DEEP CONVOLUTIONAL NEURAL NETWORKS WITH AN APPLICATION TOWARDS GEOSPATIAL OBJECT RECOGNITION

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

BY

EMRECAN BATI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

DEEP CONVOLUTIONAL NEURAL NETWORKS WITH AN APPLICATION TOWARDS GEOSPATIAL OBJECT RECOGNITION

submitted by **EMRECAN BATI** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen Dean, Graduate School of **Natural and Applied Sciences**	____________
Prof. Dr. Gönül Turhan Sayan Head of Department, **Electrical and Electronics Eng.**	____________
Prof. Dr. A. Aydın Alatan Supervisor, **Elec. and Electronics Eng. Dept., METU**	____________

Examining Committee Members:

Prof. Dr. Uğur Halıcı Electrical and Electronics Engineering Department, METU	____________
Prof. Dr. A. Aydın Alatan Electrical and Electronics Engineering Department, METU	____________
Prof. Dr. Gözde Bozdağı Akar Electrical and Electronics Engineering Department, METU	____________
Prof. Dr. Fatoş Yarman Vural Computer Engineering Department, METU	____________
Dr. Emre Başeski HAVELSAN A.Ş.	____________

Date: September 05, 2014

ABSTRACT

DEEP CONVOLUTIONAL NEURAL NETWORKS WITH AN APPLICATION TOWARDS GEOSPATIAL OBJECT RECOGNITION

Batı, Emrecan
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. A. Aydın Alatan

September 2014, 70 pages

The passion of human-being to invent intelligent systems becomes more and more meaningful day by day, as the data captured every second by artificial sensors needs to be examined and classified for many applications. The processing of ever-increasing amount of data by defining information explicitly seems nearly impossible, regarding the variability and the amount of the information, which reveals the need for intelligent systems that are capable of learning. Deep learning is a set of algorithms that attempts to find a hierarchical representation of the input data by trying to mimic the way human brain captures the critical aspects of excessive sensory data, to which it is exposed to every second. Convolutional neural networks, which are trainable learning structures, are also biologically inspired from the receptive fields in visual cortex. In this thesis, the performance of convolutional neural networks are investigated for an application towards geospatial target detection and classification from satellite images. Based on the experiments, it is observed that the utilization of preprocessing, dropout, i.e. dropping neurons randomly in the training phase, and rectified linear unit as the activation function improves the classification rate, significantly. However, the application of this deep classifier on satellite images still yields high false alarm rate, possibly due to insufficient number of training data.

Keywords: Deep learning, convolutional neural networks, backpropagation, satellite images, geospatial target detection

ÖZ

JEO-UZAMSAL NESNE TANIMAYA YÖNELİK BİR UYGULAMA İLE
DERİN EVRİŞİMLİ SİNİR AĞLARI

Batı, Emrecan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. A. Aydın Alatan

Eylül 2014 , 70 sayfa

Her saniye yapay sensörler aracılığıyla elde edilen verilerin pek çok uygulama için incelenmesi ve sınıflandırılması gerektiği göz önünde bulundurulduğunda, insanoğlunun akıllı sistemler yaratma tutkusu günbegün daha da anlamlı hale gelmektedir. Çeşitlilik ve miktarı düşünüldüğünde, bu verilerin, bilginin haricen açıklanması yoluyla değerlendirimesi imkansız görünmektedir ve bu durum, öğrenme yetisine sahip akıllı sistemlerin gerekliliğini ortaya çıkarmaktadır. Derin öğrenme, girdi verisinin kademeli gösterimini bulmaya çalışan yöntemler bütünüdür ve her saniye oldukça fazla duyu verisine maruz kalan insan beyninin bu veriden önemli bilgileri çıkarma yolunu taklit etmeye çalışmaktadır. Ayrıca, evrişimli sinir ağları da biyolojiden esinlenmiş eğitilebilen öğrenme yapılarıdır ve görsel korteksteki alıcı alanlarının suretini yaratmaya çalışmaktadır. Bu tezde derin öğrenme yöntemleri ile eğitilmiş evrişimli sinir ağlarının uydu görüntülerinde jeo-uzamsal hedef bulma ve sınıflandırma konusundaki başarımı araştırılacaktır. Önişleme, eğitim esnasında rastgele nöron eksiltme ve düzeltilmiş doğrusal birim kullanımının sınıflandırma oranını belirgin şekilde arttırdığı deneylerde gözlemlenmiştir. Buna rağmen, kullanılan derin sınıflandırıcı, muhtemelen yetersiz eğitim verisi sebebiyle uydu görüntülerinde yüksek miktarda yanlış alarm vermektedir.

Anahtar Kelimeler: Derin öğrenme, evrişimli sinir ağları, geriyayılım, uydu görüntüleri, jeo-uzamsal hedef bulma

to my mother ...

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere thanks to my supervisor, Prof. Dr. A. Aydın Alatan for his constant support, understanding and trust in me. His wisdom guide me to new research areas that otherwise I would be unaware. In addition, I would like to thank for his help and contributions to this thesis. I have always felt privileged to be a graduate student of such a brilliant person.

I would like to thank Prof. Dr. Gözde Bozdağı Akar for her understanding and guidence throughout my graduate studies and HASAT project.

I would also like to mention that being a member of Multimedia Research Group was a marvelous experience thanks to the people I met. I am thankful to Beril Besbınar, Emin Zerman and Yeti Ziya Gürbüz for their invaluable helps while writing this theses. I am also grateful to Yağız Aksoy and Ozan Şener for all the time we spent. I also acknowledge Akın Çalışkan, Ilker Buzcu, Ömürcan Kumtepe, Selin Böncü, Ahmet Saraçoğlu, O. Serdar Gedik, Alper Koz.

I am deeply thankful to Scientific and Technological Research Council of Turkey (TÜBİTAK) and Havelsan for their financial support.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xvii

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation 1

1.2 Scope and Outline of the Thesis 9

2 RELATED WORK 11

2.1 Early Attempts 11

2.1.1 Perceptron 11

2.1.2 Neocognitron 14

2.2 Feedforward Networks 16

2.2.1 Learning Procedure 19

2.2.2 Backpropagation 23

2.2.3 Convolutional Neural Network 24

2.3 Reborn of the Deep Learning 26

2.3.1 Autoencoder Types 30

Autoencoder 30

Sparse Autoencoder 31

Denoising Autoencoder 31

Contractive Autoencoder 32

3 CONVOLUTIONAL NEURAL NETWORK FOR DEEP LEARNING 33

3.1 A Layer of CNN 35

3.1.1 Convolution Sublayer 36

3.1.2 Non-linearity Sublayer 37

3.1.2.1 Sigmoid Unit 38

3.1.2.2 Rectified Linear Unit (ReLU) . . . 38

3.1.3 Local Response Normalization Sublayer 39

3.1.4 Pooling Sublayer 40

3.2 Improving Backpropagation: Rule of Thumbs 41

3.2.1 Preprocessing 41

3.2.2 Weight (Filter) Initialization 43

3.3 Experiment Setup 43

3.3.1 CIFAR-10 Dataset [50] 44

3.4 Analysis of The Results 45

4 APPLICATIONS IN REMOTE SENSING 51

4.1 Airplane Detection 52

4.2 Classification of Airport Region Targets 55

5 CONCLUSION AND FUTURE WORK 61

5.1 Summary 61

5.2 Conclusion 62

5.3 Future Work 62

REFERENCES 63

APPENDICES

LIST OF TABLES

TABLES

Table 3.1 Acronyms corresponding to fully connected architectures . . . 46

Table 3.2 Misclassification rates for experiments with fully connected architectures 47

Table 3.3 Acronyms corresponding to convolutional architectures 48

Table 3.4 Misclassification rates for experiments with fully connected architectures 49

Table 4.1 Confusion table for airplane detection problem 54

Table 4.2 Confusion table for classification of airport region targets of resized images 55

Table 4.3 Confusion table for classification of airport region targets . . . 55

LIST OF FIGURES

FIGURES

Figure 1.1 Example illustrating the need for a new representation of the data 2

Figure 1.2 General Learning Machine 3

Figure 1.3 Comparison of the (a) distributed and (b) local representation 6

Figure 1.4 Illustration of hierarchical representation [60] 6

Figure 1.5 Example computation path 7

Figure 1.6 Hierarchical structure of the mammalian brain [94] 9

Figure 2.1 McCulloch-Pitts model of a neuron 12

Figure 2.2 Revised model of a neuron 13

Figure 2.3 A typical architecture of a neocognitron [26] 15

Figure 2.4 An example that shows the reponse of a neocognitron which is trained for hand-written digits [27] 16

Figure 2.5 Artificial Neuron 17

Figure 2.6 Illustration of (a) feedforward and (b)recurrent networks . . 17

Figure 2.7 Capacity increase with increase in depth and breadth [52] . . 18

Figure 2.8 Illustration of (a) low bias - high variance, (b) high bias - low variance and (c) low bias - low variance 21

Figure 2.9 The energy calculation flow of the unsupervised training . . . 28

Figure 2.10 Stacking layers. (a) demonstrate a single layer while (b) shows the stacking of layers 29

Figure 3.1 Illustration of (a) Fully Connected Neuron, (b) Locally Connected Neuron and (c) Locally Connected & Weight Sharing Neuron 34

Figure 3.2 An overview of Convolutional Neural Network 35

Figure 3.3 Overview of a single layer 36

Figure 3.4 Non-linearty functions: (a) Logistic Sigmoid Function (b) Hyperbolic Tangent Sigmoid Function (c) Rectified Linear Function . . 39

Figure 3.5 Overview of a single layer 42

Figure 3.6 Sample images from all classes for CIFAR-10 [50] 44

Figure 3.7 Performance change with respect to the training epoch (a) for the Y|3F|R|Y and (b) 3C|2F|32|F|555 architectures 50

Figure 4.1 An example set of images from the dataset, (a) Airplane Labeled Images and (b) Non-Airplane Labeled Images 53

Figure 4.2 Learning curve for airplane detection 54

Figure 4.3 The target detection algorithm run on (a) the 1024x1024 image that contains an airport. (b) the detection result of the corresponding image where red represents false detection, green represents true detection and orange represents missed targets. 57

Figure 4.4 An example set of images from the classification dataset with and without resizing during cropping 58

Figure 4.5 Learning curve for classification of airport region targets of resized images 59

Figure 4.6 Learning curve for classification of airport region targets . . . 59

LIST OF ABBREVIATIONS

AE	Autoencoder
ANN	Artificial Neural Network
BP	Backpropagation
SAE	Sparse Autoencoder
CAE	Contractive Autoencoder
CNN	Convolutional Neural Network
FCNN	Fully Connected Neural Network
FNN	Feedforward Neural Network
PCA	Principle Component Analysis
PSD	Predictive Sparse Decomposition
ReLU	Rectified Linear Unit
ZCA	Zero Phase Component Analysis

CHAPTER 1

INTRODUCTION

Huge amount of data are captured every second by artificial sensors, such as cameras and microphones with the rapid advancement in technology. Making the computers or machines to be able to have a knowledge and understanding of this excessive sensory data with the minimum amount of human interaction is a must in Artificial Intelligence, whereas the analysis of the sensor data itself has always been a hot topic in Computer Vision. The intervention of human can be decreased by the means of implicit learning of knowledge from data, where learning is considered as making generalizations of the experience, i.e., available data, to predict the outputs of new, unseen data. A more formal definition of machine learning is made by Tom M. Mitchell as "A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [70].

1.1 Motivation

There are many different approaches to the learning problem, such as decision tree learning [76], association rule learning [1], artificial neural networks [67, 82], inductive logic programming [77], support vector machines [18], Bayesian networks [73], sparse dictionary learning [2], representation learning [7] etc. The readers are referred to [71] for the current state and foreseen future paths of machine learning.

The aforementioned well-known learning algorithms generally are not well performing on raw data, since data are not well separated in the input domain where the raw data is actually collected. For a good classification, one seeks for a new representation which enables separating the different classes, Figure 1.1 demonstrates such a hypothetical representation of the data and separating hyperplane. In Figure 1.1.a random two pixels intensities (raw data) are drawn on the axis for different classes, face and non-face. Figure 1.1.b, shows a new representation which encodes the probability of having eye and mouth is given. A separating hyperplane can easily be drawn in the new representation which explains the importance of the representation.

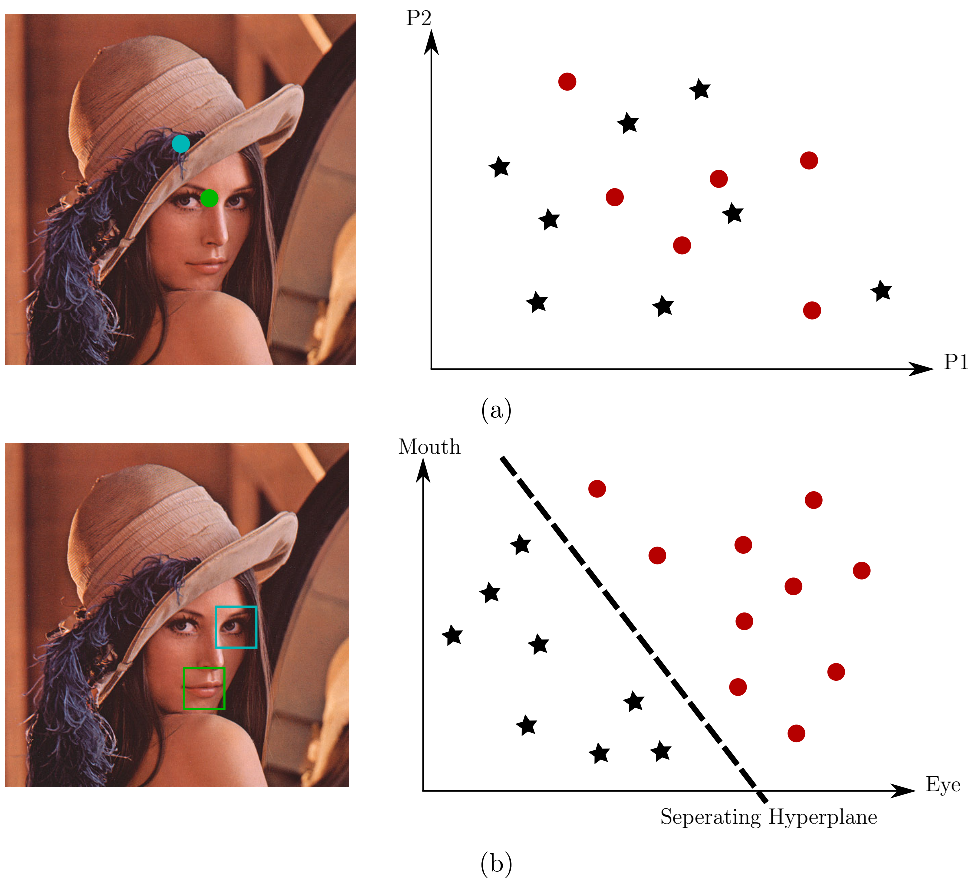


Figure 1.1: Example illustrating the need for a new representation of the data

The general framework for a learning machine can be represented as in Figure 1.2. The machine computes $\widehat{O}_p = f(X_p, W)$ for input X using parameters W. The aim of the learning algorithm is to minimize a loss function representing the discrepancy between the system output, $\widehat{O}$, and the desired output, O for the input pattern X.

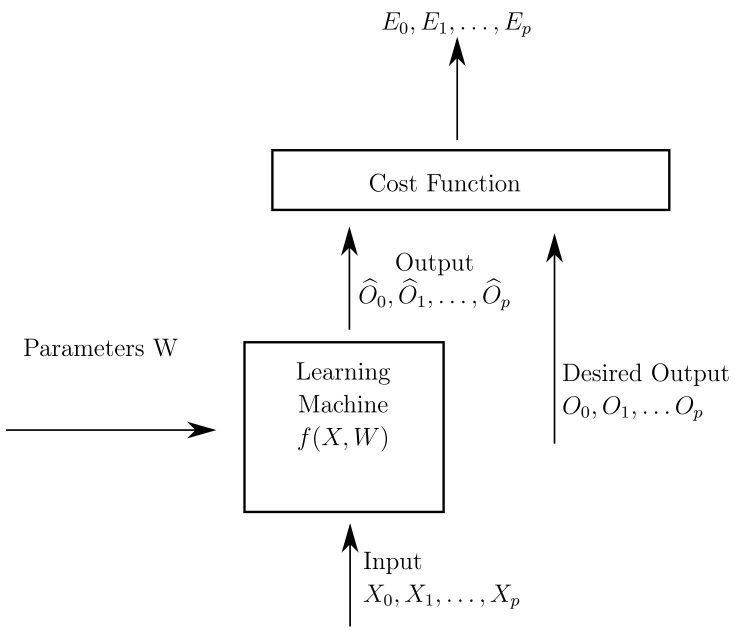


Figure 1.2: General Learning Machine

Pattern recognition, where the computer algorithms are used to reveal the regularities in data automatically in order to classify them, which is a popular research topic [11] and due to the requirement of a new representation, general framework for pattern recognition problem usually contains a stage of feature extraction followed by a trainable classifier. The state of the art techniques also employ a mid-level feature extraction, such as Bag of Words (BoW), which is a composition of simpler features, that inherently creates a hierarchy of representation. The aim of extracting features is both to reduce the dimensionality of the raw input, by removing *unrelated* information, and obtain a new representation more separable. The dimensionality of the raw data can be reduced without loss of much information, just by decorrelating the data, since the underlying process of the data is believed to generate correlated samples of the same physical world, which is captured by some generic sensors. In other words, the observations of a physical process are densely sampled, therefore highly redundant, since this physical process, the underlying cause of these observations, have a lower dimensionality than the data itself.

Dimensionality reduction is much more important for learning algorithms that solely depend on the *smoothness* prior which utilizes the general information that if example data, d', is *close* to another data d then the information of d is more correlated and informative than other data *far away* from d. Kernel learning methods, e.g. Gaussian Kernel, and SVM [19], are backed by this principle so are affected by the *curse of dimensionality* more. The curse of dimensionality is used to express the fact that the number of examples needed for efective training increases exponentially, while the dimensionality of the data increases linearly [72]. Although the generalization is not affected by the dimension but the variation of the function with the data [7].

One very usual way of dimensionality reduction is to create hand-crafted features, such as Scale Invariant Feature Transform(SIFT) [65], Speeded Up Roboust Features [6] for image data, Mel-Frequency Cepstral Coefficients(MFCC), Linear Predictive Cepstral Coefficients(LPCC) [89] for speech data etc., which may be highly application-dependent and require time and expert knowledge. This kind of dimensionality reduction, that uses hand-crafted features, imposes the human knowledge of the necessary and critical information by explicitly programming it which may not be possible for some problems. Even if it is possible, it requires designing new features for different types of data, such as image and speech data, and application, such as speaker identification and speech-to-text.

