Otto-Friedrich-Universität Bamberg

Angewandte Informatik / Kognitive Systeme

Masterarbeit

im Studiengang International Software Systems Science
der Fakultät Wirtschaftsinformatik und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Zum Thema:

Applying Deep Learning for Classifying Images of Hand Postures in the Rock-Paper-Scissors Game

Vorgelegt von:

Rambabu Gupta (Matr. No. 1839122)

Themensteller:
Prof. Dr. Ute Schmid

Abgabedatum:
14.06.2017

Abstract

The goal of our research is to build a model for visual recognition. First, I described Convolutional neural networks, its methods and techniques. Moreover, I also presented briefly the history of deep learning. After that, introduction about few architectures of convolutional neural networks were discussed. The crucial thing about this experiment is that we are dealing with fewer amounts of data for deep learning training and spend less computation power to perform training as architecture is relatively small. I also include several models with different methods and techniques to achieve the best accuracy on CIFAR10 datasets. In the experiment, architecture has to be designed, to classify images of three hand postures of "Rock-Paper-Scissors" game. An intelligent agent (named NAO) plays this game with people. So, deep learning model has to be built for identifying different hand postures of image. Based on results, similar architecture(with techniques and methods) which obtained higher accuracy in that experiment has been used to classify hand postures of the game.

Acknowledgements

I would specifically like to thank Professor Ute Schmid and Christina Zeller for the opportunity to write the master thesis and for the guidance during thesis. I also want to thank them for supervising my research on this project. Additionally, I thank all the people who collaborated during the experiment.

Contents

List of Figures

List of Tables

Abbreviations

CNN ..... Convolutional Neural Networks

MSE ..... Mean Square Error

MLP ..... Multi Layer Perceptron

SGD ..... Stochastic Gradient Descent

SVM ..... Support Vector Machines

NLP ..... Natural Language Processing

GAN ..... Generative Adversial Networks

RNN ..... Recurrent Neural Network

ReLU ..... Rectified Linear Unit

PReLU ..... Parametric Rectified Linear Unit

ILSVRC ..... ImageNet Large Scale Visual Recognition Challenge

FC ..... Fully Connected

GPU ..... Graphical Processing Unit

AWS ..... Amazon Web Services

RPSG16 ..... Rock-Paper-Scissors Game Hand postures(University of Bamberg,2016)

Chapter 1

Introduction

Human beings, the most advanced species of animals, possess better learning abilities than other species. Thanks to the five senses, we can intuitively learn by seeing, hearing, smelling, touching, and tasting. Learning through five senses help humans to gain various necessary skills for their lives such as face recognition, driving a car or language translation. These activities are considered simple for normal people. Computers, a representative of artificial intelligence agents which can solve complex problems, have failed to perform these simple intuitive tasks. Hence, the question arises as to why computers cannot perform these intuitive tasks. How can computers deal with these problem? How can we make computers achieve a high level of accuracy in performing non-algorithmic tasks?

The reason why computers cannot fulfill intuitive tasks is because these tasks cannot be described formally (by some rules or logic). In order to find solutions to these intuitive problems, researchers have come up with various models or algorithms which learn from examples or experiences using multiple machine-learning techniques. Conventional machine learning needed suitable feature vectors and internal representations of raw data. It also required substantial amounts of effort, domain skills and expert knowledge to transform raw data into feature extractors. Afterwards, by using these feature vectors, the machine learning subsystems could detect and classify patterns in the input. In contrast, feature learning or representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification[3]. Feature learning is developed from the theory of probability, statistics, and neural networks. One of the most recent successful versions of feature learning is based on the concept of neural networks. It acquires the feature from examples provided in raw input data (e.g., pixel values for images). For example, when we need to identify cats using multilayer neural networks, then it will learn edges, arrangements of edges, parts of familiar objects regardless of distinct orientation, and the locations and colors of the different images of cats . In the past, in order to build such an image-recognition algorithm, researchers had to design these features manually, then send them to some learning algorithmic machines such as Support vector machines, K-nearest neighbors, Hidden Markov models, etc. Now, there is one advanced version of a multilayer neural network model called deep learning in which the representation learning methods is associated with multiple levels of representations. Each level provides non-linear capabilities to the model and transforms the representation to a more abstract level at the consecutive higher levels. This brings promising results to image and speech recognition processes. The success in performing these simple tasks is a milestone for the

development of more sophisticated machine-learning tasks like driving cars and playing games. "The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning"[1].

The neural network concept was the focus of research in the 1980s. Although deep learning is based on neural networks concepts, it is a relatively new field of machine learning. Why are deep learning algorithms becoming more and more popular now? And how could deep learning algorithms achieve good results? Thanks to the recent success in designing and building effective neural nets with multiple layers using some techniques like convolution, pooling, batch normalization, etc,. We have also access to a substantial amount of data (Internet revolution), and computation power which has dramatically increased with the advent of GPU. Deep learning models are giving significantly better results in the field of computer vision, signal processing, and NLP. It attracted more attention when Google DeepMind deep-learning software was able to defeat a professional player in GO game for the first time in 2016. Deep learning models also even surpassed human levels of accuracy in some problems in image recognition, especially for CIFAR data[4].

I have done experiments on classifying images using CNN (deep learning model) and collected images of three hand postures of the Rock-Paper-Scissors game. We have photographed hand postures of more than 30 students and staff members at the University of Bamberg, and named this set of images as "RPSG16". We have a robot named NAO at the university lab, which would play Rock-Paper-Scissors with humans. Based on the three hand positions of the game, NAO would decide whether he won or lost according to the rules of this game. However, how NAO decides about winning and losing is not the focus of this study. Instead, this thesis tries to identify which deep learning model NAO uses to recognize his opponent's hand posture with significantly high accuracy. Which deep learning architecture is suitable to do that? How do the different positions of layers and their corresponding numbers affect the accuracy of CNN models? Finally, is it possible to get good results using deep learning models with "less" image data?

Considering the structure of this paper, in Section 2 we discuss basic concepts of deep learning and its approaches, then we mention necessary methods to understand the deep learning models. Some relevant definitions of the framework and its relation to experimental hypotheses are described in Section 3. Moreover, this section also presents the empirical study on CIFAR10 data using different CNN architectures and their accuracies. Section 4 provides information about the experiments on RPSG16 data, including details of the questionnaires, experimental procedures, results and discussion. Finally, Section 5 includes some conclusions and suggestions for further research.

