Rutgers Center for Systems and Control, Rutgers University, New Brunswick, NJ, 1991.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, and A. Rabinovich. Going deeper with convolutions. Technical report, 2014.

[27] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, Nov 2008.

[28] A. C.-C. Yao. Separating the polynomial-time hierarchy by oracles. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 1–10, Oct 1985.

[29] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535, June 2010.

[30] M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025, Nov 2011.

[31] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.