The combination of the desire for a better representation and the complication of *feature engineering* for each task leads to algorithms that can learn the representations. The algorithms can be grouped into two, namely *deep learning* which is composed of multiple nonlinear transformation of the data [9], and *shallow learning*, where the input-output relation is found directly or with less number of layers. Matching local templates like kernel machine can be seen as two layer shallow architecture, where the first layer calculates distance and second layer combines the values. The depth of the learning algorithm refers to the number of nonlinear layers. There are different motivations behind the deep architectures.

Firstly, the use of a hierarchical computational model yields compact representation of the learned function, $f(X, W)$ where X is the input and W is the tunable model parameters. The compactness of the deep architecture is presented in [43] for logical circuits, in which logical circuits with $k+1$ layers needs exponentially more computational elements to be represented in k layers, which means exponential increase in the number of parameters. The models with many computational blocks, which also use many parameters, fails to *generalize.* The learned function is said to generalize well if it is able to estimate the desired value of the output for unseen data with low error. A function with many computational blocks or parameters can memorize the given training set input-output pairs but that memorization can result in the loss of the generality.

The expressive power of the learning method can be measured by the required number of parameter for a number of input regions. One-hot representations, in which only one region is active (non-zero) at a time, like nearest-neighbor, Gaussian SVM $\mathcal{O}(N)$, the parameters can generate $\mathcal{O}(N)$ regions in the input space; for example, nearest-neighbor creates the Veronoi tesellation of the input space. On the other hand, autoencoders or multilayer neural networks can generate at most $\mathcal{O}(2^N)$ input regions using $\mathcal{O}(N)$ parameters. The ability to generate exponentially more regions is the virtue of the fact that multilayer neural networks are utilizing the *distributed representation.* Dividing the space into overlapping regions with each division representing an underlying a factor is called distributed representation. An illustration of the comparison of a neural network(distributed representation), Figure 1.3a, and a nearest-neighbor(local representation), Figure 1.3b, given in the Figure 1.3. P1, P2 and P3 are the parameters. The parameters are the labeled data available in the training set for nearest-neighbor while the parameters for neural network is corresponding to the parameters of the line equation. The input domain is partitioned with 3 parameters into 3 regions with nearest-neighbor and with 3 parameters into 7 regions with neural network. In the distributed models, underlying factors are used to represent the data so that the information from each example is utilized for every new data, not the information from the examples that are in the basin of the new data. Regions that do not have an example in the training set can be created by the utilization of distributed representation, since the factors can be combined in such a way that is meaningful but do not appear in the training set.

This property is the core factor that prevents the curse of dimensionality [9].

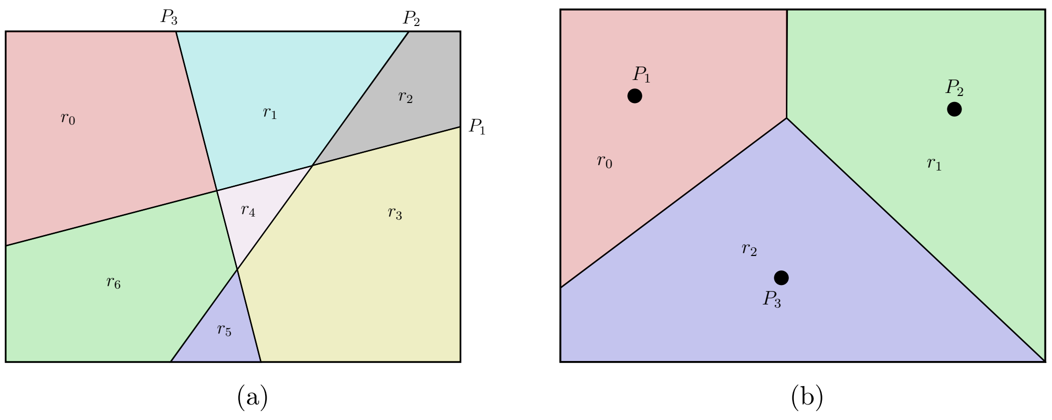


Figure 1.3: Comparison of the (a) distributed and (b) local representation

In addition to the above fact, increase in the depth have statistical and computational advantages. The hierarchical representation will construct an abstraction where first layers corresponds to low level simple features and upper levels evolved from the lower level feature and have more abstract meanings. The abstraction also leads to invariance. For example, for convolutional neural networks that are examined in Chapter 3, pooling mechanism is a way of generating more invariant features. Figure 1.4 depicts the learned features of deep neural network trained with face images [60].

The reuse of computational blocks will generate an exponential gain with linear increase in depth. This can be observed easily that m layers with k computation block in each layer, that is total of $k \cdot m$ computational element can compute k^m different functions or paths an example of the computation path for the given structure is illustrated in Figure 1.5.

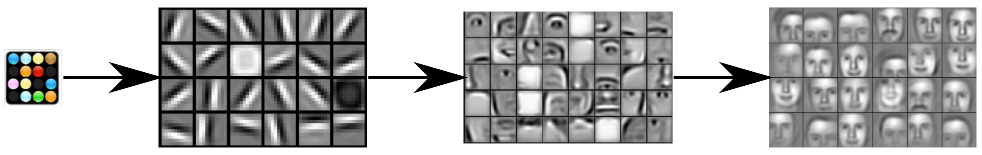


Figure 1.4: Illustration of hierarchical representation [60]

Finally, in deep learning paradigm, there is a strong inspiration from mammalian brain. The human brain is actually a gigantic computing machine. A *neuron* is the structural element of nervous system and it processes and carries the information as electrical signals in general. The information is transmitted through

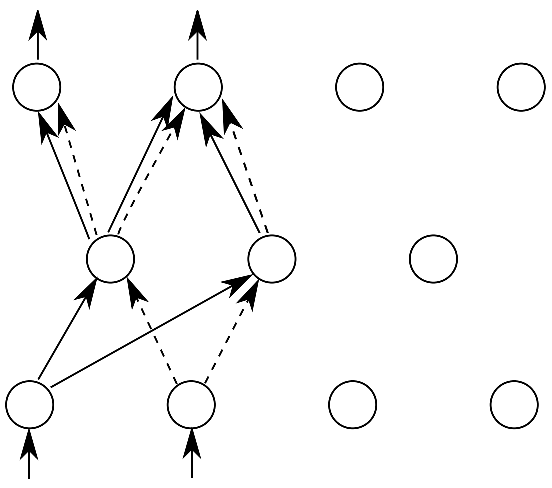


Figure 1.5: Example computation path

the different neurons by the help of chemical reactions at *synapses* where the neurons interacts each other. The most recent estimated number of neurons in the human brain is 86 billion according to [4], although 100 billion had been a common number [34, 46]. [4] reports that there are approximately 16 billion neurons in human cerebral cortex, which is responsible for memory, concious, perception, thoughts, etc. More interestingly, another research [88] which actually assumes 10 billion neurons in human cortex, claims that there are 60 trillion synapses or connections in human cortex.

In addition, the neurons are much slower than silicon logical gates by 5 to 6 orders in magnitude [34]. In other words, neural events occur in milliseconds, whereas operations are possible in silicon gates in nanoseconds. However, it still takes longer to solve simpler problems in high technology computing machines than how fast human process and reacts sensory data.

It is also known that the human brain has the ability of adaptation [15, 22] by forming new synapses between neurons or modifying the existing ones. In addition, brain is composed of organizations in different scales and each of these organizations has different functions at different levels [88]. For example, *local circuits* are groups of neurons that gathered for some localized functional operations and *interregional circuits* are assemblies of local circuits and entail in different parts of the brain.

It is not only the physiology of the brain that shows a hierarchical structure, but also the way it perceives the outside world. In 1990s, some experiments were conducted to examine the ventral stream in adult monkeys [48, 49, 92]. Ventral stream, together with dorsal stream, constitutes the visual system, where ventral stream is related to the object identification and recognition and dorsal stream is involved in the spatial localization of objects [33]. In these experiments, adult monkeys were the subjects and they are examined in anaesthetized contiditions in order to better understand the way of discriminative learning from very few samples. For example, in [48], monkeys are examined while being exposed to images with reduced complexity. Starting with the images of complex three dimensional objects, images are simplified by eliminating a feature at each step and the response of different neurons are observed throughout this series of images stimuli. In [49], monkeys are trained to discriminate 28 moderately complex shapes and the responses from trained and untrained monkey are investigated to understand how the discriminative power, visual system is attained. It is found that the responsive cells recorded from trained are significantly larger than the ones obtained from the control set. In addition, different cells had different subsets of stimuli that they responded and there were partial overlaps between those subsets of cells. The experiments also showed that receptive field size, the complexity of the preferred stimulus and tolerance to position and scale changes gradually increases in different parts of the visual system.

Supported later works [80], the visual system is concluded to demonstrate a hierarchy of increasingly sophisticated representations, which is illustrated in Figure 1.6.

All these results indicate that the power of human brain comes from its structure with such a massive interconnections and its ability to adapt its surrounding environment by hierarchical representations, which are the key properties neural network research takes into account and tries to design systems by mimicking them.

Multilayered feed forward neural networks are promising deep learning architectures, in which each layer is composed of the weighted sum operation followed

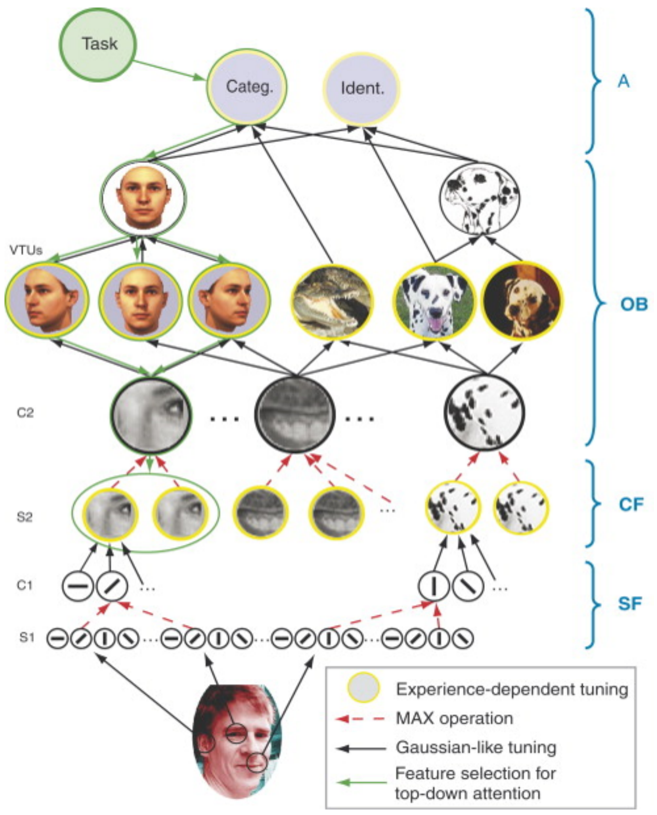


Figure 1.6: Hierarchical structure of the mammalian brain [94]

by a nonlinear activation imitating a neuron. Parameters, namely weight and bias, are trained using *backpropagation* algorithm which is first introduced in 1981 [85]. A multilayer neural network with weight sharing and local connectivity has a special name, Convolutional Neural Network (CNN) [55]. CNN architecture utilizes the general prior that each sample's (a pixel for image) correlation with other is inversely proportional with the spatial distance of the samples,that is applicable for image, video and many other signals. The concepts of the multilayer feed forward neural networks will be examined in Chapter 2 in more details.

1.2 Scope and Outline of the Thesis

An architecture of feed forward neural network is examined, namely, Convolutional Neural Network(CNN). CNN is applied to the target detection problem from satellite images, in the scope of this thesis, whereas pretraining the models using unsupervised techniques, such as Restricted Boltzmann Machines (RBM), second degree optimization techniques for training neural networks and recurrent neural networks are not in the scope of this thesis.

In Chapter 2, the attempts of creating biologically inspired models of deep architectures presented in the basics of deep neural networks. Convolutional Neural Networks are analyzed by using a well-known dataset in Chapter 3. The application to the satellite images are presented in Chapter 4. The thesis concludes with the last chapter.

CHAPTER 2

RELATED WORK

2.1 Early Attempts

The early attempts for machine learning using biologically inspired techniques can be summarized in two fundamental approaches, namely Perceptron and Neocognitron.

2.1.1 Perceptron

The mathematical model of an artificial neuron is first proposed by McCulloch and Pitts in 1943 [68]. This model, illustrated in figure 2.1, first obtains the linear combination of m-dimensional inputs $z_1, z_2, \dots z_m$ with corresponding weights $w_1, w_2, \dots, z_m$ and then compares this weighted sum to a constant threshold leading to a binary result. Therefore, the McCulloch-Pitts model is also known as *linear threshold gate*. In this highly simplified model, the inputs are allowed to have only binary values, such as 1 for *true* and 0 for *false* and combined with binary weights, which makes the model useful for only logical operations. This first mathematical model, that is not able to learn, lacks the flexibility for complex tasks; however, it was a substantial and inspiring beginning.

Moreover, in 1949, the neural bases of learning is explained by psychologist Donald O. Hebb in his well-cited book "The Organization of Behavior". Although there was no explicit mathematical statement, the concept known as *Hebb's synapse*, which explains how learning process modify the cellular structure in

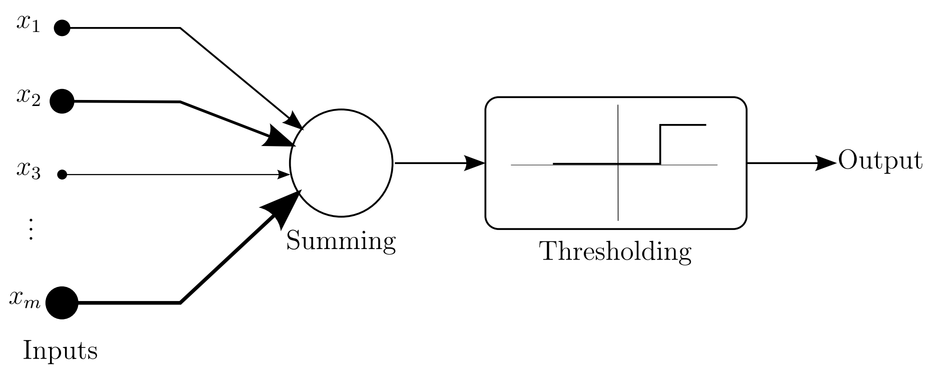


Figure 2.1: McCulloch-Pitts model of a neuron

brain, inspired the neural modellers. The exact statement was "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased", which implies the increase in strength between nearby cells which are active simultaneously [35].

Using the nonliner neuron model by McCulloch-Pitts and Hebb's postulate, Frank Rosenblantt changed the description of artificial neurons by *perceptron* in 1958 [83]. The model was similar to McCulloch-Pitts model with the difference of adjustable weights. The perceptron realizes two-class classification by comparing the linear combination of inputs with adjustable weights, w, with an addition of external bias b, i.e., $\sum_{i=1}^{m} x_i w_i + b$, to the hard threshold. The weights are generally normalized in the range of $(0, 1)$ and two-class classifier can be summarized as shown in Equation 2.1 where w is the weight term and b is the bias term.

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} x_i w_i + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.1}$$

$$\tag{2.2}$$

Note that the bias term can also be treated as a weight to be learned corresponding to an input that is always have the value one, concatenated to the input vector. In other words, we can redefine the input and weight as $\mathbf{x} = [1, x_1, \ldots, x_m]^T$,

$\mathbf{w} = [b, w_1, \dots, w_m)]^T$ and $\mathbf{u} = \sum_{i=1}^{m} x_i w_i + b = \mathbf{w}^T\mathbf{x}$ by increasing the dimension of input space by one. Figure 2.2 illustrates the redefined model.

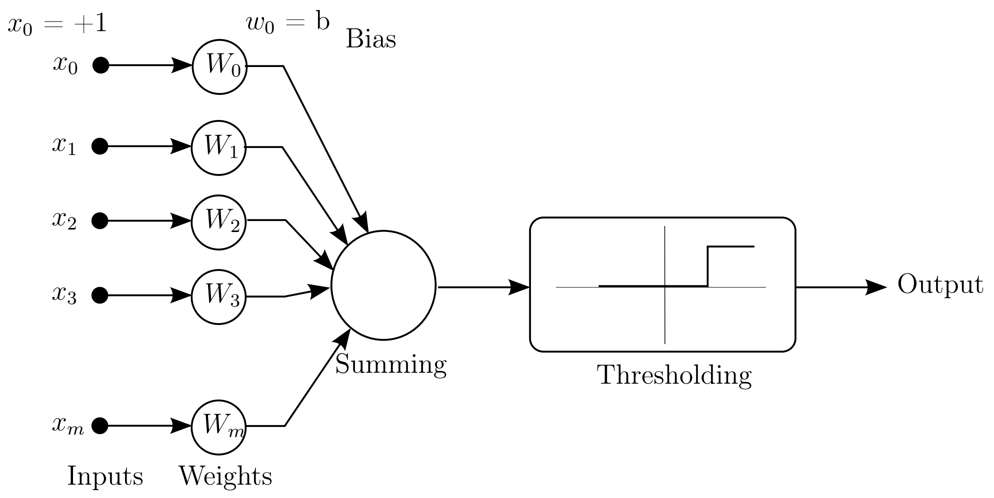


Figure 2.2: Revised model of a neuron

The adjustable weights in the perceptron make the model able to learn and supervised-training is achieved with a numerical algorithm. Suppose that the training set of S samples is represented with $\{\mathbf{x}(j), d(j)\}_{j=1}^{S}$, where $\mathbf{x}(j)$ is a m-dimensional input vector and $d(j)$ is the desired class of $\mathbf{x}(j)$. The learning algorithm initializes the weights and bias for all S samples. Then, for each iteration, a random sample is chosen from the training set and the corresponding class of the sample attained by perceptron is calculated with these initial weights and bias. If the resulting class label is correct for the randomly selected sample, the algorithm does not make any change in weights; but otherwise, the weights are updated. The update rule for an arbitrary iteration is given in Algorithm 1 for the randomly chosen training sample $(\mathbf{x}(j), d(j))$ where η represents the learning rate. The iterations continue until convergence , which is proved to be present for certain type of data in 1962 [84].

In 1969, Marvin Minsky and Seymour Papert analysed Rosenblatt's perceptron in their book and stated that the perceptron is capable of solving only linearly separable functions [69]. This attack about the limitations of the perceptron decreased the interest in the artificial neural network (ANN) research.