Chapter 2

Deep learning - An Overview

2.1 Basic Concepts

Deep Learning is a relatively new sub-field of Machine learning. It is inspired by the neuronal architecture of the brain. It which usually consists of multiple hidden-layers and has fewer connections among the nodes. Deep Learning is also termed as hierarchical learning, feature or representation learning as it does not require hand-crafted feature vectors but, uses raw data instead.

2.1.1 Background and History of Deep Learning

The history of deep learning algorithms can be traced back to 1965, when Ivakhnenko and Lapa modeled the neural networks having multiple layers of non-linear features. Nevertheless, instead of using back-propagation , this model employed polynomial activation function which was analyzed by statistical methods. In 1979, Fukusima created another model consist of multiple convolutional and pooling layers, but it also did not use back-propagation for the training. Rumelhart, Hinton, and Williams in 1985 showed that back-propagation in neural networks yielded distributed representations(connectionism). For this reason, Deep learning is sometimes considered as rebranding of connectionism. In 1989 at bells lab, LeCun had practically applied convolutional networks with backpropagation(LeNet) on handwritten digits. Even the advance was slow in this field in the 1990s, support vector machine(SVM) overshadowed it later.

With the advent of GPUs, the computational speed of computers increased drastically. As neural networks are computationally expensive and requires intensive matrix multiplication, it was not feasible to train deep neural networks before availability of modern GPUs. Neural networks attain much better results than SVM with same amount of data and they also continue to improve with more training data. However, the main problem during the training process of deep neural networks from vanishing gradient, where features in early layers could not be learned because no learning signal reached these layers. In 2006, Hinton and Salakhutdinov suggested one of the first solution given for feed-forward networks by pre-training or initializing features at early layers. As the speed of GPU increased rapidly, it is possible to train deep learning model without using

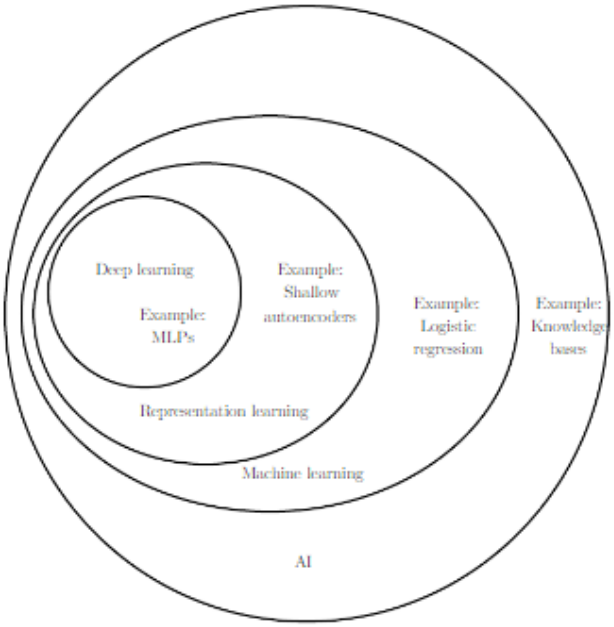


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

Figure 2.1: Venn diagram[1]

pretraining. Krizhevsky, Sutskever, and Hinton used a convolutional networks with rectified linear units and dropout for regularization(AlexNet). They got outstanding results in ILSVRC-2012 ImageNet competition in 2012. With an increase of data, advance in hardware and software technologies as well as techniques to train deeper network model, we are now able to solve some complex problems in the field of NLP, Speech recognition, Computer vision. After Google DeepMind won GO game against humans in 2016 using deep learning models, all major companies such as Facebook, NVIDIA, Microsoft, Google and Intel have started to invest in deep learning startups and created research teams for deep learning in their companies.

Machine Learning is ability to acquire their own knowledge, by extracting patterns from raw data. For instance, machine learning algorithm called naive bayes, can classify legitimate email from spam email. Conventional machine learning algorithms are heavily relied on representation of data they are given. For example, practitioners used logistic regression to recommend casarean delivery and doctors provide several pieces of relevant information for the algorithm. These inputs by doctors included in the representation of patient which is known as a feature. Logistic regression uses these features for predictions. If logistic regression was given as MRI scan instead of doctor reports, it would not able to make predictions using pixels of MRI scan. The process of finding features is called feature engineering. It requires lot of effort and time to find the features. Conventional machine learning algorithms perform quite well in mapping representation to output. But designing these representation or feature vector from raw data (such as pixel values of image) is challenging which requires careful engineering and considerable domain knowledge. The methods of automatically discovering representation from raw data needed for detection or classification is called representation learning.

In representation learning, for example, when analyzing image of car, our main aim is to separate factors of variation like position of car, its color and angle and brightness of the sun. And most applications require us to disentangle factors of variation. To extract such high level, abstract features are not easy. Deep learning solves this problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. For instance, in image recognition, deep learning system represents the concept of image by combining similar concepts, such as corners and contours, which are in turn defined in terms of edges.

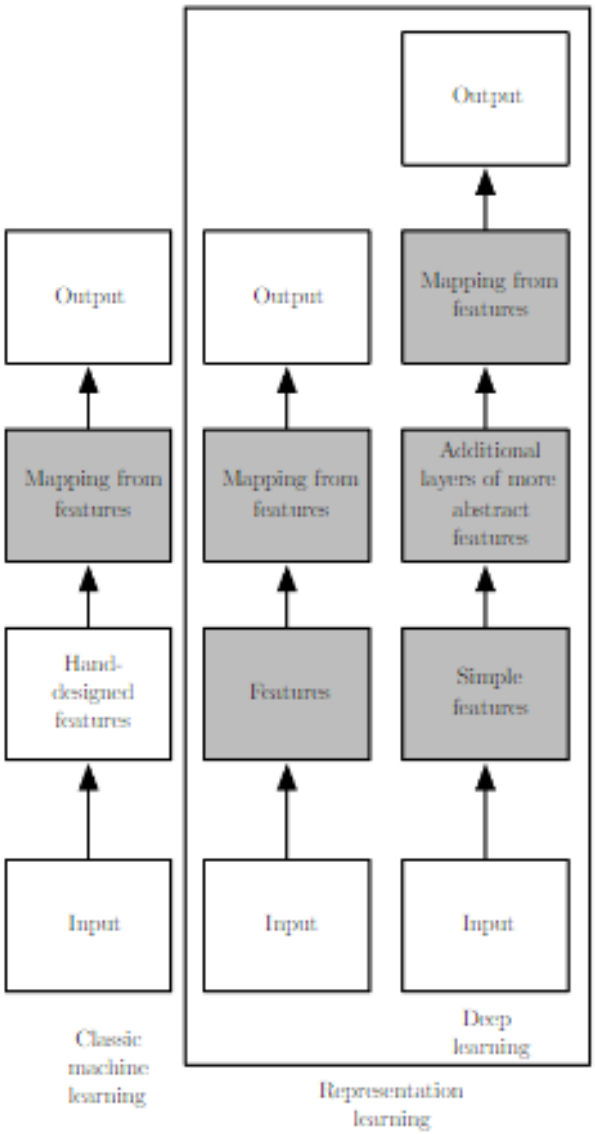


Figure 2.2: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines[1].