Algorithm 1 Perceptron Weight Update

```
repeat
    Randomly select a sample $x(j)$
    if $f(\mathbf{x})(j)$ (Eq. 2.1) not equals to $d(j)$ then
        $e = d(j) - f(\mathbf{x})(j)$
        $\Delta w_i = \eta\, e\, x_i(j)$
        $w_i = w_i + \Delta w_i$
    end if
until convergence
```

2.1.2 Neocognitron

David H. Hubel and Torsten Wiesel realized the *cat experiment* [95] by inserting microelectrodes to the primary visual cortex of a anesthetized cat to observe the changes with the different lighting patterns on the screen in front of the cat. According to the result of this experiment, some neurons, which are defined as *simple cells*, are found to respond differently to edges at different orientations. In addition, these neurons are also observed to react differently to the light and dark patterns [95]. On the other hand, the term *complex cells* is used for the neurons that gave similar responses to line patterns regardless of from where they are applied. In 1962, they made additional comments on the visual cortex which they believed to have a complex structure with interconnections between simple and complex cells [40].

This interconnected model of visual system inspired the *neocognitron*, which is a hierarchical multilayered neural network that might be considered as the first attempt to generate hierarchical representations [87]. The neocognitron is proposed in 1980 and developed in the following years by Fukushima [25, 26, 28, 29], who introduced the convolutional layers during these works. The algorithm tries to imitate the mammalian visual perception by realizing a convolution of input pattern with a rectangular unit composed of adjustable weights. The aim is to generate a hierarchical representation of the input pattern, which may be interpreted as feature extraction. Figure 2.3 shows the typical architecture employed by a neocognitron.

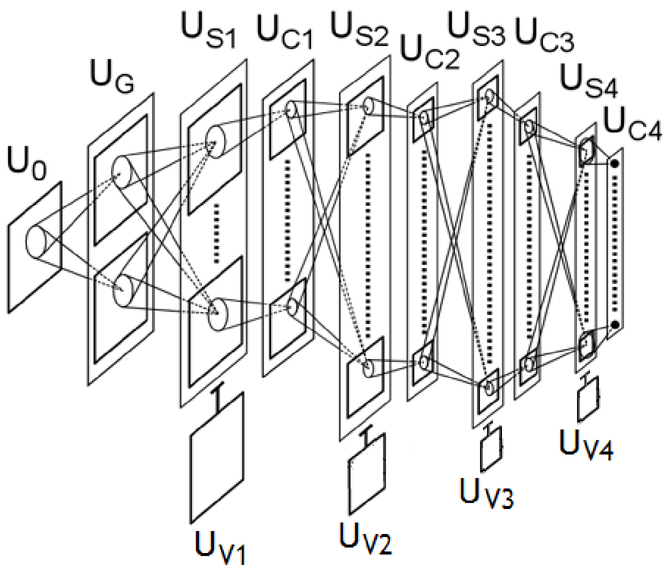


Figure 2.3: A typical architecture of a neocognitron [26]

The first layer of the architecture represents the photoreceptors of the retina whereas the ordered connections between layers is inspired by receptive field, which indicates the volume in the physical world inside which the light can trigger the corresponding photoreceptor. The model itself is composed of *S-cells* and *C-cells* which resemble simple and complex cells, respectively. Each cell in higher layers are connected to a limited area on the preceding layer which is called local connection.

Analogous to human visual perception [61], first layers in the architecture are expected to capture low level features, such as edges and blobs by the help of S-cells, whose input connections are variable and therefore adjusted during learning. On the other hand, C-cells have invariable and fixed input connections and they are expected to compensate the positional errors in the features captured by S-cells.

The training of neocognitron can be supervised or unsupervised and learning means modifying the interconnection of S-cells. The training allows the network to self-organize their architecture according to two principles. According to the first principle, among cells located in a small area, only the cell with highest response is allowed to strengthen its connections proportional to the intensity of the response. The second principle states that the winner cell that has the highest response also controls the growth of neighbor cells, and this principal is said to be introduced in order to preserve translational symmetry.

Figure 2.4 exemplifies the response of several layers of a neocognitron which is trained to recognize hand-written digits. Even though the weights are not learned in a global manner and rather set by local learning rules, the layer responses clearly indicates that the neocognitron has the ability to learn hierarchical representations. Down-sampling between layers during hierarchy is realized by spatial averaging.

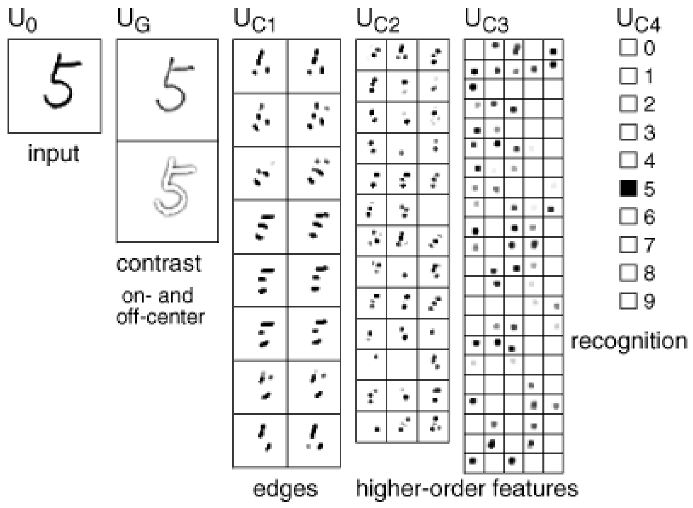


Figure 2.4: An example that shows the reponse of a neocognitron which is trained for hand-written digits [27]

2.2 Feedforward Networks

Neural Networks can be seen as a graph which is composed of computational blocks, each of which computes weighted sum of the inputs and fed it to a nonlinearity, a computational model, namely artificial neuron model, is given in Figure 2.5. Artificial neuron models the synaptic transfer by using weighted sum of the inputs and activation function models the probability of firing a neuron.

The computation graph that can be represented as an acyclic graph is so called feed forward, whereas the graphs that have cyclic connections are denoted as recurrent neural network. Feedforward (Figure 2.6a) and recurrent (Figure 2.6b) networks are shown in the Figure 2.6.

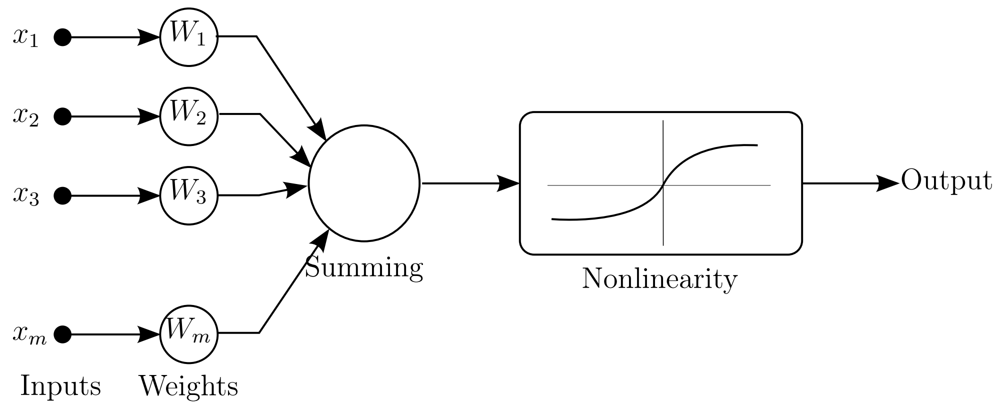


Figure 2.5: Artificial Neuron

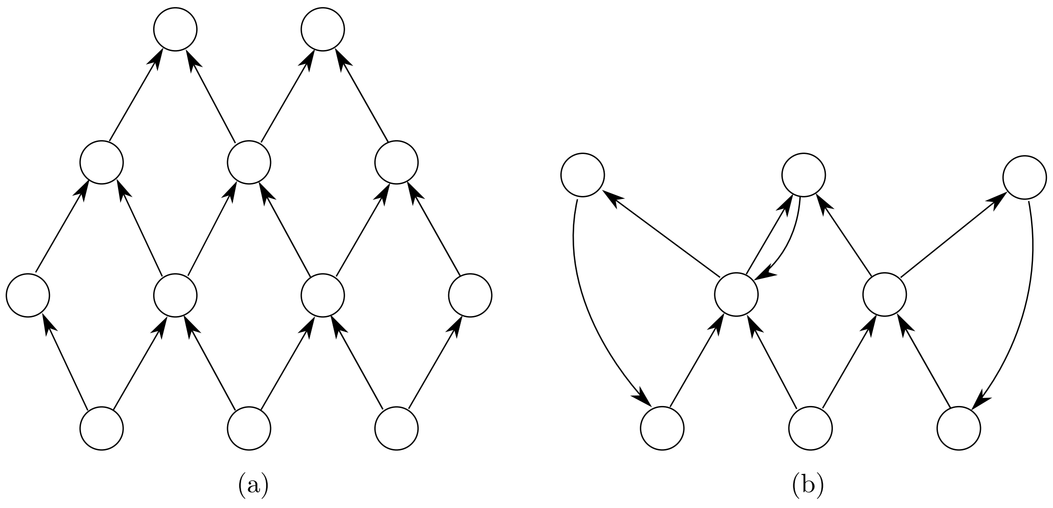


Figure 2.6: Illustration of (a) feedforward and (b)recurrent networks

The *capacity* of a network can be adjusted by appending more neurons to a layer or by adding extra layers that are connected to previous layer. The capacity of a network is defined as the ability of the network to distinguish varying cases. As stated in Chapter 1, increasing the depth generates exponential gain in terms of compactness and computational complexity. This is illustrated in Figure 2.7, a neuron can generate a hyperplane that can separate linearly separable classes, while increasing the neuron number and depth, more complex separating hyperplanes can be generated.

Neural networks operate in two different modes; first is the *forward propagation*, where the input is propagated towards upper layers to generate an output, the second is *backpropagation*, where the error is propagated downwards in the hierarchy to learn the parameters. Backpropagation algorithm is detailed in Section

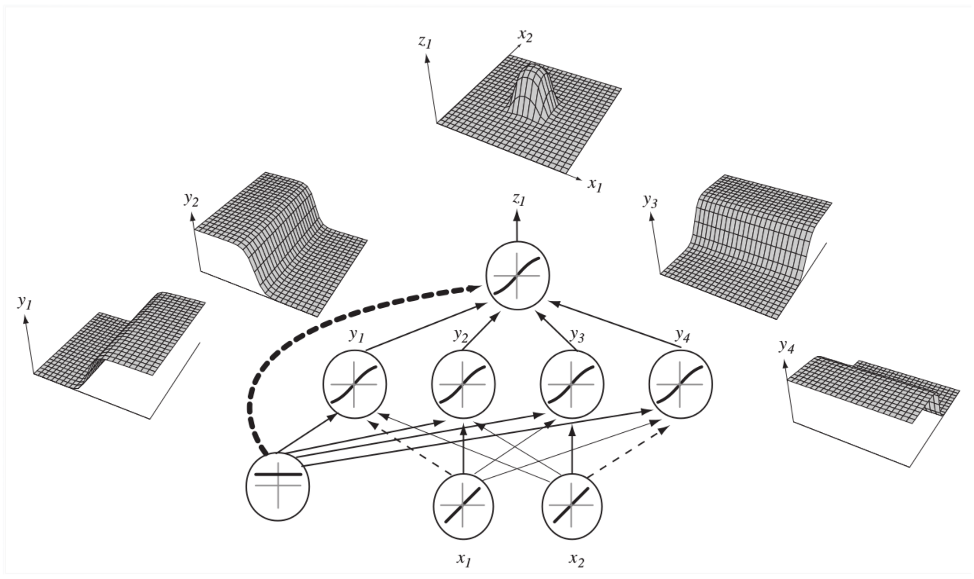


Figure 2.7: Capacity increase with increase in depth and breadth [52]

2.2.2.

In this formulation, k^{th} input to i^{th} neuron at the j^{th} layer is denoted as h^k_{j-1} and the preactivation, $p^k_j(i)$, that is the weighted sum of inputs where weight vector is w^i and bias b^i is calculated as in Equation 2.3. The result after activation, that is after the nonlinear function $s(\bullet)$,which is generally logistic sigmoid or hyperbolic tangent function, is applied is h^k_j. N and M in the Equation 2.3 are the width of the layer and the number of inputs, respectively. h^k_0 is the raw input data.

$$p_j = \begin{bmatrix} - & w^1 & - \\ - & w^2 & - \\ & \vdots & \\ - & w^N & - \end{bmatrix}_j \begin{bmatrix} | & | & & | \\ h^1 & h^2 & ... & h^M \\ | & | & & | \end{bmatrix}_{j-1} + \begin{bmatrix} b^1 & b^1 & \dots & b^1 \\ b^2 & b^2 & \dots & b^2 \\ \vdots & \vdots & \ddots & \vdots \\ b^N & b^N & \dots & b^N \end{bmatrix}_j \tag{2.3}$$

$$h_j = s(p_j) = \begin{bmatrix} | & | & & | \\ s(p^1) & s(p^2) & \dots & s(p^N) \\ | & | & & | \end{bmatrix}_j \tag{2.4}$$

At the last layer, for classification, a *softmax* activation is used in the literature. The output of the network is treated as the posterior, the class label given input, $p(c|X)$. Since the output is probability, it should obey the axioms of the probability, converting from energy of being a class to a probability can be done using softmax activation, as in Equation 2.5, where $f_i(X)$ is the network output at the i^{th} neuron on the top layer and p_i the probability associated to that neurons class as follows:

$$p_i = \frac{\mathrm{e}^{f_i(X)}}{\sum\limits_j \mathrm{e}^{f_j(X)}} \tag{2.5}$$

2.2.1 Learning Procedure

A general framework of a learning machine is depicted in Figure 1.2. Multilayer feed forward network is not an exception. For this framework, weight, W, and bias, b, parameters are learned so that average training error, E_{train}, is minimized. The error can be selected as the mean squared error for regression problems and negative log likelihood for classification and detection problems. The error is usually computed as the mean squared error (MSE) for regression problems, since MSE punishes the misclassification according to the wrong label. For example, in hand-written digit classification problem, classifying the digit '3' as '4' or '8' corresponds to different amount of errors when MSE used. On the other hand, negative log likelihood error does not take the difference of the true label and the wrong label into consideration and regards the same problem as the probability of the output being '3'. Therefore, the error in neural network framework is computed as negative log likelihood for the classification and detection problems.

The standard second order nonlinear optimization techniques are not well suited for neural networks, due to the requirement of the computationally expensive Hessian of the error. The algorithm generally used for optimization is gradient descent shown below, where (W_i^t, b_i^t) is the tunable parameters of the system, namely, i^{th} weight at time (learning iteration) t and i^{th} bias at time t, η is the learning rate and $\nabla_{(W_i,b_i)}E_{train}$ is the gradient of the energy with respect to parameters.

$$E_{train} = -\sum_{i} \log(E(\mathcal{O}_i, \hat{\mathcal{O}}_i)) \tag{2.6}$$

$$(W_i^t, b_i^t) = (W_i^{t-1}, b_i^{t-1}) - \eta \nabla_{(W_i, b_i)} E_{train} \tag{2.7}$$

The reuse of computational blocks by constructing layers are also bringing computational efficiency while computing gradients. The gradient of error with respect to parameters is not calculated separately but the error propagated from top layer to bottom step by step. Each step uses the previously calculated value. The procedure of calculating gradient by propagating error is so called *backpropagation*.

The complete pass through of the training set, which is called *epoch*, is necessary to calculate the *true* gradient in Equation 2.7. The complete pass through is referred as *batch* learning as the entire batch of the data is used to calculate the weight update, whereas the *stochastic* learning is used to describe the gradient computation using a random sample from the training set for each weight update. The *mini-batch* learning is the term used for the gradient calculation from a small number of examples from the dataset. The use of mini-batches are advantageous for two main reasons: it is much faster than the batch learning and speed does not depend on the training set size.

The convergence of the gradient descent is not guaranteed, since the function is highly nonlinear and nonconvex. There are many local minimas and plateaus that the iterative algorithm can trap. The use of stochastic learning adds noise to the gradient computation, since not all of the sample gradients are averaged as in the Equation 2.6, but only the gradients that are generated by the randomly selected samples are averaged. The added noise, caused by the randomness, will help to escape from the trap [58].

The concept of *momentum* can be used to smooth the gradient and improve the speed of convergence [8,36,58]. Momentum is the name of temporally smoothing the gradient as in the Equation 2.8, where β is the free parameter that impose the amount of smoothness.

$$\nabla x_t \leftarrow \beta \nabla x_t - (1 - \beta) \nabla x_{t-1} \tag{2.8}$$

The parameters are updated in the opposite direction of the gradient with magnitude scaled with the learning rate, η. η is an important hyperparameter, generally selected in the range of $[10^{-6}, 1]$. An advised strategy for updating learning rate is *annealing* [8]. The learining rate is decreased at each iteration with the Equation 2.9 where t is the iteration, T is the total iteration or limit of the early stopping, and η_0 is the initial learning rate.

$$\eta_t = (1 - \frac{t}{T})\eta_0 \tag{2.9}$$

The weight update depends on the training error; however, the main aim of learning is to generalize well to unseen data which can not be measured during training. The theoretical analysis decomposes the generalization error into two: variance and bias. Let f be the function to be learned and g be the function learned with the available dataset $\mathcal{D}$. The variance is the diversity of the learned functions with different sets of training data, while the bias is the difference between the output of the target function f and the average output of different learned functions g. The illustration of different bias and variance conditions are given in the Figure 2.8, where the target function is f and the turquoise colored region is the varieties of learned functions with different training sets.

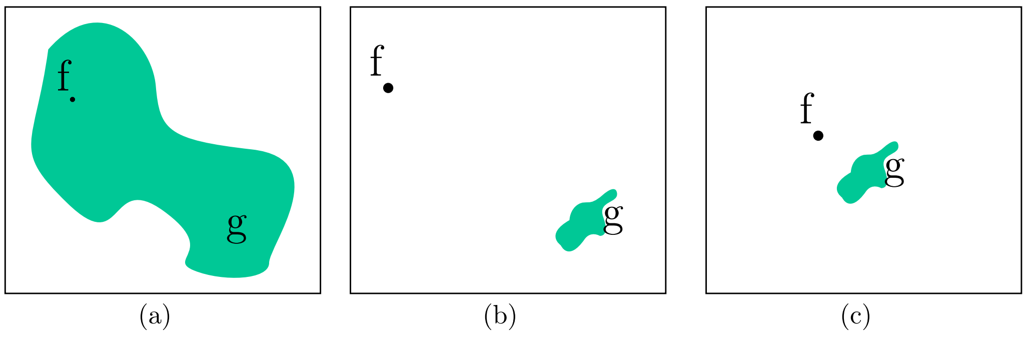


Figure 2.8: Illustration of (a) low bias - high variance, (b) high bias - low variance and (c) low bias - low variance

The bias can be lowered by using a network having a larger capacity. Therefore, increasing depth or width will lower the bias, but possesses the risk of memorization. The network with larger depth or width can memorize the training data due to the increased expressive power, which will lead to the loss of generalization, and therefore, increase in the variance.

There are ways to decrease the variance. The first technique that one can come with is the *regularization*. Regularization is a term added to the energy, that is minimized, imposing the parameters not to grow in magnitude. A revised training energy with an additional regularization term is given in Equation 2.10.

$$E_{train} = -\sum_i \log(E(\mathcal{O}_i, \hat{\mathcal{O}}_i)) + \sum_j (W_j)^2 \tag{2.10}$$

A simpler way of applying the same constraint is *early stopping*. The learning procedure is iterative and the network fits better to the training data at each iteration. Stopping before reaching a minimum is called early stopping. The equivalence of early stopping and l_2 regularization is shown in [90].

Another method used lately is *Dropout* [38], which also prevents overfitting by dropping neurons in the network randomly at each training iteration. The pruning of network decreases the effective size of network. Dropout randomly drops neurons that will be an input to the next layer. Since the presence of the inputs together is not guaranteed the factors learned can not depend on many inputs together , in other words, dropout prevents co-adaptation by randomly dropping neurons. Omitting neurons also forces the network to learn *disentangled* factors. Disentangled factors means the factors that are uncorrelated from others. Entangling factors might be regarded as the coordinates of the manifold. For example, image of an object can be retrieved from different distances and orientations and in addition to the problem of different poses, the appearance of the object may present differences. However, the set of possible images of the object constitutes a manifold in a higher dimensional space of all possible images with the same size and classification is achieved when the underlying variation factors, which combine to generate the data, and therefore, can be seen as the coordinates of the manifold, are disentangled.

Another way to limit the overfitting is to find a network architecture that have less parameters but keeps the capacity approximately equal. Such a goal can be achieved by using general priors applicable to the area of interest. Convolutional Neural Network is an example of this pattern.

2.2.2 Backpropagation

Backpropagation is a method for calculating gradient in computation graphs using well-known *Chain Rule* in calculus. Chain rule is used to calculate derivative of function $f(\vec{x})$ with respect to y, where $\vec{x}$ is a vector with each element x_i being a function of y, and $\frac{\partial f(\vec{x})}{\partial x_i}$ calculated before, is shown in Equation 2.11.

$$\frac{\partial f(\vec{x})}{\partial y} = \sum_i \frac{\partial f(\vec{x})}{\partial x_i} \frac{\partial x_i}{\partial y} \tag{2.11}$$

For feedforward neural network, first, the gradient of the training error is calculated with respect to topmost layer parameters found. Then, training error with respect to the input of the topmost layer is calculated, which is analogous to calculating $\frac{\partial f(\vec{x})}{\partial x_i}$. Afterwards, the chain rule is used to find the training error gradient with respect to the parameters of the second layer from the top. The pseudo code for the backpropagation is given in Algorithm 2, where $\nabla_\theta L$ denotes the gradient of the negative log energy $-\log(E(\mathcal{O}, \hat{\mathcal{O}}))$ with respect to θ and m for the total number of layers.

Algorithm 2 Backpropagation

```
Forwardpropagate the input $X$ to calculate $\hat{\mathcal{O}}$
Compute the output gradient before activation $\nabla_{p_{L+1}} L$
for k from m+1 to 1 do
    Compute gradients $\nabla_{W_L} E$, and $\nabla_{b_L} L$
    Compute gradient of layer below $\nabla_{h_{k-1}} L$
    Compute gradient of layer below before activation $\nabla_{p_{k-1}} L$
end for
```

The algorithm starts with computing output gradient with respect to the input of *softmax* layer, where the gradient is given in Equation 2.12a. y is the true class and $\hat{\mathcal{O}}$ is a vector each element of is the probability calculated for input X to be belonging to class c where the total number of classes is C. $\vec{e}(y)$ which has 1 at y^{th} element and 0 elsewhere is a vector. For each layer, gradients with respect to parameters are calculated using the gradient computed at the previous step as given in Equation 2.12b and 2.12c for a fully connected layer. Finally, the gradient that will be used for the layer below is calculated in two steps: first, the gradient with respect to activation of the layer below using Equation 2.12d, then the gradient with respect to the preactivation is calculated with Equation 2.12e, where the nonlinear activation function is denoted as $s(\bullet)$ and derivative of the activation function denoted as $s'(\bullet)$.

$$\nabla_{\vec{p}^{(L+1)}(\mathbf{x})} L = -(\vec{e}(y) - \hat{\mathcal{O}}) \tag{2.12a}$$

$$\nabla_{\mathbf{W}^{(k)}} L = (\nabla_{\vec{p}^{(k)}} L)\ \vec{h}^{(k-1)} \tag{2.12b}$$

$$\nabla_{\mathbf{b}^{(k)}} L = \nabla_{\mathbf{p}^{(k)}} L \tag{2.12c}$$

$$\nabla_{\mathbf{h}^{(k-1)}} L = \mathbf{W}^{(k)^{\mathsf{T}}} \nabla_{\mathbf{p}^{(k)}(\mathbf{x})} L \tag{2.12d}$$

$$\nabla_{\mathbf{p}^{(k-1)}} L = (\nabla_{\mathbf{h}^{(k-1)}} L \odot [\ldots, s'(p^{(k)}(\mathbf{x})_j), \ldots] \tag{2.12e}$$

2.2.3 Convolutional Neural Network

The bias can be lowered by increasing the capacity of the network; however, the important point is to find a way to decrease the bias, while preserving the variance. The use of general priors, that are not application specific and can be used without loss of generality, is a feasible and highly-applied way to achieve this goal. Smoothness is the prior behind many local learners and it means that a function is learned such that $x \approx y$ implies $f(x) \approx f(y)$. On the other hand, the general prior behind distributed learning is *multiple underlying factors*, which states that the data generating distribution is affected by multiple factors and these factors can be used to make generalizations in many configurations. Finally, *hierarchical representations* assume that the world can be described by using concepts that generate hierarchy, there is simple information in low levels and abstraction increases while going to higher levels in the hierarchy, which is the motivation behind the deep learning.

Convolutional Neural Networks(CNN) represents another general prior. Video or image data have a temporal and spatial coherence, that is, each voxel or pixel is correlated inversely proportional to the temporal and spatial distance between voxels or pixels. *Locally connected* neurons are used to capture that correlation, since the neighboring pixels are correlated most, only the pixels that are in the vicinity of the neuron is connected to the input of that neuron. The *weight sharing*, that is each locally connected neuron, have the same weights decrease the number of parameters but also imposes another general prior that same feature should be searched everywhere on the image. The well-known descriptors also operate on the same principle and the spatial location is irrelevant while describing the point.

CNN is similar to Fukushima's Neocognitron [25] in the sense that both uses convolution filters as weight sharing, locally connected neurons and a pooling layer after the activation. In addition, CNN is heavily inspired from the biologically plausible models without trying to directly copy it [57].

CNN has an interesting property that by the general priors introduced, 6-7 layered CNN can be trained with randomly initialized weights, while fully connected networks of the same depth are nearly impossible to train. The *fundamental problem* of deep learning, diminishing or exploding gradients, affects fully connected networks more. The increase in the depth of the network makes backpropagation harder, since the gradient is calculated by multiplying gradients at each layer. If one layer is saturated, i.e, gradient is close to 0, then it can not backpropagate the gradient, so diminishes or if the gradient in a layer is too large then it backpropagete this large value so will explode the gradient. The small number of fan-in, the number of inputs to a neuron, is used as an explanation of CNN not being affected from the fundamental problem as much as fully connected networks. An overview of the CNN is given in Figure 3.2 and more details can be found in Chapter 3.

2.3 Reborn of the Deep Learning

The deep architectures are hard to train due to many local minima and plateaus. Increase in the depth also increases the number of local minima, so the difficulty of training deep architectures is more than the shallow ones [24]. The difficulty is to find a *good* local minima that generalizes unseen data well. The backpropagation stucks on the path from higher layer to lower layer, if the gradient is zero on one layer. This leads to initial layer not learned and the overall network trapped to a local minima. That fact made neural networks with many layers to perform worse than the networks with few layers although the deeper neural networks have a larger capacity [53]. The shallow networks also promoted by the fact that superposition of sigmoidal function, which is 2 layer network with a sigmoid activation, is a global approximator that can approximate continuous functions [20]. The statement is generalized in [39] such that the feedforward neural network architecture makes the neural networks global approximator not the specific choice of the nonlinearity. These circumstances reduced the popularity of deep architectures until 2006.

In 2006 a breakthrough is initiated [10, 37, 79, 93]. The commonality of these works is *greedy layerwise training* that each layer of the network is trained using an unsupervised criterion and stacked on top of each other to generate a deep architecture. Layerwise training has two stages, in the first stage the given input X is encoded as Y, in the second stage, generative stage, the optimal code Y^* is decoded to reconstruct the input X, and the reconstructed version is $\hat{X}$.

Unsupervised learning involves an energy minimization in which energy of the training samples are minimized. Trying to minimize the energy of the samples is not enough because that will end up with a totally flat, collapsed, energy surface which is not informative. The energy of the unobserved data should be maximized while minimizing the energy of the observed data.

There are two paradigms for layerwise training. First approach uses probabilistic graph modeling, such as Restricted Boltzmann Machines(RBM) [37], while other branch works on the computational graph interpretation of the neural network.

These models differ from each other by selecting the mechanism to minimize energy of the observed data, increase the energy of the unobserved data, and how to parametrize this mechanism.

Most of the layerwise training techniques can be examined in a unified energy based framework [78]. The energy of the system is the combination of *encoder energy*, *decoder energy* and *code energy*. The discrepancy between the code Y and the optimal code Y^* is called the encoder energy while the discrepancy between the input X and reconstructed input $\hat{X}$ is named as decoder energy. Some methods imposes constraints on the optimal code Y^* which is so called code energy, an example of which may be sparsity. The energy calculation flow is given in Figure 2.9. The layers can be stacked by removing decoder and feeding the code Y to a new encoder as depicted in Figure 2.10. This architecture can be used to initialize a multilayer neural network and the training can continue with supervised training which explains the reason for unsupervised learning of the data distribution being also called unsupervised *pretraining*.

The probabilistic models define a joint probability of visible units (input) and hidden units (code), $p(x, y)$. Codes are obtained by using the posterior $p(y|x)$. Training is finding the model parameters that maximizes the likelihood of the training data. By the definition of the probability, $\int p(x)\mathrm{d}x = 1$ where $p(x)$ represents probability density function, so maximizing the probability of training data will pull down the probabilities of unseen data. Posterior distribution, $p(y|x)$, and the *partition function* that normalizes the total probability to one, Equation 2.13a, are intractable. The latent variable probabilities are used to generate a code by taking expectation or getting the most likely value. Probabilistic models can be formalized as an energy problem by using Gibbs energy as shown in 2.13b, where X is the input (visible units) and Y is the latent variable (hidden representation, code).

$$\mathrm{Z} = \int_x \mathrm{e}^{-\beta \mathrm{E}(x,Y)} \mathrm{d}x \tag{2.13a}$$

$$\mathrm{P}(X, Y) = \frac{\mathrm{e}^{-\beta \mathrm{E}(X,Y)}}{Z} \tag{2.13b}$$

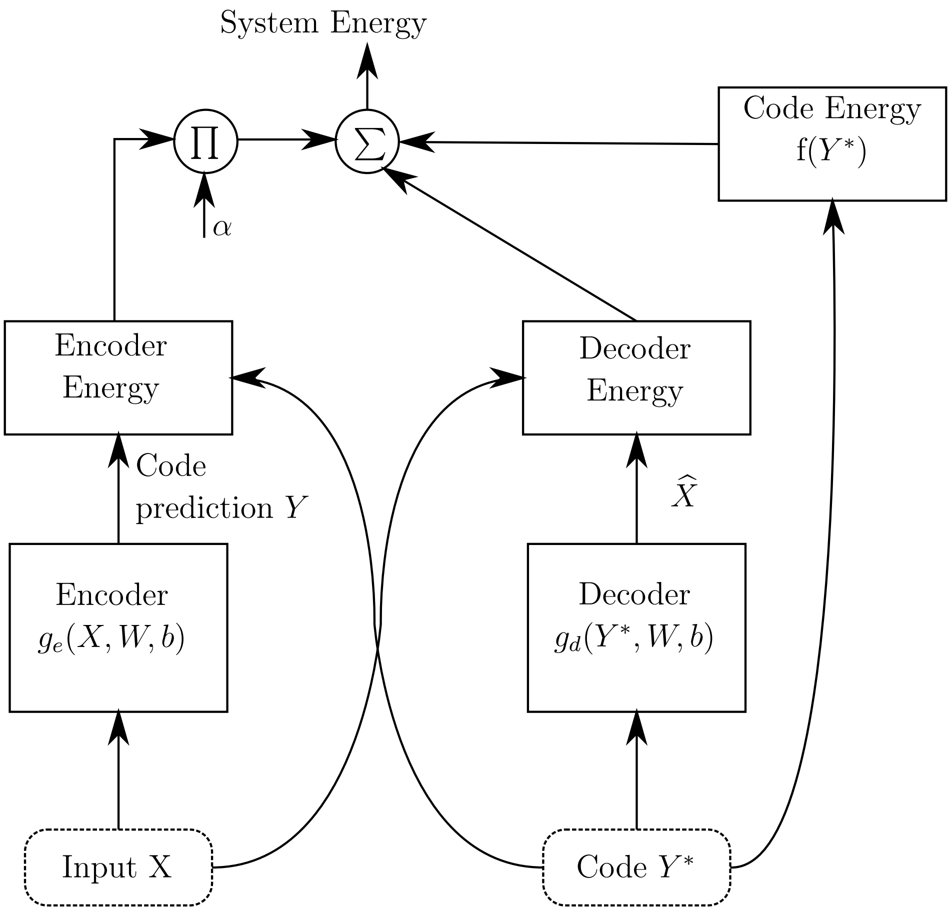


Figure 2.9: The energy calculation flow of the unsupervised training

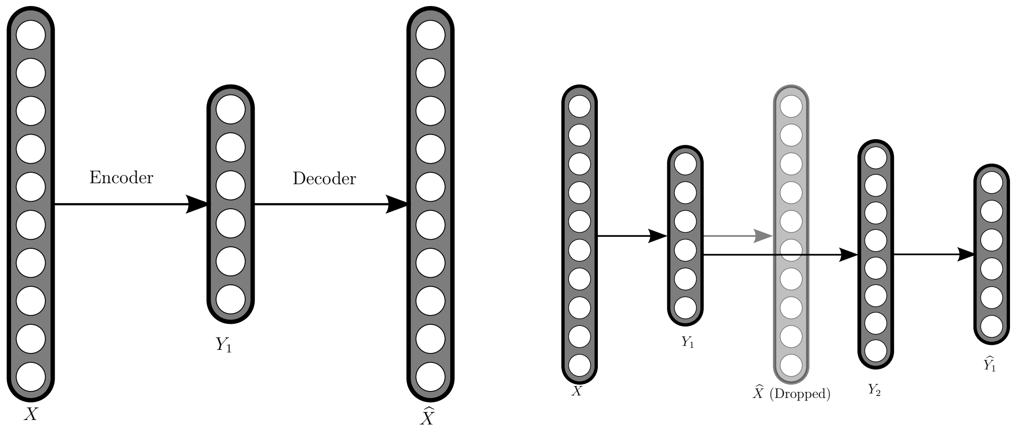
The partition function drops the probability of the unseen data. Energy of a probabilistic model can therefore be written as in Equation 2.14.

$$\mathrm{E}(X, Y) = -\log \mathrm{P}(X, Y) \tag{2.14}$$

The probability of the dataset $\mathcal{D}$ is $\prod\limits_{x_i \in \mathcal{D}} \mathrm{P}(x_i, Y)$ which is shown in Equation 2.15 in terms of energy.

$$\mathrm{E}(X, Y) = -\log \sum_{x_i \in \mathcal{D}} \mathrm{E}(x_i, Y) + \frac{1}{\beta} \log \int_x \mathrm{e}^{-\beta \mathrm{E}(x,Y)} \mathrm{d}x \tag{2.15}$$

The energy of the training set is subject to a minimization as a learning process. From the second term, it can be seen that energy of all data should be maximized, while the training samples energy is minimized. As a result, the samples from



(a) (b)

Figure 2.10: Stacking layers. (a) demonstrate a single layer while (b) shows the stacking of layers

the training set have low energies while other inputs give high energies.

On the other hand, computational graph based methods compute a parametric function to find a code from the input as a direct mapping. Autoencoder variants are the examples of this group and described in Section 2.3.1.

The probability distribution of the observed data, p(x), or the energy distribution of the observed data, E(x), is not directly related to posterior probability of the class given the input, p($c|x$) which is the main aim of the supervised learning. However the underlying factors discovered in unsupervised learning, which is the description of x, are also important while discriminating different classes.

The effect of unsupervised pretraining is examined in the literature as a regularization [23, 24] and as a method of better optimization [10]. The pretraining is equivalent to infinite penalty for solutions outside of a region of parameter space as there is a local minima, which parameters will be plunged into, in the vicinity of the pretrained parameters. Literature shows that for a small labeled dataset, unsupervised pretraining yields better generalization and locate a distinct optima which ensures a lower the variance, however if the capacity is lowered, the pretraining hurts the performance, since some of the underlying factors that describe the distribution are not useful in classification and these neurons are wasted.

In recent researches [16,17,32,51], the error rate of classification is decreased in well-known datasets when the learning is realized purely supervised with a large number of labeled data. These works has shown that the proper initialization and choice of the non-linearity play an important role in supervised training of deep neural networks, in order not to require any kind of layerwise pretraining. The reason behind the delay of this discovery is the advancement in the technology which is as important as the algorithmic improvement. With modern computers, the training is as fast as 60000 times as it was in 1990s. Moreover, the utilization of the graphical processor unit (GPU) even increases the speed 50-100 times more. Other factors leading to these achievements are dropout technique, convolutional architecture and rectified nonlinearities.

2.3.1 Autoencoder Types

An autoencoder encodes input X to Y, then reconstruct X as $\hat{X}$ from the code, Y, and minimizes the distance between X and $\hat{X}$. The cross entropy is the distance measure used for binary input and the l_2 norm is used for real valued input. If the only constraint was the distance, the encoder will learn the identity function so there are varieties of additional constraints that makes difference between autoencoder types. The general framework for an autoencoder architecture is given in Figure 2.10. $g_e(\bullet)$ and $g_d(\bullet)$, in the equations of this section, are encoding and decoding functions which are generally sigmoid functions.