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more

abstract level. Picture of deep learning finding edge deep learning

The quintessential example of deep learning model is the feedforward deep network or Multilayer perceptron(MLP).

2.1.2 Feedforward Deep Network (MLP)

A feedforward neural network is a biologically inspired nonlinear classification algorithm. It consists of several simple neuron-like processing units called as nodes and organized in layers. Every node in a layer is connected with all the units of the previous layer. Each connection may have a different weight.

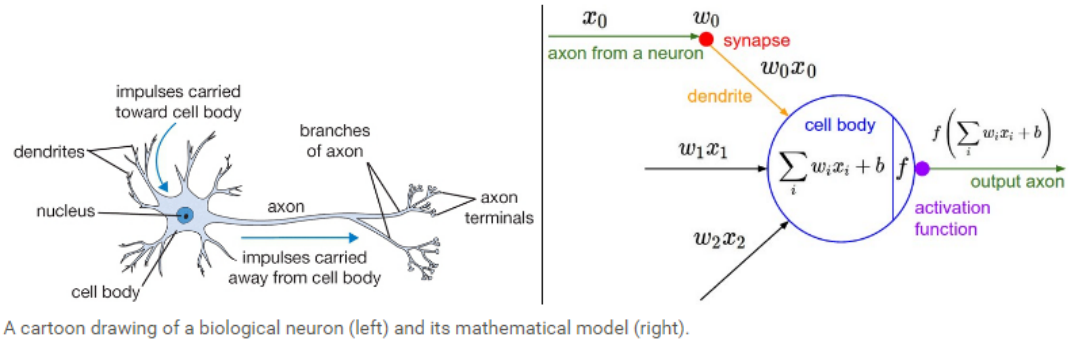


Figure 2.3: Analogy between biological neuron and node of artificial neural networks[2].

The main goal of feedforward deep networks to approximate some function f. For example, for a classifier, $y = f(x)$ maps an input x to a category y. A feedforward network defines a mapping y = $f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation.

y = $f(x;\theta)$ where θ consists of w and b. So, $f(x;w,b) = xw + b$. Here w represents weight and b represents bias in the equation.

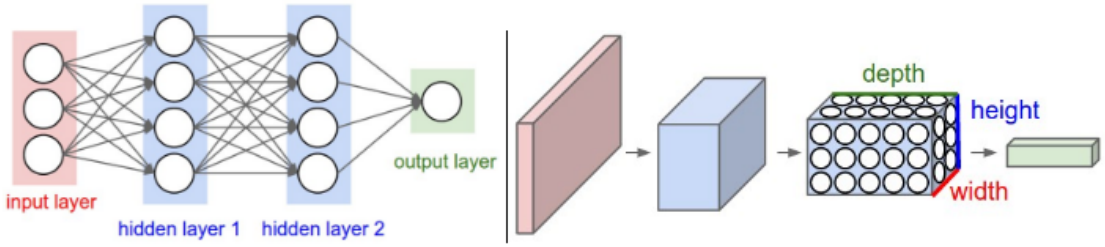


Figure 2.4: Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)[2].

If there will multiple layer, then output of previous layer would be input of current layer. To make model to learn nonlinear function, we use nonlinear function called activation function. Activation function can be sigmoid, tanh or rectified linear unit. Modern Neural networks use activation function called rectified linear unit. It is defined as g(z) = max{0,z}. Typical non-linear function are sigmoid, logistic and rectified linear unit.

A three-layer neural network could analogously look like $s = W3max(\theta, W2max(\theta, W1x))$, where all of W3,W2,W1 are parameters to be learned and s is a scores of visual categories. In the below picture, It has been briefly shown that how different nodes are connected with each in typical three-layer neural network and methods to get error derivatives.

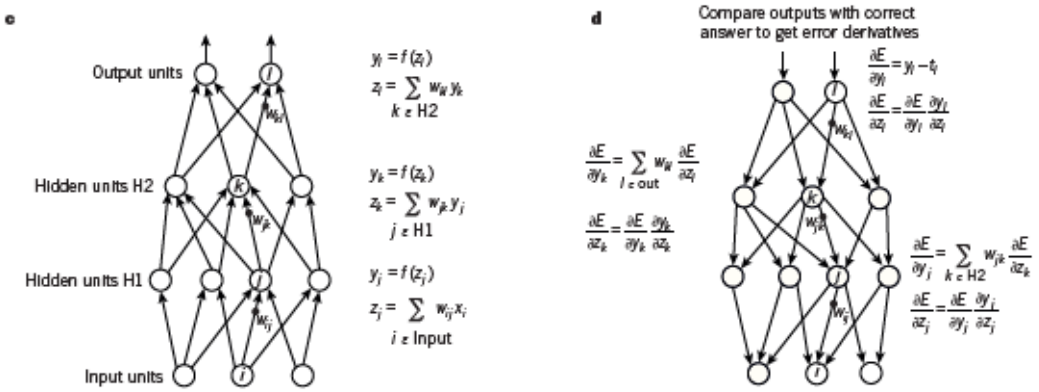


Figure 2.5: Multilayer neural networks and backpropagation[3].

A multilayer neural network (shown by the connected dots) can distort the input space to make the classes of data (examples of which are on the red and blue lines in 2.6) linearly separable. Note how a regular grid (shown on the left) in input space is also transformed (shown in the middle panel) by hidden units. This is an illustrative example with only two input units, two hidden units and one output unit, but the networks used for object recognition or natural language processing contain tens or hundreds of thousands of units[3].

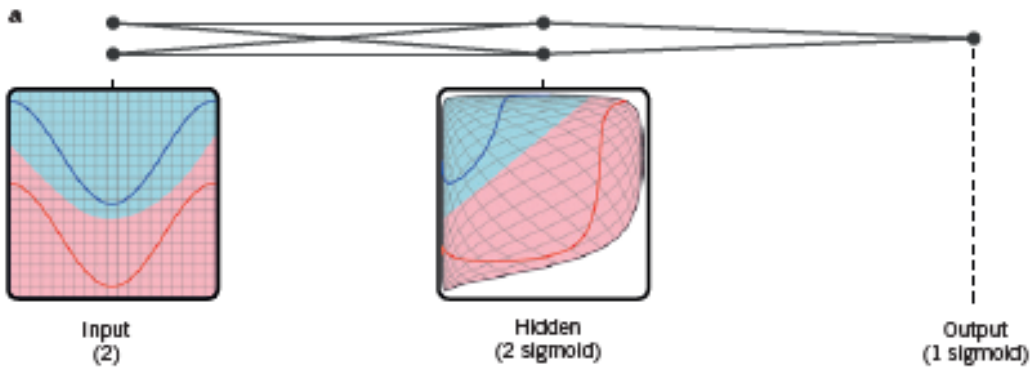


Figure 2.6: Picture of input changing in MLP[3].

Architecture of deep feedforward model can be different size and depth depending on the practical problems. If there are more nodes in architecture, then it would increase

space of representation functions. Networks with more node can express more complicated function(shown below in figure 2.7). And network would able to classify more complicated data. But it also raises overfitting issue.

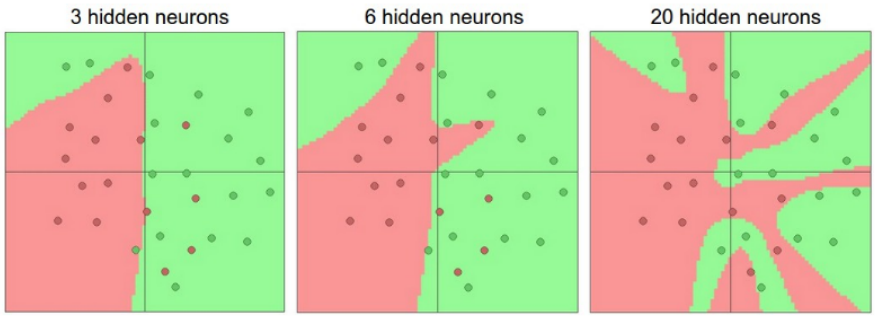


Figure 2.7: Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath[2].

It is shown through empirical study that if the model has sparse connection between their nodes of layers then test accuracy further increases[1](figure: 2.8). A special type of deep feedforward network which generalize better than networks with fully connected adjacent layers, used sparse connection between nodes of layers and some other specialized techniques is convolutional neural network. It is discussed in detail at later sections. If the depth size of neural network is more then it increases test accuracy. So, if we increase number of parameters in neural network like in CNN then it would improve test accuracy as well as require less computation and time to train the model.

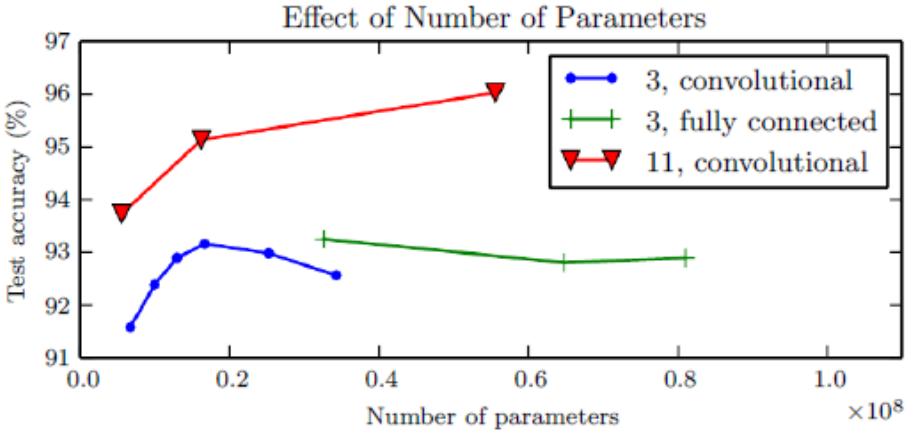


Figure 2.8: Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow et al. (2014d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance. The legend indicates the depth of network used to make each curve and whether the curve represents variation in the size of the convolutional or the fully connected layers. [1].

Stochastic Gradient Descent Stochastic gradient descent is iterative way for stochastic approximation of gradient descent optimization method for minimizing loss function. SGD is effective when training steps use mini-batch. Gradient of the loss over a mini batch is an estimate of the gradient over training sets, which quality improves as the batch increases and also computation over a batch is more efficient than individual examples.

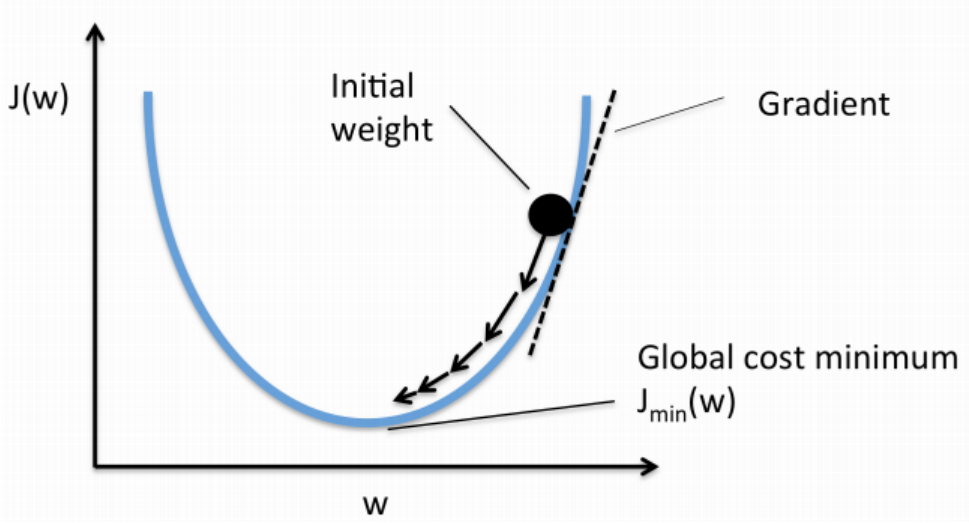


Figure 2.9: Analogy: A person is stuck in the mountains and is trying to get down (i.e. trying to find the minima). SGD: The person represents the backpropagation algorithm, and the path taken down the mountain represents the sequence of parameter settings that the algorithm will explore. The steepness of the hill represents the slope of the error surface at that point. The instrument used to measure steepness is differentiation (the slope of the error surface can be calculated by taking the derivative of the squared error function at that point). The direction he chooses to travel in aligns with the gradient of the error surface at that point. The amount of time he travels before taking another measurement is the learning rate of the algorithm. Source: `http://sebastianraschka.com/images/faq/closed-form-vs-gd/ball.png`