Autoencoder The simplest version of the autoencoder does not include any constraints but the dimension of the code is less than the dimension of the input which creates a bottleneck, that prevents learning the identity function. The encoding and decoding functions are given in Equations 2.16a and 2.16b, respectively, where $s(\bullet)$ is the nonlinearity used, generally a sigmoid function. The weight W and W' is said to be *tied* weight if $W' = W^T$ where W^T denotes transpose of W.

$$Z = g_e(X, W, b) \qquad = s(W\, X + b) \tag{2.16a}$$

$$\hat{X} = g_d(Z, W', b) \qquad = s(W'\, X + b) \tag{2.16b}$$

Sparse Autoencoder Sparse autoencoder imposes a sparsity constraint on the code, that is, the dimension of the code can be greater than the dimension of the input, which enables learning an overcomplete basis. The energy is defined as follows:

$$E(X, Y^*) = \frac{1}{2}||X - W'\, Y^*||_2^2 + \lambda||Y^*||_1 \tag{2.17}$$

The inference of the code is achived by convex nonquadratic energy minimization. This slows down the operation, however, a variation, *predictive sparse decomposition* (PSD [47]) simultaneously finds the code by inference and also learn a mapping of the input to the optimal code Y^*. The energy is defined as follows:

$$E(X, Y^*) = ||X - W'\, Y^*||_2^2 + \alpha||Y^* - g_e(X, W, D) + \lambda||Y^*||_1 \tag{2.18}$$

In the regular usage after training, the inference is not used and only the learned mapping is used to find the code. Since inference is nonlinear, decoding can be linear.

Denoising Autoencoder Denoising Autoencoder is an exception that does not fit in the framework. Denoising autoencoder first adds noise to the input and generates $\tilde{X}$ in the training phase, then it reconstructs $\hat{X}$ from the noisy input $\tilde{X}$ and uses the original input, while measuring the reconstruction error that enforces the system to find representations that are independent from the noise. [93]

Contractive Autoencoder A generalization of Denoising Autoencoder is *Contractive Autoencoder* in which instead of adding noise to make the code noise immune, the jacobian of the code is used as a code energy [81]. The Jacobian of the code measures the variation of the code corresponding to infinitesimal change in the input. To minimize the total energy the Jacobian should be minimized as well, that makes the code invariant to changes in the input. However the variations that are important to reconstruct the input are considered in the code to make the reconstruction energy low. The energy is defined as in Equations 2.19 where $\mathcal{J}$ is the Jacobian.

$$E(X, Y^*) = ||X - g_e(Y^*, W', b)||_2^2 + \sum ||\mathcal{J}||_2^2 \tag{2.19}$$

CHAPTER 3

CONVOLUTIONAL NEURAL NETWORK FOR DEEP LEARNING

The training set should be at least as large as the tunable model parameters for a good learning. The prior information about the task is used to decrease the number of model parameters without loss of the ability to generalize, in other words, without the loss of network capacity.

A very strong assumption for image data is that the correlation of the raw input data at point $\vec{x}$ and at point $\vec{y}$ will decrease with the distance between point $\vec{x}$ and point $\vec{y}$. That suggests the use of locally connected (each neuron is connected to a set of neurons from a layer below that are in a neighborhood) neurons instead of using fully connected(each neuron has a connection with every neuron from a layer below) neurons as in classical neural networks. Another important prior about the image data is that features searched in every part of the image should be the same. For example, the well known hand-crafted features, such as Scale Invariant Feature Transform (SIFT) [66] and many others, describe a point in the same way for every part of the image. This clue can be used in conjunction with the local connectivity. The weights can be shared across each locally connected neurons. Exploitting these priors, CNNs have less parameters to train, amount of data required for training is less and training is easier while the performance is slightly worse compared to well trained fully connected alternatives which are nearly impossible to train with randomly initialized weights [7]. These features makes CNNs an attractive and good choice for deep learning.

Convolutional Neural Network(CNN) has less parameters to train, so less data is needed to train, and training easier with respect to fully connected counterparts.

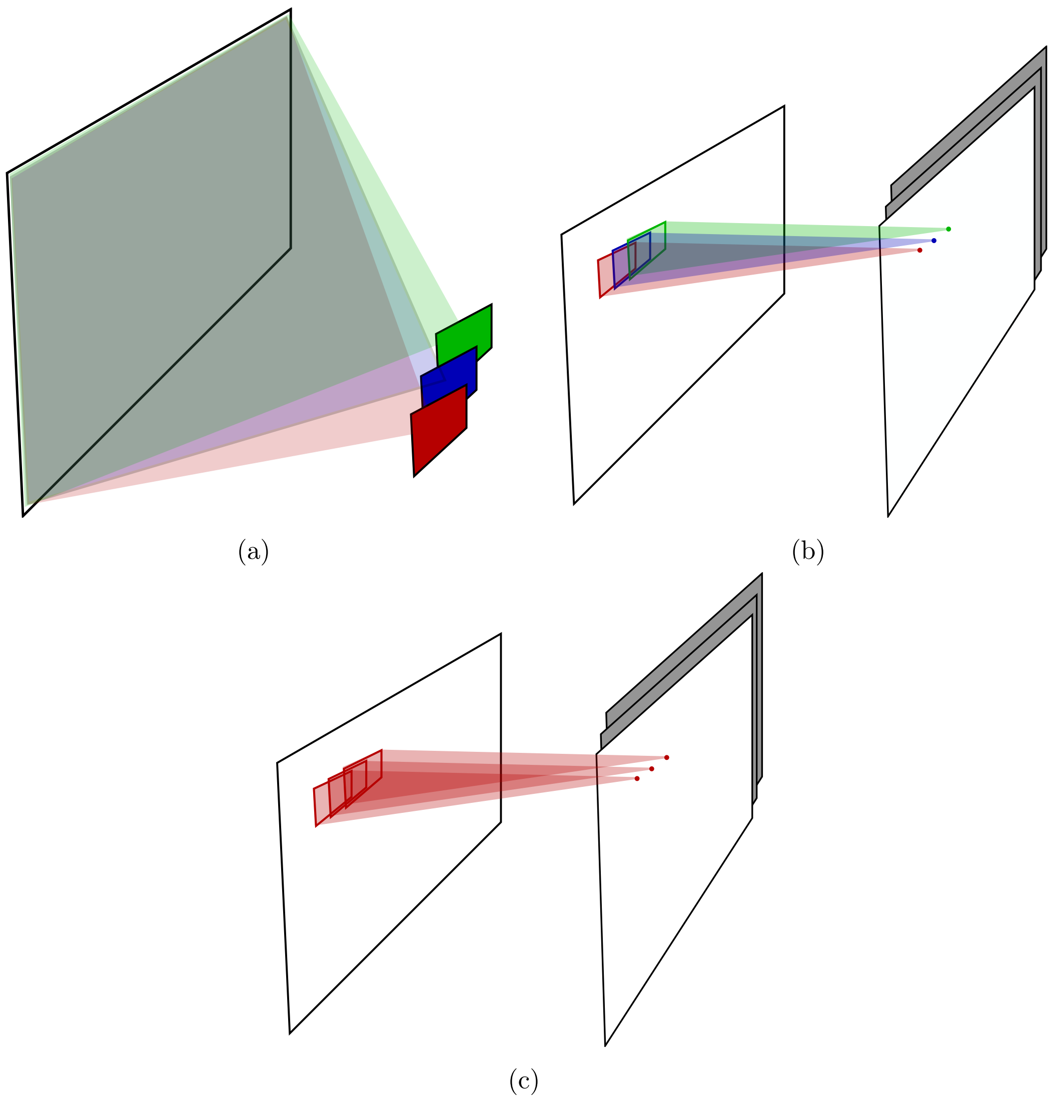


Figure 3.1: Illustration of (a) Fully Connected Neuron, (b) Locally Connected Neuron and (c) Locally Connected & Weight Sharing Neuron

The power of the CNN is supported with the fact that it has been used in many areas from 90's to today. An early example is the AT&T bank check reading system in 90's. Microsoft developed OCR and handwriting recognition systems using CNN, CNNs also used in object detection. Google developed a system using CNNs to detect license plate and face to protect privacy in StreetView. CNN also used for a project for long-range obstacle detection. [59]

Convolutional Neural Network architecture [56, 57] is a model that uses *weight sharing* and *local connectivity*. CNN is an exceptional model which can be

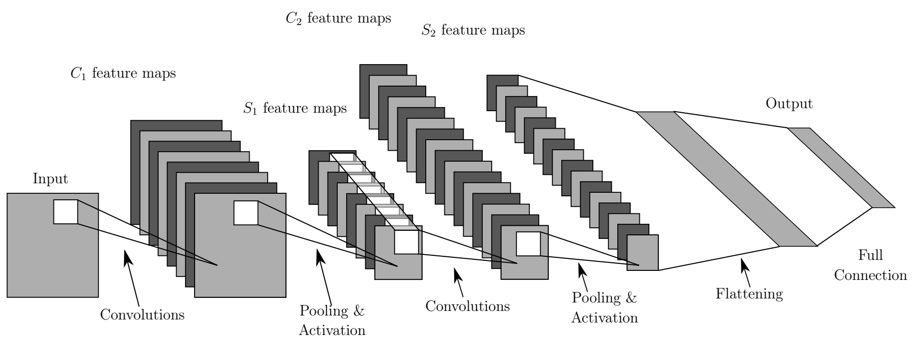


Figure 3.2: An overview of Convolutional Neural Network

trained from random initialized weights that is impossible for the same depth fully connected networks [7]. It is explained by an unproven hypothesis [7] that small number of inputs to a neuron (local connectivity of neurons or small receptive fields of neurons or small fan-in) will prevent *fundamental problem of deep learning*, which is the term used for exploding or diminishing gradients that is observed in deep learning. Therefore, gradient can propagate back more layers. Similar to other deep architectures, Convolutional Neural Networks have similar layers on top of each other, with each layer consists of a series of sublayers, namely a *convolution (filter bank)* sublayer, a *non-linearity* sublayer, a *local response normalization* sublayer, and a *feature pooling* sublayer that is shown in the Figure 3.2. A general framework consists of some layers of this type followed by a fully connected neural network, with a softmax layer at the output stage. The network capacity can be tuned by selection of the depth (number of layers) or breadth (number of and size of filters used).

3.1 A Layer of CNN

An overview of a layer is shown in Figure 3.3. Convolution sublayer can be thought as neurons arranged in 2D grid opposed to neurons in ordinary neural networks. Convolution operation applies the same filter to various locations. It is rational to apply the same filter everywhere in the images, especially for satellite images, since the data statistics are stationary across different locations. Locally connected, weight sharing neurons result in less number of parameters, so overfitting is not a problem, as much as it is for the fully connected networks. Such an approach makes using dropout unnecessary for this kind of layers [38].

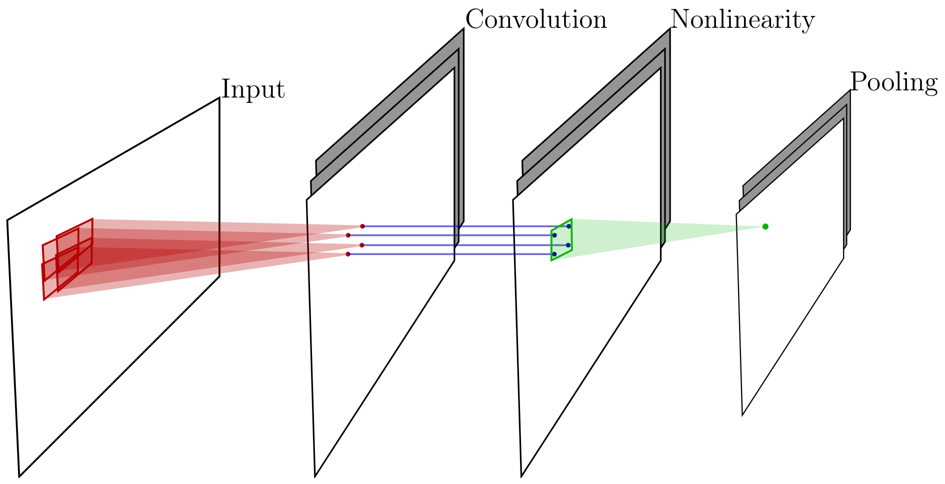


Figure 3.3: Overview of a single layer

3.1.1 Convolution Sublayer

This layer does a linear operation on input, convolves input with filters and adds bias which is different for each filter. The input image (h) of dimensions $n_{iy} \times n_{ix} \times n_c$, in which n_{iy} is the number of rows of the image, n_{ix} is the number of columns of the image, and n_c is the number of channels of the image, is convolved with rows and columns flipped filters (W) $\widetilde{W}$ with dimensions $n_{fy} \times n_{fx} \times n_c \times n_f$, in which n_{fy} is the number of rows of the filter, n_{fx} is the number of columns of the filter, n_c is the number of channels of the filter which is the same as the number of channels of the image, and n_f is the number of the filter. After convolution, bias b of size n_f is added. Output of this operation is matrix $\mathbf{a}$ which is in dimensions of $n_{ay} \times n_{ax} \times n_f$ where n_{ay} is the number of columns in resulting image, which is equal to $n_{iy} - n_{fy} + 1$, n_{ax} is the number of rows in the resulting image which is equals to $n_{ix} - n_{fx} + 1$, and n_f is the number of channels of the resulting image which is equal to number of filters. Flipped filters are used for convolution, since the convolution operation implicitly flipping filters by definition so giving flipped filters flipped back to original which makes understanding of the results easier. The relationship between convolution and correlation is given in Equation 3.1, where $*$, $\star$ represents convolution and correlation, respectively. W and b parameters are trainable parameters. The Equations 3.2 and 3.3 show the relationship of the output, input, weight and bias. Each filter will identify, hopefully different and important, features of the image.

$$A * \tilde{B} = A \star B \tag{3.1}$$

$$\mathbf{a} = h \star W + b \tag{3.2}$$

$$\mathbf{a}(x, y, z) = \sum_{p,r,s} h(x + p, y + r, s) W(p, r, s, z) + b_z \tag{3.3}$$

The gradient of the loss (L), required for back-propagation is given in Equations 3.4. The gradient equations in 2.12 are replaced by these gradients. $\tilde{\star}$ is full correlation that creates a result of size $(x + z - 1, n + t - 1)$ for input size (x, y) and (z, t) while valid correlation creates result of size $(x - z + 1, n - t + 1)$.

$$\nabla_{\mathbf{W}^{(k)}} L = \nabla_{\vec{p}^{(k)}} L \star \mathbf{h}^{k-1} \tag{3.4a}$$

$$\nabla_{\mathbf{b}^{(k)}} L = \sum \nabla_{\vec{p}^{(k)}} L \tag{3.4b}$$

$$\nabla_{\mathbf{h}^{(k-1)}} L = \sum_{j} \nabla_{\vec{p}^{(k)}} L \,\tilde{\star}\, \mathbf{W}^{(k)} \tag{3.4c}$$

The weights should be initialized randomly as in ordinary fully connected networks that is described in 3.2.2.

3.1.2 Non-linearity Sublayer

Linear operations (say **op**) are associative; hence adding more linear layers having weights (W_1, W_2, W_3, ..., W_n) will be the same as using one layer that does the same op with weight W equals to $W_n\mathbf{op}...\mathbf{op}W_3\mathbf{op}W_2\mathbf{op}W_1$. This fact demonstrates the importance of the non-linearity in the hierarchical network. A variety of non-linear functions can be used as the nonlinearity and two well-known examples are sigmoid and rectified linear function. Those widely used functions are explained in the following parts and the derivatives, which are required for backpropagation, are given. The nonlinearity is applied to the output of the convolution sublayer.