2.1.3 Types of Deep Neural Network Architectures

Convolutional Neural Network (CNN)

Convolutional neural networks are specialized kind of neural networks. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. According to Yann LeCun, there are no fully connected layers in a convolutional neural network and fully connected layers are in fact convolutional layers with a 1×1 convolution kernels. However, non-linearity is built within the neurons in fully connected layer, but there is separate non-linearly layer in CNN. CNN typically consists of three types of layers such as convolutional layers, activation layers, pooling layers and fully connected layers. CNN usually takes input as 2D/3D matrix for learning. It is incredibly successful in many practical application like image recognition, NLP or signal processing. It usually consist of convolutional layer, pooling layers, dropout layers, fully connected layers, classifier layers(SVM or Softmax) etc.

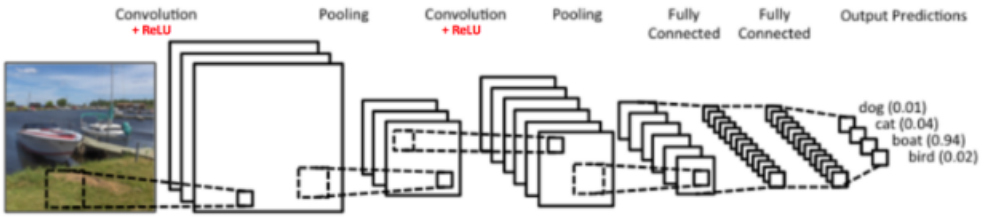


Figure 2.10: Typical CNN architecture

Deep Reinforcement Learning In some problems, Deep learning can be applied with reinforcement learning to learn complex things directly from raw data. For example, deep learning model wants to learn a game directly using the moving pixel values of the agents. So, Deep learning model like CNN would learn their own knowledge from raw data such as vision tasks in game and how object is playing their moves could be learned by reinforcement learning. Reinforcement learning is based on rewards or punishments by learning moves with trial-and-error method. One of most popular reinforcement learning is Q-learning. Combining Q-networks with deep learning models is termed as Deep Q-Networks. It stores all of the agent's experiences and then randomly samples and replays these experiences to provide training data. This model could be to used to learn Atari game and also similar kind of approach had been used to learn GO game. DeepMind's deep learning algorithm was able to defeat human player in GO game in 2016.

Generative Adversial Networks It is first introduced by Ian Goodfellow in 2014. It consists of two neural networks models namely generator and discriminator. Generator takes noise as inpute and generate samples. Discriminator receives samples from generator as well training data and distinguish two different sources. Generator learns to produce more and more realistic samples and discriminator becomes better in distinguishing generated and real data. These two networks are trained simultaneously and also GAN can backpropagate gradient information from discriminator back to generator network, so the generator can adopt its parameters. The main goal of networks to generate samples indistinguishable from real data. It is widely applied for modelling natural images.

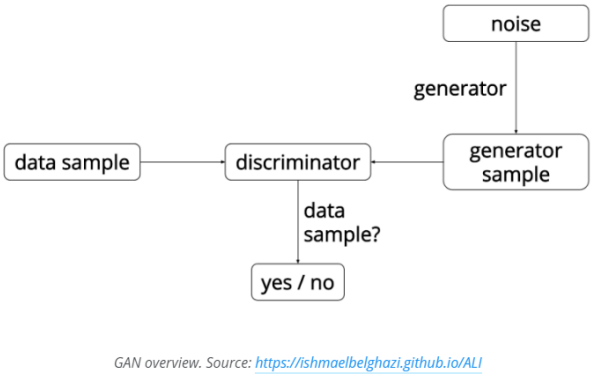


Figure 2.11: Generative Adversial Networks overview

Recurrent Neural Network (RNN)

Recurrent neural networks are special type of neural network architecture used for sequential data. It use parameter sharing concept in which each output is a function of previous member of output. It is another variant of neural network which is applied usually on one dimensional, sequential data. RNN produces out at each time event. Here "time" is not real time. It is sequential in a way. It can be applied for solve problem which have sequence like natural language processing. Sometimes integrating both CNN and RNN models solve complex problems(figure: 2.12).

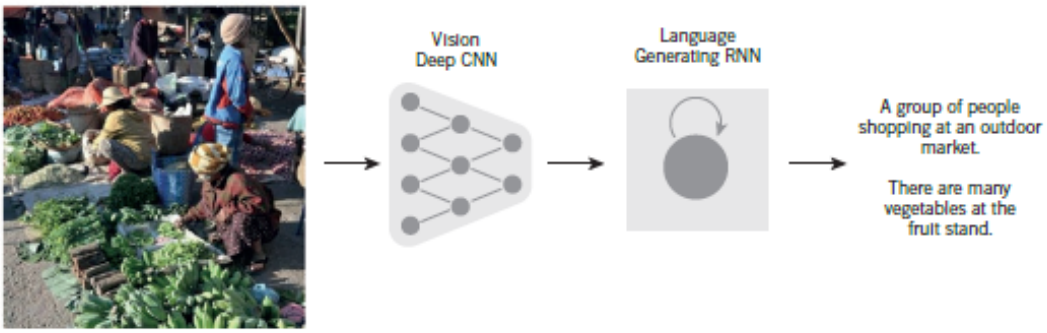


Figure 2.12: CNN and RNN both are applied to find logical textual information directly from images. CNN does identify/Classify the images and RNN generates meaningful sentences out of it.

2.2 Approaches and Methods

2.2.1 CNN Approaches

There are a few most important approaches which has been most widely used while building CNN architecture by practitioners. These approaches could be used in CNN architecture based on amount of training data, computational power and other factors. Some of common approaches are discussed below.

Convolution layer CNN must have at least one convolutional layers in the architecture. It is based on concept of convolution which is special type of mathematical operation on two functions. Lets suppose if we have 2 dimensional as input denoted by I, then there would be 2D kernel denoted by K and feature map or output denoted by S . Equation for convolution would be below:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Figure 2.13: Convolutional matrix equation[1]

Convolution can be understood by thinking sliding window function applied to a matrix.

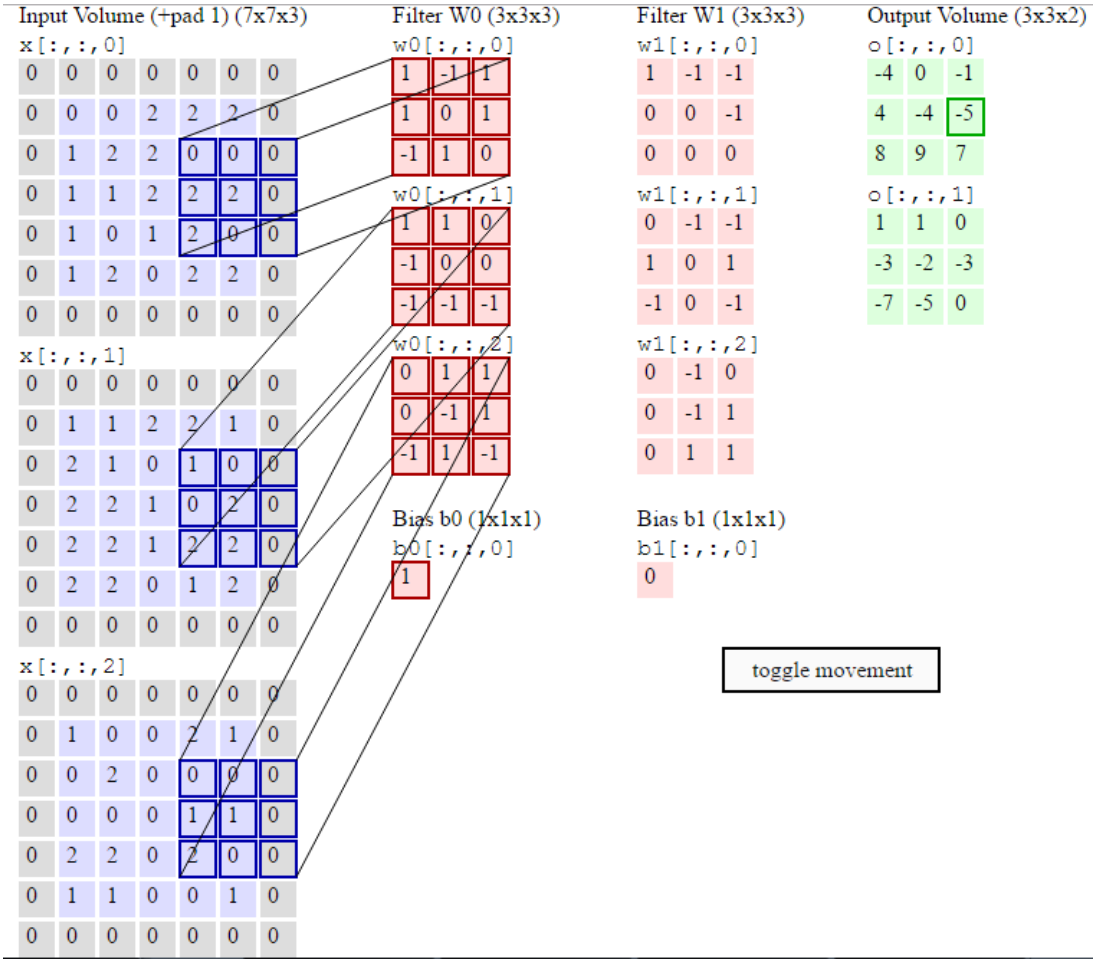


Figure 2.14: The visualization below iterates over the output activations (green), and shows that each element is computed by elementwise multiplying the highlighted input (blue) with the filter (red), summing it up, and then offsetting the result by the bias[2]

Convolution leverages three ideas to improve learning system: sparse interactions, parameter sharing and equivalent representation.

1. **Sparse interaction**

   In CNN, the kernel is quite smaller than input. In case of images, small and meaningful features can be detected by small kernel for size 10 or 100 pixels for input images which contain millions of pixels. Also in CNN, there is few connections between nodes so computation time significantly decreases and it makes convolutional layer spatially sparse sparse. When dealing with high-dimensional data such as images, it is impractical to connect neurons to all neurons. Instead, it connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter which is called as the receptive field of the neuron (equivalently, this is the filter size)[2]. The depth axis is always equal to the depth of input volume. For example, suppose that the input of size $[32 \times 32 \times 12]$. If the filter size of $[3 \times 3]$, then each neuron of convolutional layers will have weights to $[3 \times 3 \times 12]$ region in input volume(depth size is 12 here).

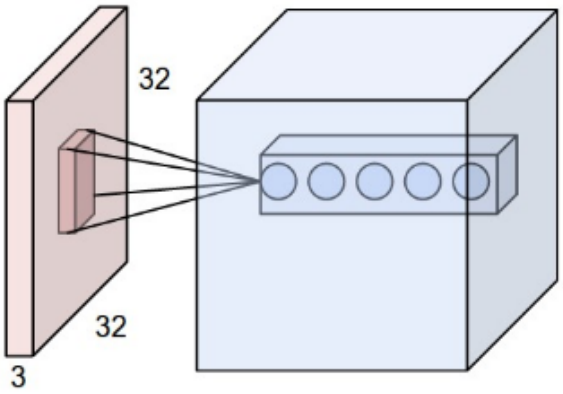


Figure 2.15: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below[2].

There are three hyperparameters which controls the size of output volume: the depth, stride and zero-padding. Depth corresponds to number of filters to be used. Each filters learns something different in the input. How we slide the filter in layers is called stride. If "stride" is 1, then filter will move by one pixel(in case of images). Higher the stride would produces smaller output volumes spatially. To control the size of output volumes, sometimes one need to pad input volume with zeros around the corner and this technique is called zero-padding.