3.1.2.1 Sigmoid Unit

Logistic sigmoid function, Equation 3.5a, with derivative given in equation 3.5b and shown in Figure 3.4 and hyperbolic tangent sigmoid function, Equation 3.6a, with derivative given in Equation 3.6b and shown in Figure 3.4 are widely used non-linearities. Sigmoid functions are monotonically increasing functions with smooth derivatives and asymptotes at finite value. Networks with logistic sigmoid function are harder to train, since the outputs are unnormalized values which will be an input to other layer and the normalization of the input makes convergence faster [58]. Hyperbolic tangent with parameters $\alpha = 1.5179$ and $\beta = 2/3$ is used for all experiments that uses hyperbolic tangent non-linearity as these values are suggested in [54].

$$\mathrm{sigm}(a) = \frac{1}{1+\exp(-a)} \tag{3.5a}$$

$$\mathrm{sigm}'(a) = \mathrm{sigm}(a)(1-\mathrm{sigm}(a)) \tag{3.5b}$$

$$\alpha \tanh(\beta\, a) = \alpha \frac{\exp(\beta\, a) - \exp(-\beta\, a)}{\exp(\beta\, a) + \exp(-\beta\, a)} \tag{3.6a}$$

$$\alpha \tanh'(\beta\, a) = \alpha\, \beta\, (1 - \alpha \tanh^2(\beta\, a)) \tag{3.6b}$$

3.1.2.2 Rectified Linear Unit (ReLU)

Rectified linear function, with Equation 3.7a, derivative given in equation 3.7b and shown in Figure 3.4, is shown to improve the discriminative power of the network [44] and also biologically plausible [75].

$$\mathrm{relu}(a) = \max(0, a) \tag{3.7a}$$

$$\mathrm{relu}'(a) = a > 0 \tag{3.7b}$$

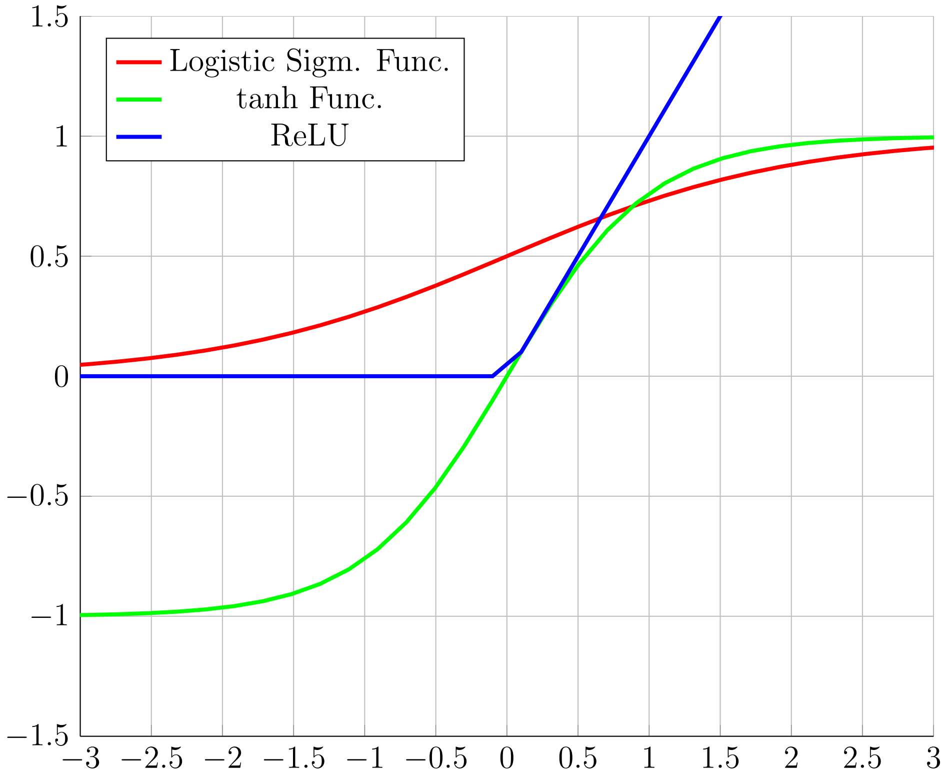


Figure 3.4: Non-linearty functions: (a) Logistic Sigmoid Function (b) Hyperbolic Tangent Sigmoid Function (c) Rectified Linear Function

3.1.3 Local Response Normalization Sublayer

Local response normalization sublayer is advised after rectified linear activation function for its ability to increase generalization by generating competition between neighboring filters. The activation of the neighboring neurons at the same spatial location, in other words, neurons at the same location belonging to different convolution kernels, are normalized according to equation 3.8b. That normalization is analogous to lateral inhibition found on the biological neural network. Gradient is also presented in equation 3.8c. [51]

$$\gamma = 1 + \alpha \sum_{j=i-\frac{N}{2}}^{j=i+\frac{N}{2}} (p(x,y,j))^2 \tag{3.8a}$$

$$u(x,y,i) = (\gamma)^{\beta} \tag{3.8b}$$

$$\nabla_u L = \frac{\nabla_u L}{\gamma^{\beta}} - \frac{2\,\alpha\,\beta\,p(x,y,i) \sum_{j=i-\frac{N}{2}}^{j=i+\frac{N}{2}} (p(x,y,j))}{\gamma^{\beta+1}} \tag{3.8c}$$

3.1.4 Pooling Sublayer

Pooling, groups spatially close activations together, similar to *bag of words*. Pooling operates a function, maximum (max-pooling) or average (avg-pooling) are two well known examples [12], on a region which can be overlapping. Overlapping pooling windows are empirically shown not to provide significant improvements over non-overlapping ones [86]. Function applied on a neighborhood ($\mathcal{N}$) of size n is shown in Equations 3.9, 3.10 for average and maximum pooling, respectively:

$$h(x,y,z) = \frac{1}{n} \sum_{i_x, i_y \in \mathcal{N}} u(i_x, i_y, z) \tag{3.9}$$

$$h(x,y,z) = \max(u(i_x, i_y, z)) : i_x, i_y \in \mathcal{N} \tag{3.10}$$

Pooling sublayer has two important functions. Firstly, pooling makes the system invariant to small transformations, since the pooling operates on a region. Secondly, pooling works as a dimension reduction technique. The aim of pooling is to store important features while discarding others.

Pooling is biologically inspired from complex cell [40–42]. Simple cells extracts features like oriented edges and complex cells combine simple cell activations in a spatial neighborhood [86]. The convolution and nonlinearity sublayers together imitates simple cells, while the max pooling sublayer mimics the complex cell.

Similar techniques are also widely used in hand crafted feature extraction algorithms. For example, Scale Invariant Feature Transform(SIFT [66]), Histogram of Oriented Gradient(HoG [21]) and derivatives use gradient pooling, in which gradient orientations are pooled in a neighborhood with max pooling, getting the maximum intensity direction.

3.2 Improving Backpropagation: Rule of Thumbs

3.2.1 Preprocessing

For some nonlinearities and deep learning architectures, preprocessing is not necessary; however, it improves the speed of the convergence and converge to a better optima, better in the sense that it can generalizes to unseen data better. Inputs to a network should have zero mean and also outputs should also have zero mean, since the output of a layer will be an input for the next layer. The bias on the input enforces weights to be updated together as detailed in [58]. Common motion of the weights will make gradient descent to zigzag, so slow down the convergence. Another factor slowing down the convergence is the variance of the pixel, that is the change of the intensity of that pixel across the images, and correlation between the pixels in the same image. Different variances along the axes generates plateaus that have small derivative. Removing the linear correlation between pixels are relatively easy using Principle Component Analysis(PCA [45]).

PCA PCA finds orthogonal bases, that are the eigenvectors of the covariance matrix of the data. These vectors are in the direction of the scatter of the data. Transformation applied to the data to project it on to the eigenvectors are shown with $\mathcal{T}$. Covariance matrix is diagonal after transformation with diagonal entries equal to the eigenvalues. Transformed data is normalized with standard deviation(σ) of each basis. The resulting data matrix is denoted as $\mathcal{D}_{pca}$ in Algorithm 3.

Zero Phase Component Analysis (ZCA) ZCA is the same as PCA, however at the end of finding uncorrelated data, it is transformed back using inverse transform $\mathcal{T}^{-1}$ which is equals to projecting back to original bases. This transformation results in a new uncorrelated and normalized data representation $\mathcal{D}_{zca}$ using the original bases or in other words in the original domain as correlated version. Visualization is more comprehensible and spatial distance will be meaningful with such a representation unlike $\mathcal{D}_{pca}$. This property is important for convolutional neural network, which has an assumption that correlation of the pixel intensity with others are inversely proportional to the spatial distance.

This procedure is depicted in Figure 3.5 and the algorithm is given in Algorithm 3, where $\mathcal{D}$ is the original data and x^T denotes transpose.

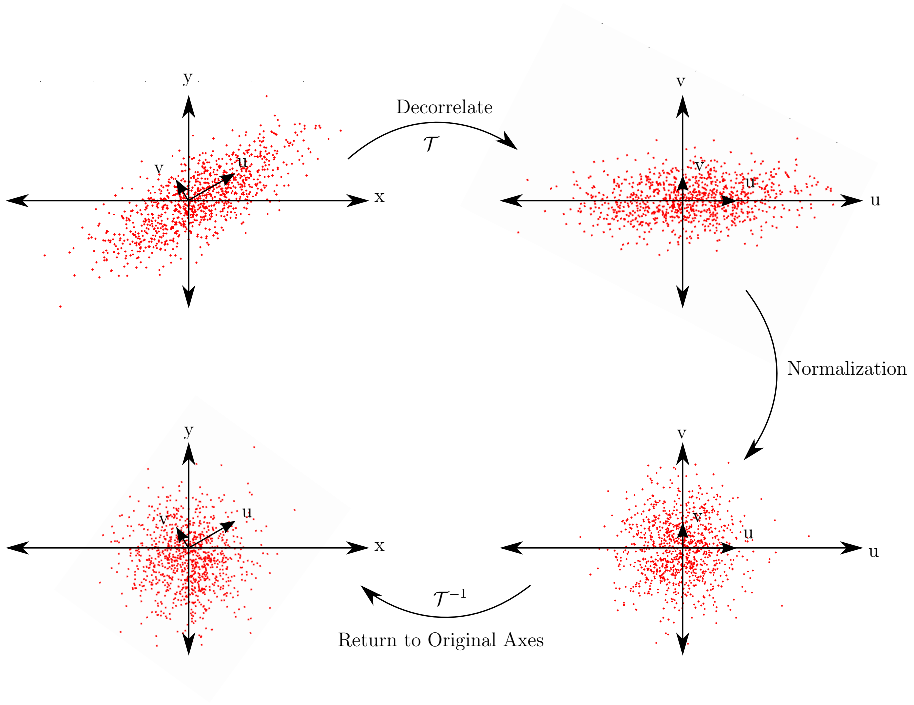


Figure 3.5: Overview of a single layer

Algorithm 3 ZCA

```
$\Sigma = \mathcal{D} * \mathcal{D}^T$
Calculate Eigenvectors($u$) and Eigenvalues($\lambda$) Of $\Sigma$
$\mathcal{D}_{uncorrelated} = \mathcal{T}(\mathcal{D}) = u^T \cdot \mathcal{D}$
$\mathcal{D}_{pca} = (I \cdot \frac{1}{\sqrt{\lambda}}) \cdot \mathcal{D}_{uncorrelated}$
$\mathcal{D}_{zca} = \mathcal{T}^{-1}(\mathcal{D}_{pca}) = u \cdot \mathcal{D}_{pca}^T$
```

3.2.2 Weight (Filter) Initialization

Weights of a neural network should be initialized randomly. If weights are initialized from the same value, each neuron will be updated by the same values. Such a situation leads to an overall system that learned only one feature. Weights random initialization is important to break the symmetry, while biases can be initialized from zero.

If a sigmoid function is used as a non-linearity, selection of weights gets more crucial, since these types of neurons might saturate. Weights in this circumstance initialized so that activation of neuron will be in the linear region. That will result in first learning the linear properties then non-linear ones [58].

The weights are randomly sampled from uniform distribution, $u(-r, r)$ values of r depends on fan-in and fan-out of the network as suggested in [31, 58] and the value of r in equation 3.11a is used for logistic function, r in equation 3.11b is used for hyperbolic tangent, is taken from [8].

$$r = 4\sqrt{\frac{6}{\text{fan-in} + \text{fan-in}}} \tag{3.11a}$$

$$r = \sqrt{\frac{6}{\text{fan-in} + \text{fan-in}}} \tag{3.11b}$$

For ReLU non-linearity, weights should be initialized with in a larger range so that the activations should not be in negative side of the function which makes derivative zero so makes learning impossible. Alternatively, starting bias with positive value will shift the inputs of ReLU to positive side [38].

3.3 Experiment Setup

The parameter affects are analyzed with experiments on CIFAR10 dataset.

3.3.1 CIFAR-10 Dataset [50]

The CIFAR-10 dataset is composed of 60,000 32x32 colour images belonging to one of ten classes, with 6,000 images per class. The CIFAR-10 is a challenging dataset with the top performing method [64] can reach at most %89.6 classification rate.

The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains 1,000 randomly-selected images from each class. The training batches contains the remaining images in random order. The training set contains 5,000 images from each class. Figure 3.6 visualize samples from the dataset with labels.

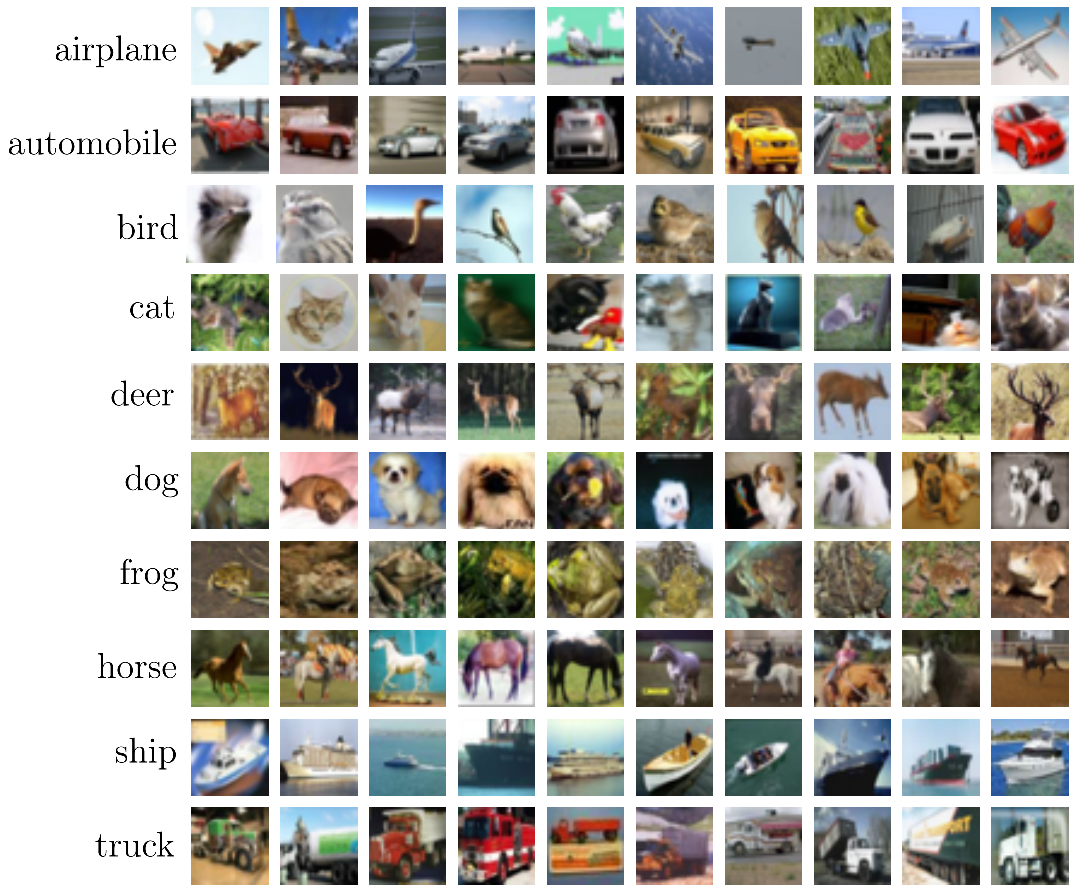


Figure 3.6: Sample images from all classes for CIFAR-10 [50]

3.4 Analysis of The Results

The training set and test set *misclassification rate* change is the key factor to analyze the network performance. The misclassification rate is given in Equation 3.12, where MC is the number of misclassified samples, i.e. the samples whose class labels assigned by the network are not the same as the associated label in the dataset, and T is the total number of samples and MR is the misclassification rate.

$$MR = \frac{MC}{T} \tag{3.12}$$

Tables 3.2 and 3.4 demonstrate the results for CNN and fully connected architectures. Names of the tested architectures are given as acronyms and the corresponding explanation of this notation is provided on Tables 3.1 and 3.3.

In all of the experiments, the architectures are followed by a final softmax layer to assign class probabilities. The pooling layer is followed the 2^{nd} and 3^{nd} layers, if it exists in the architecture. Mini-Batch Stochastic Gradient with momentum is used as the training algorithm. The gradients are calculated over the training set and the best parameters, according to validation set performance, is used for the test performance.

Important and simple observation from the tables is the fact that the validation set and test set performances are quite close to each other which makes using validation set to select best parameters meaningful.

The results show that ReLU activation usually performs better than the others, and finds an optima that can memorize the training set for the same sized network which suggest that the optimization with ReLU is easier so that it can optimize till the network finds an optima.

Preprocessing the inputs has great impact on the performance. The preprocessing decreases the misclassification rate by 20% on the average which is consistent with the previous works in the literature [57].

Table3.1: Acronyms corresponding to fully connected architectures

Acronym	Preprocessing	Number Of Fully Conn. Layer	Activation	Dropout
N|0F|-|N	No	0	softmax	No
N|1F|L|N	No	1	logistic	No
N|1F|T|N	No	1	tanh	No
N|1F|R|N	No	1	ReLU	No
Y|1F|L|N	Yes	1	logistic	No
Y|1F|T|N	Yes	1	tanh	No
Y|1F|R|N	Yes	1	ReLU	No
N|2F|L|N	No	2	logistic	No
N|2F|T|N	No	2	tanh	No
N|2F|R|N	No	2	ReLU	No
Y|2F|L|N	Yes	2	logistic	No
Y|2F|T|N	Yes	2	tanh	No
Y|2F|R|N	Yes	2	ReLU	No
N|3F|L|N	No	3	logistic	No
N|3F|T|N	No	3	tanh	No
N|3F|R|N	No	3	ReLU	No
Y|3F|L|N	Yes	3	logistic	No
Y|3F|T|N	Yes	3	tanh	No
Y|3F|R|N	Yes	3	ReLU	No
Y|3F|L|Y	Yes	3	logistic	Yes
Y|3F|T|Y	Yes	3	tanh	Yes
Y|3F|R|Y	Yes	3	ReLU	Yes

The effect of the dropout can be observed by the difference of the test misclassification rate of Y|3F|R|Y and Y|3F|R|N, even though the training set misclassification of is higher for Y|3F|R|Y. This shows the improvement brought by the utilization of dropout.

The convolutional networks outperform the fully connected networks having the same depth as shown in Table 3.4. The best overall performance is obtained by using the architecture 3C|2F|32|F|555. Different filter sizes are tested: convolutional layers with increasing filter size, convolutional layers with decreasing filter size and convolutional layers with the same filter size. The same sized filters are

Table3.2: Misclassification rates for experiments with fully connected architectures

Architecture	Test Misclassification Rate	Training Misclassification Rate	Validation Misclassification Rate
N|0F|-|N	0.76	0.76	0.77
N|1F|L|N	0.90	0.90	0.90
N|1F|T|N	0.90	0.90	0.90
N|1F|R|N	0.87	0.85	0.85
Y|1F|L|N	0.57	0.17	0.57
Y|1F|T|N	0.64	0.48	0.64
Y|1F|R|N	0.53	0.00	0.52
N|2F|L|N	0.84	0.84	0.84
N|2F|T|N	0.90	0.90	0.90
N|2F|R|N	0.80	0.81	0.82
Y|2F|L|N	0.61	0.43	0.61
Y|2F|T|N	0.63	0.47	0.63
Y|2F|R|N	0.57	0.00	0.57
N|3F|L|N	0.80	0.80	0.80
N|3F|T|N	0.83	0.83	0.83
N|3F|R|N	0.87	0.87	0.87
Y|3F|L|N	0.61	0.41	0.61
Y|3F|T|N	0.63	0.45	0.63
Y|3F|R|N	0.56	0.00	0.55
Y|3F|L|Y	0.64	0.59	0.63
Y|3F|T|Y	0.61	0.46	0.61
Y|3F|R|Y	0.48	0.01	0.48

shown to perform better then others.