2. **Parameter sharing**

   Parameter sharing is used in convolutional layer to control the number of parameters. It dramatically reduces the number of parameter by making one assumption. The assumption is that if one feature is useful to compute some spatial position (x1,y1) then it should also be useful at some different position (x2,y2). So, each member of kernel is used at almost all position of input. In this way, we learn few parameters compare to learning separate parameters for every position of inputs.

3. **Equivalent representations**

   Function is called as equivariant if it's input changes, the output changes in same way. When processing images using CNN, Convolution create 2D map of certain feature of input. If we move object in the input, its representation also moves in same amount in the output. In convolutional layers, if it detects edges and same type of edge found everywhere in image. So, it is plausible to share parameters across the entire image.

Spatial pooling Pooling layers are generally used between two convolutional layers. The function of pooling is to reduce number of parameters and computation in the CNN architecture. It operates on every depth slices of input and resizes spatially. In this way, it reduces the spatial size of representation. In case of max pooling, it resize using 'MAX operation'. 'MAX operation' would only input maximum of matrix on given filters.

The paper "Striving for Simplicity: The All Convolutional Net"[5] propose that CNN is performing well even without use of pooling layers and larger stride have been used to reduce size of representation. So, CNN architectures may not have pooling layers in future.

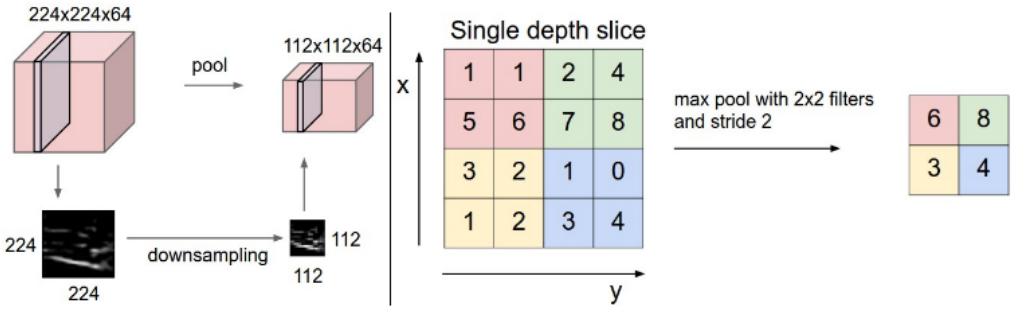


Figure 2.16: Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square)[2]

Batch normalization Batch normalization is method of performing the normalization for each training mini-batch. It allows us to use much higher learning rates and be less careful about initialization. Batch normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by significant margin in terms of accuracy[6]. Batch normalization solves the problem called internal covariate shift. Covariate shift means that distribution of features is different at different parts of data. If this happens in network from one layer to another, then it is called as internal covariate shift. This happens because, as the network learns and the weights are updated, the distribution of outputs of a specific layer in the network changes. This forces the higher layers to adapt to that drift, which slows down learning. Batch normalization helps to reduce internal covariate shift. It accomplishes this via a normalization step that fixes means and variance of layer inputs.

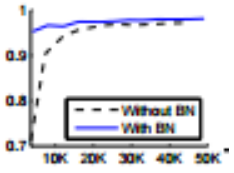


Figure 2.17: The test accuracy graph of the MNIST network trained with and without Batch Normalization[6]

Batch normalization also regularizes the model and reduces the need of dropout. It helps network to train faster and achieve higher accuracy[6].

Dropout layer Deep neural networks with large number of parameters are very powerful machine learning systems. However, Overfitting is serious problem in such networks.

Dropout is technique for randomly dropping out input from neural nets during training. This prevents from co-adapting too much. Hence, it prevents overfitting in the network. Dropout also provides a way to combine exponentially many neural network architectures efficiently.

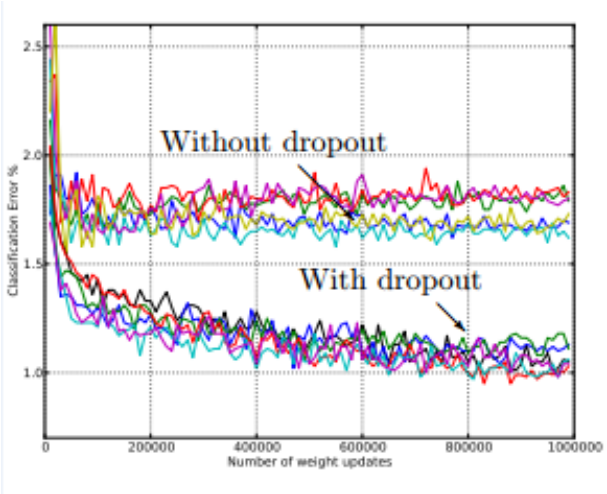
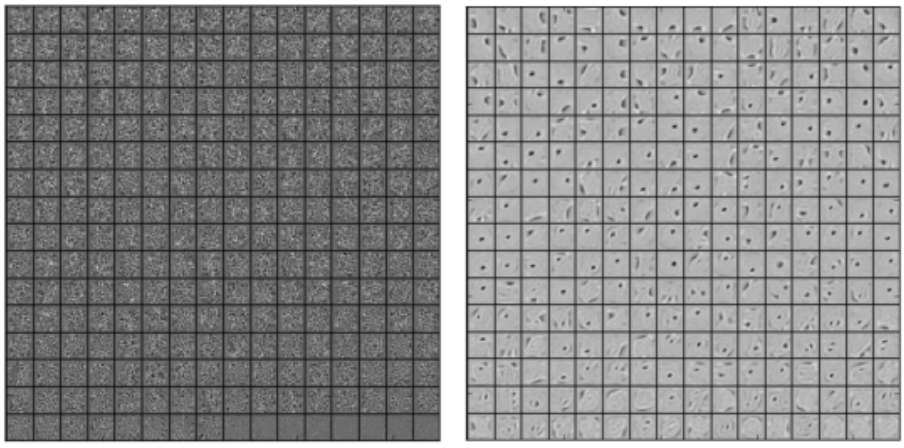


Figure 2.18: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 20148 units.[7]

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout. A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$ or less. For each presentation of each training case, a new thinned network is sampled and trained. So, training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

Figure 2.19: Effect on features by dropout on MNSIT data with one hidden layer autoencoders having 256 rectified units[7]

If dropout are applied near fully connected layers of CNN. Its called as fully connected dropout. If dropout is used at max pooling layer then its called as Max-pooling dropout. Both increase accuracy if applied on convolutional networks. Generally, CNN layers with max-pooling dropout layer and fully connected drop out layers had shown better results than CNN architecture without max-pooling layers. However, it is also shown that models which contains only convolutional performed quite well.