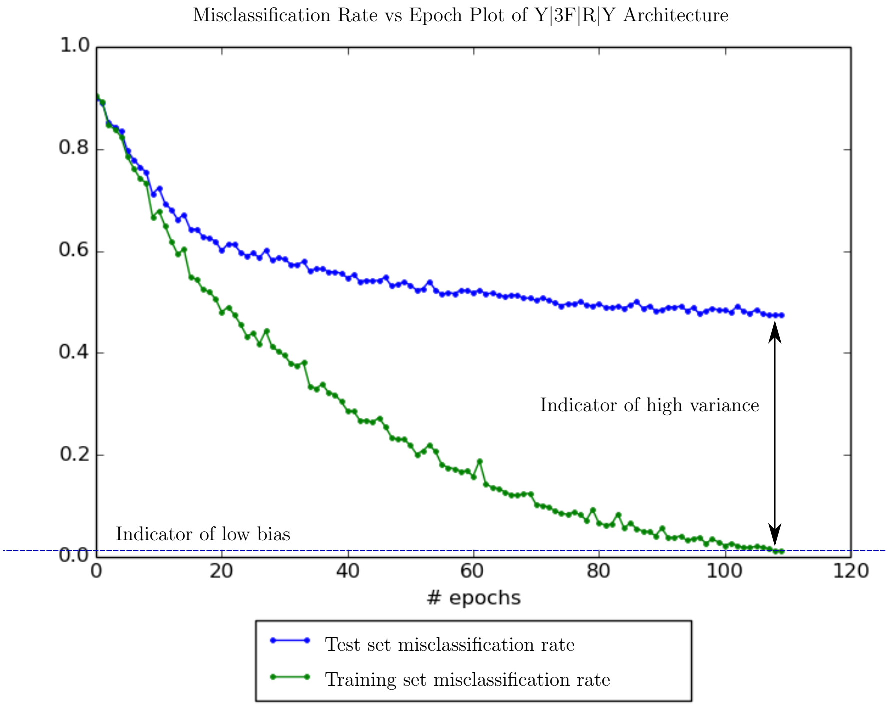
The bias of the network can be measured with the misclassification rate on the training set, while the variance can be observed with the difference of the misclassification rate on the training set and test set. The bias is measured with training set performance, due to its optimization criteria which is to minimize the average error, in other words, the function that is approximated, is the function mapping training samples to the labels. The variance measured by the performance on different datasets, training and test sets can be used for that purpose. Figure 3.7 shows the evolution of the performance with the training iteration and the bias-variance analysis is also given on the plot.

Table3.3: Acronyms corresponding to convolutional architectures

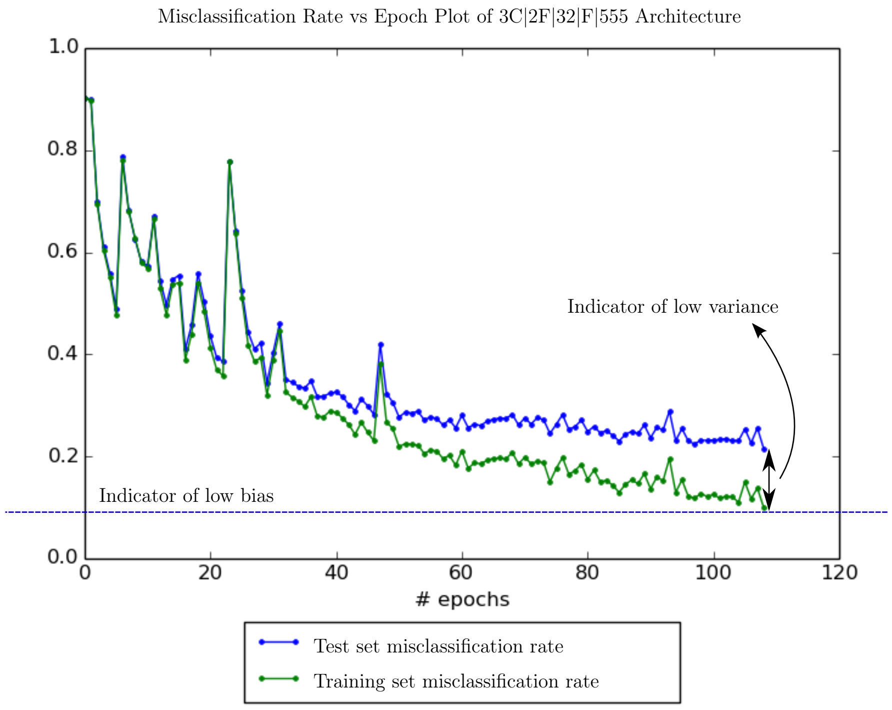
Acronym	Number Of Conv. Layer (CL)	Number Of Fully Conn. Layer (FCL)	Pooling	Dropout	Filter Shapes
1C|2F|N|N|555	1	2	No	No	5x5
2C|1F|N|N|555	2	1	No	No	5x5 5x5
3C|1F|N|N|555	3	1	No	No	5x5 5x5 5x5
3C|2F|N|N|555	3	2	No	No	5x5 5x5 5x5
3C|2F|22|N|555	3	2	2x2 max	No	5x5 5x5 5x5
3C|2F|22|A|555	3	2	2x2 max	Yes	5x5 5x5 5x5
3C|2F|44|N|555	3	2	4x4 max	No	5x5 5x5 5x5
3C|2F|44|A|555	3	2	4x4 max	Yes	5x5 5x5 5x5
3C|2F|33|N|555	3	2	3x3 max	No	5x5 5x5 5x5
3C|2F|32|A|555	3	2	3x3 max overlapping	Yes	5x5 5x5 5x5
3C|2F|32|F|555	**3**	**2**	**3x3 max overlapping**	**After FCL**	**5x5 5x5 5x5**
3C|2F|32|F|357	3	2	3x3 max overlapping	After FCL	3x3 5x5 7x7
3C|2F|32|F|753	3	2	3x3 max overlapping	After FCL	7x7 5x5 3x3

Table3.4: Misclassification rates for experiments with fully connected architectures

Architecture	Test Misclassification Rate	Training Misclassification Rate	Validation Misclassification Rate
1C|2F|N|N|555	0.34	0.11	0.34
2C|1F|N|N|555	0.39	0.00	0.38
3C|1F|N|N|555	0.65	0.61	0.66
3C|2F|N|N|555	0.43	0.30	0.44
3C|2F|22|N|555	0.33	0.11	0.32
3C|2F|22|A|555	0.29	0.27	0.29
3C|2F|44|N|555	0.28	0.10	0.27
3C|2F|44|A|555	0.25	0.21	0.24
3C|2F|33|N|555	0.28	0.10	0.27
3C|2F|32|A|555	**0.22**	**0.16**	**0.21**
3C|2F|32|F|555	0.20	0.11	0.20
3C|2F|32|F|357	0.25	0.18	0.25
3C|2F|32|F|753	0.21	0.10	0.21



(a)



(b)

Figure 3.7: Performance change with respect to the training epoch (a) for the Y|3F|R|Y and (b) 3C|2F|32|F|555 architectures

CHAPTER 4

APPLICATIONS IN REMOTE SENSING

Due to the availability of many commercial satellite systems such as SPOT(Satellite Pour l'Observation de la Terre), QuickBird, GeoEye, IKONOS etc., access to satellite images has become much easier and cheaper. The resultant excessive data needs a deep analysis and interpretation for a range of applications like map making, urban planning, resource management, climate change observations and naval warfare.

The satellite images can be acquired panchromatic or multispectral, according to the available radiometers on the satellite. Panchromatic images are composed of a single band, in which the pixel intensities are directly proportional to the the radiation captured by the sensors. On the other hand, multispectral images are composed of many channels each of which corresponds to a band of electromagnetic spectrum. In addition to red(R), green(G) and blue(B) channels, near-infrared (NIR), middle-infrared(MIR) and far-infrared(FIR) information may be acquired from some available satellite systems.

Target detection from multispectral satellite imagery is a challenging yet necessary task. In general, the word *target* may either be used for geospatial objects such as vehicles (ships or airplanes) and buildings, or it can resemble some region of interest, such as sea, shorelines, harbours, airports, forests, roads etc.

Main approaches to the object detection problem can be categorized as appearance-based methods, methods benefiting local descriptors and shape based methods. Appearance-based methods, such as template-matching [13] generally mean an exhausting search of the whole image for high responses to a transformed set of the template of the target [74] or to specially designed filters [14]. Local descriptor based methods benefit the geometric structure of the objects to extract high-level features to be matched between training examples and test images [62,63]. In addition, templates based on shape properties are also proposed for the object recognition problem [5, 30]. In the third category, segmentation is a first step for the extraction of target shape, which is then described by using semantic relations, texture and/or low level features such as edges and corners [3, 91]. Saliency based region of interest extraction can be utilized before applying any of these individual or composed methods. However, the difficulty of the object detection in satellite images problem lies behind the variability of size, pose and color of the object with additional weather and illumination effects. Moreover, the infrared channels, which cannot be captured by human eye, carry valuable information. The lack of knowledge of human on this additional information makes explicit description of the target even harder.

Therefore, a learning based classification method, which preferably generates the hierarchical representation of input data, may have many strengths in the object detection problem in multispectral satellite images either with unsupervised, semi-supervised or supervised training.

4.1 Airplane Detection

In this thesis, the airplane object is chosen as the target for testing the detection performance. In spite of the distinctive geometrical structure, airplanes may vary highly in size, orientation, pose and especially color.

The training and also test data are obtained using the ground truth of centroid points labeled by an expert. The data has a 2m of spatial resolution and four spectral bands: R, G, B and NIR. The samples are extracted from bigger satellite data by cropping windows of 32x32 pixels. An example set of images for *target* class (airplane) and *non-target* class can be seen in Figure 4.4. Total 9900 training data is collected, of which 900 of them contain target and 9000 of them is non-target samples. 21200 training images are generated by rotating and cropping 900 sample images that contain target. 140 target and 140 non-target patches are also cropped as the test set. Test set data are not cropped from the images used to extract training set.

ZCA is applied as the preprocessing step. The covariance and mean is calculated from the training set and the same transformation is applied to the test set.

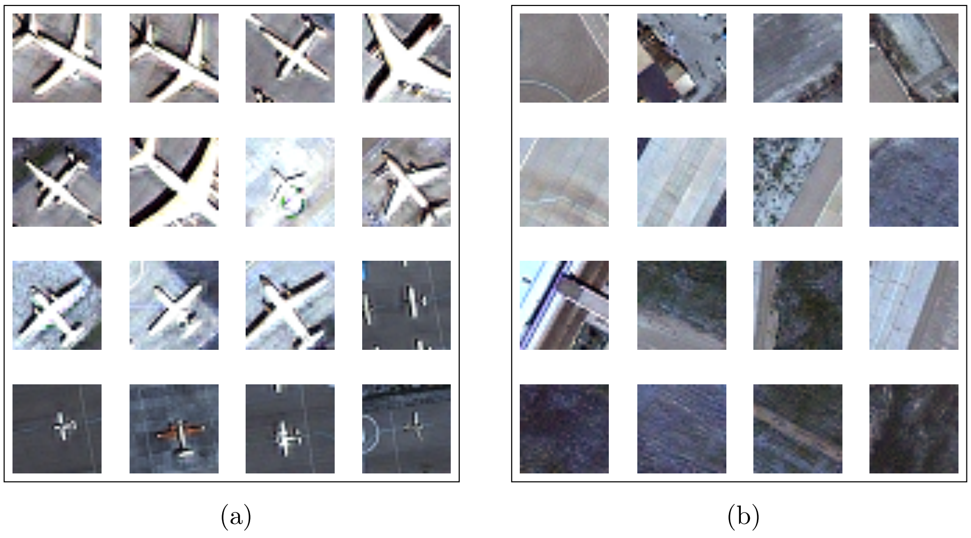


Figure 4.1: An example set of images from the dataset, (a) Airplane Labeled Images and (b) Non-Airplane Labeled Images

From the tests with different parameters, best performance of 7.03% misclassification rate is achieved with an architecture which has 3 convolutional layers, each of which is followed by 2x2 max-pooling and fed to 2 layer fully connected network with dropout layers which improves test set performance. All layers have ReLU activation since the best results are obtained by using ReLU, in the CIFAR10 tests. The confusion table for airplane detection is given in Table 4.1. The learning curve for the airplane detection algorithm is presented in Figure 4.2.

Learning procedure is an optimization problem which aims to minimize the training error, E_{train}, by selecting the best parameters for the model which is expected to generalize to unseen data as well, if the test set and training set characteristics are similar. However, Figure 4.2 shows that the learning behavior of training and test data used in the tests differs slightly. The reason behind

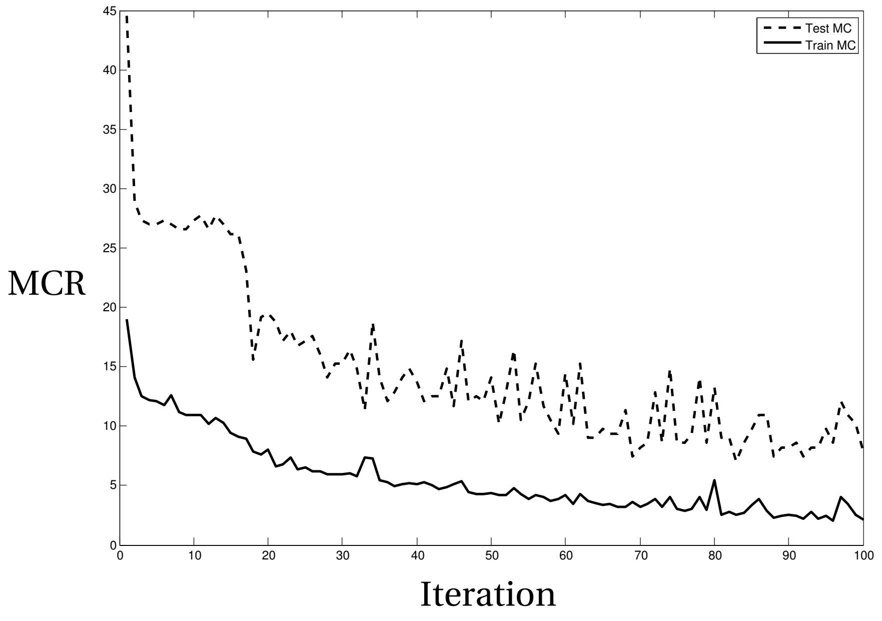


Figure 4.2: Learning curve for airplane detection

this difference is the limited number of the training samples, which leads to memorization instead of generalization.

Table4.1: Confusion table for airplane detection problem

GT \ Assigned Label	Airplane	Non-Airplane
Airplane	0.90	0.10
Non-Airplane	0.11	0.89

The target detection algorithm is also run on the whole patch of 1024x1024 image that contains an airport. Images obtained by sliding a window of size 32x32 pixels are fed to the learned network and the output labels are concatenated to form the detection result as an image. Examples of the detection result with the corresponding input images are depicted in Figure 4.3. Due to the fact that training images that contain small airplanes resembles small rectangular shapes, road lines and small buildings are confused with the airplane. Although there are false alarms, the missed targets are rare.

4.2 Classification of Airport Region Targets

Three targets are selected that are found on the airports, namely, airplane, building and dispersion areas. This 3-class classification problem is attacked using a CNN architecture. Two datasets are created by cropping the same satellite images used to form airplane detection dataset. The first set is optained by cropping 32x32 image patches centered at the object centroid.On the other hand, for the second *resized*, the object's bounding box is cropped and then resized to 32x32. Both training sets are composed of 890 airplanes, 3900 buildings and 100 dispersion areas. Test sets are composed of 144 airplanes, 675 buildings and 5 dispersion areas.

The training dataset augmentation is used to increase the number of training samples by rotation the input images, at the end of augmenting the training set the number of airplanes, buildings and dispersion areas became 21384, 31200 and 5256, respectively.

The confusion tables for both datasets, resized and not resized, are given in Tables 4.2, 4.3 and the learning curves that indicates the evolution of misclassification rate with training iterations are given in Figures 4.5 and 4.6, respectively.

Table4.2: Confusion table for classification of airport region targets of resized images

GT \ Assigned Label	Airplane	Building	Dispersion Area
Airplane	0.65	0.35	0.00
Building	0.05	0.94	0.01
Dispersion Area	0.80	0.20	0.00

Table4.3: Confusion table for classification of airport region targets

GT \ Assigned Label	Airplane	Building	Dispersion Area
Airplane	0.66	0.34	0.00
Building	0.02	0.97	0.00
Dispersion Area	0.80	0.20	0.00

Dispersion areas can not be learned as observed from the error rates in the confusion tables. The main reason behind that fact is the low number of training samples that dispersion area target have. The problem is tried to be overcame by augmenting the training set with the rotated training samples but it is not adequate for dispersion area target.

The building target has the largest number of samples so it generalizes better.

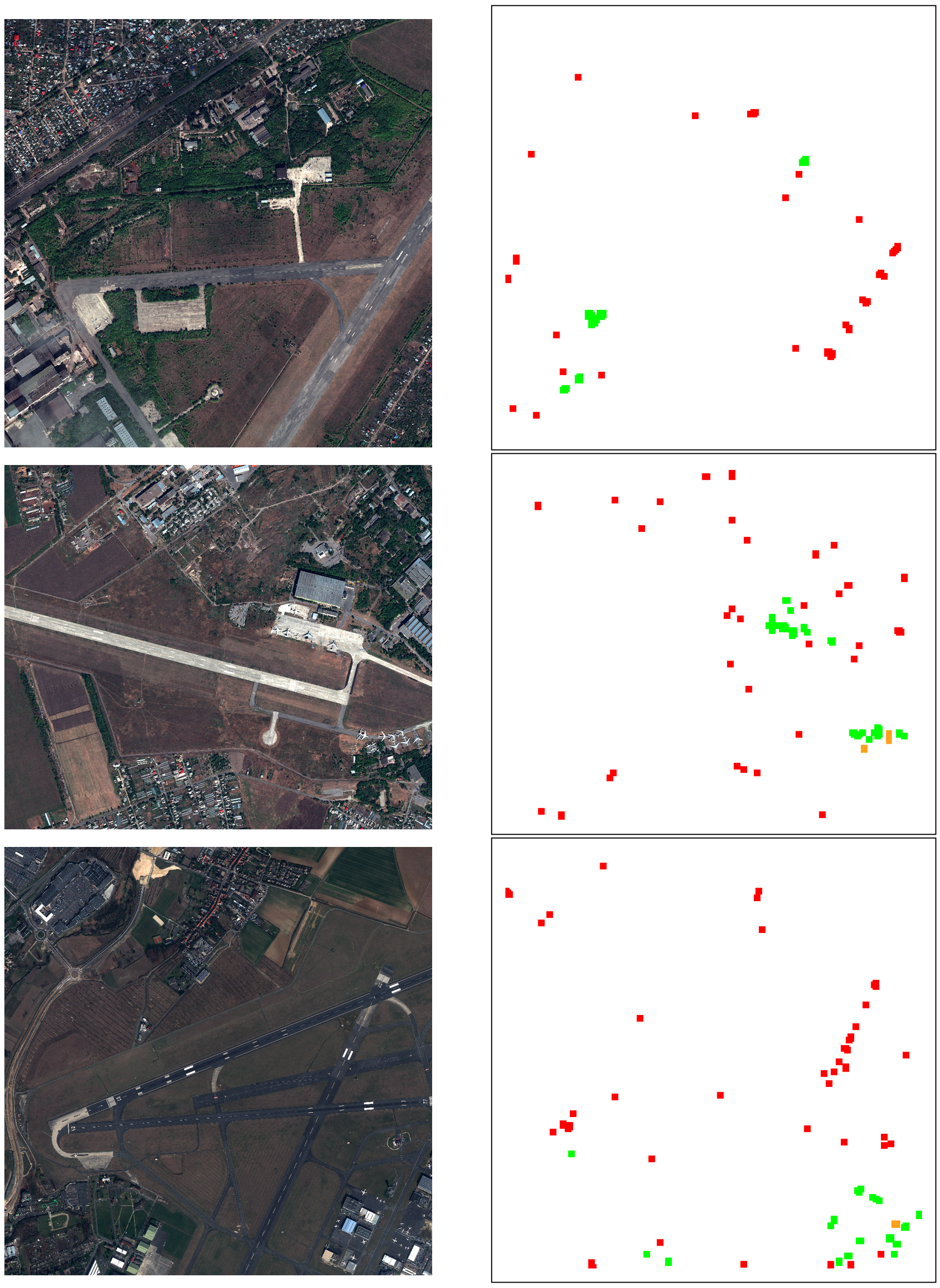
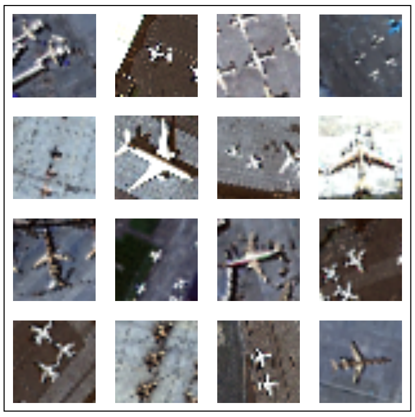
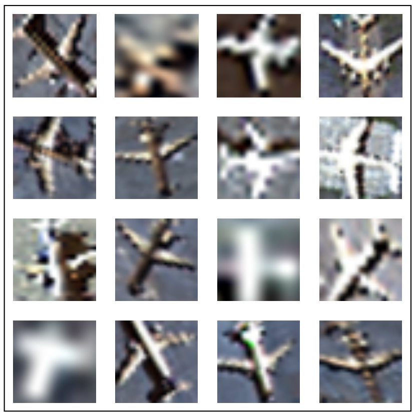
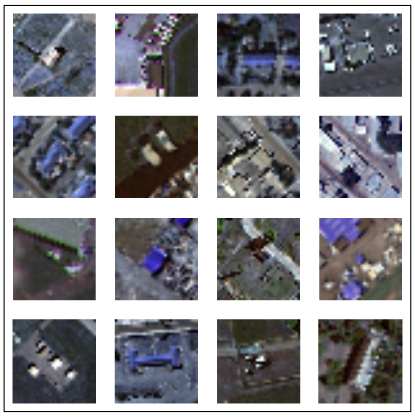
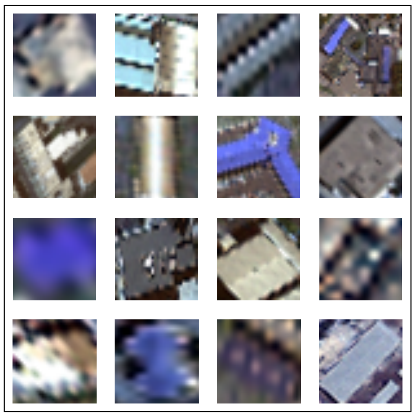

(a) (b)

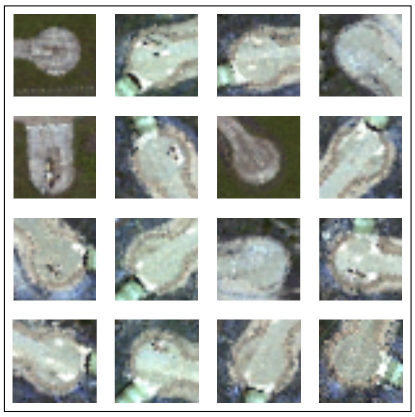
Figure 4.3: The target detection algorithm run on (a) the 1024x1024 image that contains an airport. (b) the detection result of the corresponding image where red represents false detection, green represents true detection and orange represents missed targets.


(a) Airplane Class


(b) Airplane Class Resized


(c) Building Class


(d) Building Class Resized


(e) Dispersion Area Class

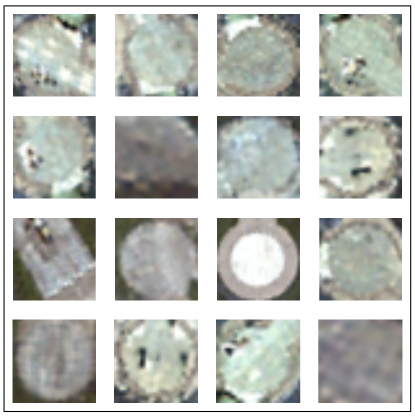

(f) Dispersion Area Class Resized

Figure 4.4: An example set of images from the classification dataset with and without resizing during cropping

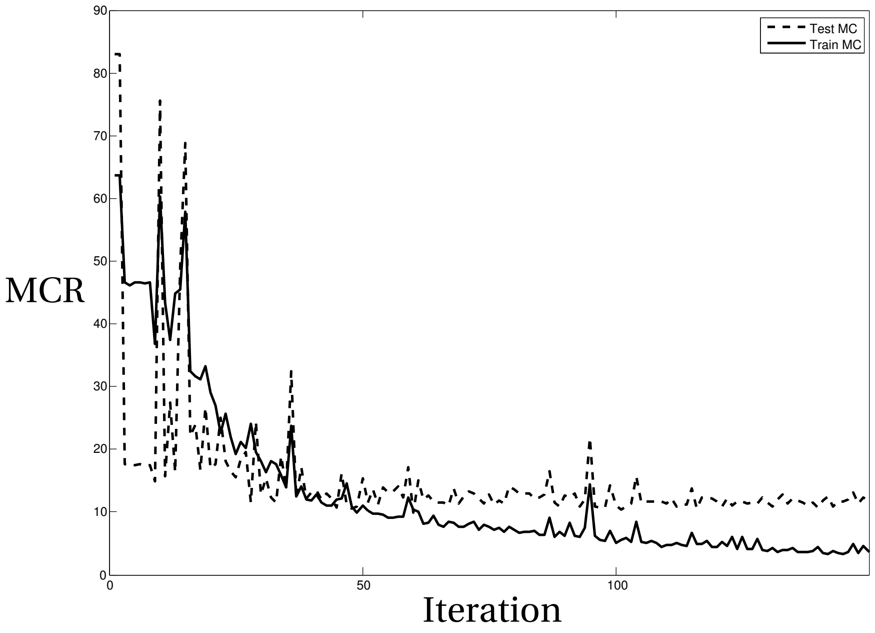


Figure 4.5: Learning curve for classification of airport region targets of resized images

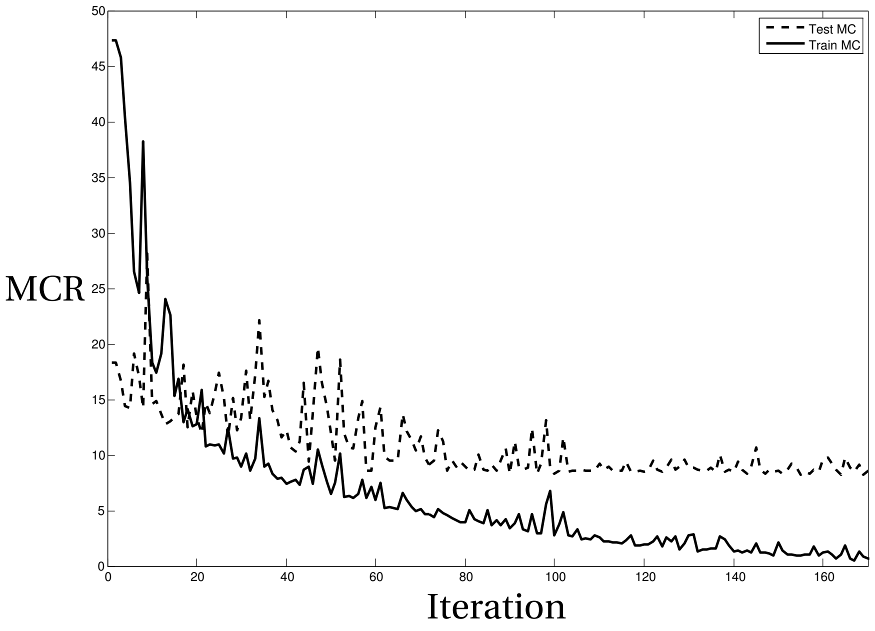


Figure 4.6: Learning curve for classification of airport region targets

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Summary

The convolutional neural network architecture and its internal parameters are examined to get the understanding of the architecture. Although setting the parameters of a deep neural network is referred to as an *art than a science* [57], inspecting within the context of bias-variance guide the tuning to a valid direction with the knowledge of the the underlying factors about the parameter effects. The CIFAR-10 dataset is used to test the implementation and to examine the performance of the architectures with different ingredients. Various activation functions, pooling sizes, filter sizes and depths are tested with the same learning method to observe the outcomes. It is observed that the increase in the depth also increases the performance as long as the overfitting can be avoided. The relation between overfitting and the convolutional network is also considered. The convolutional network with the prior of local connectivity contributes to performance in positive direction.

The deep convolutional neural network architecture for an airplane detection is designed with the know-how gained on the CIFAR-10 dataset. CNN is also utilized for 3-class target classification, namely, airplane, building, and dispersion area. Test set performance of the CNN for satellite image object detection and classification on satellite images is not satisfactory due to the dataset used. The major factor affecting that result is the fact that, the training set is small, which also means the training set images may not be good representatives of the test set cases.

5.2 Conclusion

Based on the test results, it is shown that the Rectified Linear Unit performs better with the average 10% less misclassification rate, observed in CIFAR-10 dataset. The fully connected networks suffer from overfitting that can be deduced from the fact that the training misclassification rate of the 3 layer fully connected network with ReLU activation is less than 1 layer convolutional network followed by 2 layers of fully connected network. However, on the test set, convolutional one outperforms the fully connected network. On the other hand, dropout and preprocessing of the data with ZCA always increase the performance. Relatively similar sized kernels are used to test the effect of the kernel size ordering in the hierarchy. The effect is not drastic but the same kernel size in every layer performs, at least, as well as others. Pooling increases the performance as long as the information loss, which is generated by the pooling, is overcame the benefit gained from the dimension reduction. This can be observed by comparing the test results of the architectures that contains no pooling, 2x2 pooling and 4x4 pooling. 2x2 pooling is the best performing one.

5.3 Future Work

As an application, the deep learning algorithms are preferred as it is a generic algorithm that can be applied to various data. The relationship between the dataset (CIFAR-10, MNIST, ImageNet), dataset type (image, speech, video) and the best performing architectures on the specific dataset can be examined to correlate the task to network architecture.

Unsupervised pretraining of the convolutional networks for various pretraining algorithms can be investigated for the benefits on small datasets.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[2] M. Aharon, M. Elad, and A. Bruckstein. -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.

[3] H. Akcay and S. Aksoy. Automatic detection of geospatial objects using multiple hierarchical segmentations. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(7):2097–2111, July 2008.

[4] F. A. C. Azevedo, L. R. B. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. L. Ferretti, R. E. P. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, 2009.

[5] X. Bai, Q. Li, L. J. Latecki, W. Liu, and Z. Tu. Shape band: A deformable object detection approach. In *CVPR*, pages 1335–1342. IEEE, 2009.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[7] Y. Bengio. *Learning Deep Architectures for AI. Foundations and Trends in Machine Learning, V2(1).* Now Publishers, 2009.

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

[9] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems (NIPS)*, 2007.

[11] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[12] Y.-L. Boureau, J. Ponce, and Y. Lecun. A theoretical analysis of feature pooling in visual recognition. In *27TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, HAIFA, ISRAEL*, 2010.

[13] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009.

[14] H. Cai and Y. Su. Airplane detection in remote sensing image with a circle-frequency filter. volume 5985, pages 59852T–59852T–6, 2005.

[15] P. S. Churchland and T. J. Sejnowski. *The Computational Brain*. MIT Press, Cambridge, MA, USA, 1st edition, 1994.

[16] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets for handwritten digit recogntion. *Neural Computation*, 22(12):3207–3220, 2010.

[17] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[18] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[19] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[20] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[21] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)*, pages 886–893, 2005.

[22] J. J. Eggermont. *The correlative brain : theory and experiment in neural interaction / Jos J. Eggermont*. Springer-Verlag Berlin ; New York, 1990.

[23] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, Mar. 2010.

[24] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 153–160, 2009.

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

[26] K. Fukushima. Neocognitron for handwritten digit recognition. *Neurocomputing*, 51:161–180, 2003.

[27] K. Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

[28] K. Fukushima. Increasing robustness against background noise: visual pattern recognition by a Neocognitron. *Neural Networks*, 24(7):767–778, 2011.

[29] K. Fukushima. Artificial vision by multi-layered neural networks: Neocognitron and its advances. *Neural Networks*, 37:103–119, 2013.

[30] D. Gavrila. A bayesian, exemplar-based approach to hierarchical shape matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(8):1408–1421, Aug 2007.

[31] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[32] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier networks. In *AISTATS*, volume 15, pages 315–323, 2011.

[33] M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. *Trends in Neurosciences*, 15(1):20–25, 1992.

[34] S. Haykin. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2007.

[35] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[36] G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, May 2006.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. Technical Report arXiv:1207.0580.

[39] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[40] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[41] D. H. Hubel and T. N. Wiesel. Binocular interaction in striate cortex of kittens reared with artificial squint. *Journal of neurophysiology*, 1965.

[42] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

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

[44] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[45] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[46] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science*. McGraw-Hill Medical, 4th edition, July 2000.

[47] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL-TR-2008-12-01, Computational and Biological Learning Lab, Courant Institute, NYU, 2008.

[48] E. Kobatake and K. Tanaka. Neuronal selectivities to complex object features in the ventral visual pathway of the macaque cerebral cortex. *J. Neurophysiol.*, 71:856–867, 1994.

[49] E. Kobatake, G. Wang, and K. Tanaka. Effects of shape-discrimination training on the selectivity of inferotemporal cells in adult monkeys. *Journal of Neurophysiology*, 80:324, 1998.

[50] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS 2012)*, page 4, 2012.

[52] H. Larochelle. Capacity of neural network, September 2009. Neural Networks Course Slides, Internet: http://info.usherbrooke.ca/hlarochelle/cours/ift725_A2013/contenu.html.

[53] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

[54] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.

[55] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[56] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann, 1990.

[57] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[58] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[59] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256, May 2010.

[60] 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.

[61] T. S. Lee, D. Mumford, R. Romero, and V. A. Lamme. The role of the primary visual cortex in higher level vision. *Vision Research*, 38(15–16):2429 – 2454, 1998.

[62] Y. Li, X. Sun, H. Wang, H. Sun, and X. Li. Automatic target detection in high-resolution remote sensing images using a contour-based spatial model. *Geoscience and Remote Sensing Letters, IEEE*, 9(5):886–890, Sept 2012.

[63] Z. Li and L. Itti. Saliency and gist features for target detection in satellite images. *IEEE Transactions on Image Processing*, 20(7):2017–2029, 2011.

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

[65] D. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[66] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[67] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[68] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, USA, 1969.

[70] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[71] T. Mitchell. The discipline of machine learning. Technical Report CMU ML-06 108, 2006.

[72] T. Oommen, D. Misra, N. Twarakavi, A. Prakash, B. Sahoo, and S. Bandopadhyay. An objective analysis of support vector machine based classification for remote sensing. *Mathematical Geosciences*, 40(4):409–424, 2008.

[73] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, Aug. 1985.

[74] X. Perrotton, M. Sturzel, and M. Roux. Automatic object detection on aerial images using local descriptors and image synthesis. In A. Gasteratos, M. Vincze, and J. Tsotsos, editors, *Computer Vision Systems*, volume 5008 of *Lecture Notes in Computer Science*, pages 302–311. Springer Berlin Heidelberg, 2008.

[75] N. Pinto, D. D. Cox, and J. J. DiCarlo. Why is real-world visual object recognition hard? *PLoS Comput Biol*, 4(1):e27, January 2008.

[76] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[77] L. D. Raedt. A perspective on inductive logic programming.

[78] M. Ranzato, Y. Boureau, S. Chopra, and Y. LeCun. A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*, 2007.

[79] M. Ranzato, C. Poultney, S. Chopra, and Y. Lecun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2006)*, pages 1137–1144, 2006.

[80] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[81] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.

[82] N. Rochester, J. Holland, L. Haibt, and W. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *Information Theory, IRE Transactions on*, 2(3):80–93, September 1956.

[83] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[84] F. Rosenblatt. *Principles of Neurodynamics.* Spartan, New York, 1962.

[85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[86] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010.

[87] J. Schmidhuber. Deep learning in neural networks: An overview. Technical Report IDSIA-03-14 / arXiv:1404.7828v1 [cs.NE], The Swiss AI Lab IDSIA, 2014.

[88] G. M. Shepherd, editor. *The Synaptic Organization of the Brain.* Oxford University Press, Oxford, 1990.

[89] U. Shrawankar and V. Thakare. Techniques for Feature Extraction In Speech Recognition System : A Comparative Study. *ArXiv e-prints*, May 2013.

[90] J. Sjöberg and L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. In *In Preprint IFAC Symposium on Adaptive Systems in Control and Signal Processing*, pages 669–674, 1992.

[91] X. Sun, H. Wang, and K. Fu. Automatic detection of geospatial objects using taxonomic semantics. *Geoscience and Remote Sensing Letters, IEEE*, 7(1):23–27, 2010.

[92] J. W. Tanaka and M. Taylor. Object categories and expertise: Is the basic level in the eye of the beholder? *Cognitive Psychology*, 23(3):457–482, July 1991.

[93] P. Vincent, L. Hugo, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[94] D. B. Walther and C. Koch. Attention in hierarchical models of object recognition. In P. Cisek, T. Drew, and J. F. Kalaska, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*. Elsevier, Amsterdam, 2007.

[95] D. H. Wiesel and T. N. Hubel. Receptive fields of single neurones in the cat's striate cortex. *J. Physiol.*, 148:574–591, 1959.