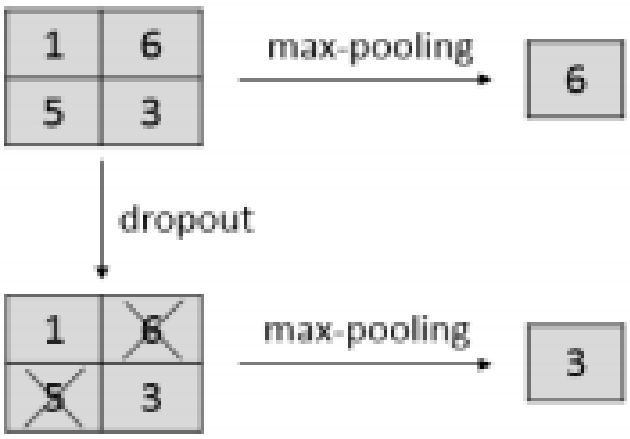


Figure 2.20: Max-pooling dropout[8]

Rectifiers Rectifier linear units are essential for state-of-the-art neural networks. It is activation function and used for non-linearity in the neural network model. It overcomes vanishing gradient descent problem unlike tanh or signmoid function. It is defined as $f(x) = max(\theta, x)$. PReLU, the variant of ReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. One of model which used PReLU for CIFAR10 and imageNet data had surpassed human-level performance(5.1%)[4] on visual recognition challenge.

Fully connected Fully connected layer is traditional multi-layer perceptron. A fully connected layer is a kind of convolutional layer with 1×1 convolutional kernel as per Yann LeCun. It usually placed at the end of CNN architecture. It takes one dimensional matrix as input. The output of convolutional layers and pooling layers represent high level feature and low dimensional feature space, and fully connected is learning a nonlinear function in that space. The main task of fully connected layer is to classify input data into various classes using features which is output of convolutional layers.

Classifiers Classifiers are generally used on top of fully connected layers of CNN. Most widely used classifiers is softmax activation function in the output layer. It could use other classifiers like SVM also. Softmax is activation function. It provides probability of each classes in the output which sum to one for all classes/groups and cross-entropy loss is what we use to measure loss error at a softmax layer.

2.2.2 CNN Training Methods

Regularization Regularization is strategies required to reduce test error, sometimes at cost of increased training error. One of the main problem in machine learning is develop an algorithm which can perform well not just on training data, but also in new data. These strategies reduce generalization error, but not its training error. Regularization is one of the methods to build those kind of algorithm. Strategies like putting constraints on model, such as adding restriction on parameters values or also add extra term in objective functions. These constraints and penalties do promote generalization in machine learning model and prevent overfitting. There are several regularization strategies, some of important regularization methods of deep learning are discussed below.

1. L1 and L2 Regularization

   A "Regularizer term" has to be added in loss function in order to prevent the coefficients to fit so perfectly to overfit. L1 regularization is sum of the weights and L2 regularization is sum of the square of weights.

L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} |w_i|$$

L2 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

Figure 2.21: L1 and L2 regularization terms in mathematical formula

2. Dataset Augmentation

   One of best approach to prevent overfitting in deep learning models is to use more and more data. As sometimes, we have limited amount of data. One way to get around this problem is to create fake data out of existing data. In case of object recognition, this trick works quite well. One can generate multiple images from one images by randomly changing its orientation, contrast, or colour etc.

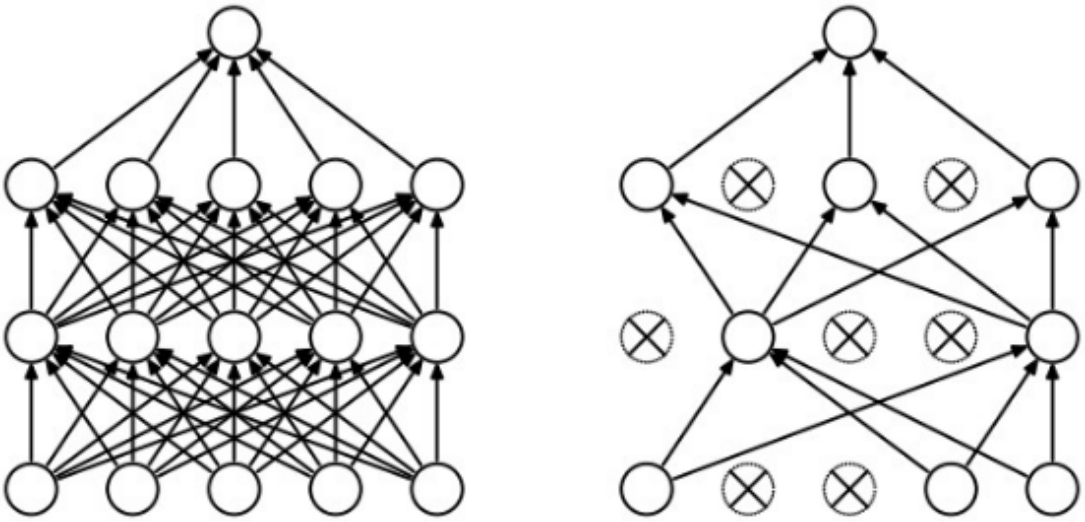
3. Early Stopping

   When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again[1]. Thats make U-shape graph. This means we can obtain a model with better validation set error (and thus, hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error. Instead of running our optimization algorithm until we reach a (local) minimum of validation error, we run it until the error on the validation set has not improved for some amount of time. This strategy is called early stopping. One of the hyperparameters of early stopping is number of training steps. It also

reduces computational cost as it limit the number of training steps. It is unobtrusive form of regularization, in that it requires no change in objective function or set of allowable parameters.

4. Dropout

   Dropout can be method of applying bagging. Bagging is regularization method in which several models are trained, then all the models vote to output the test examples. It reduces generalization error. Bagging is impratical in neural networks as training very large neural networks need runtime and memory. Dropout make it possible by bagged ensemble of exponentially many neural networks. Every new input, Dropout randomly removes some nodes from layers.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.22: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.[7]

5. Learning Rate as Hyperparameter

   Learning rate is one of hyperparameter used in neural networks. It could be one of regularization methods. So, choosing optimal learning rate do impact overfitting and improves generalization on test set(figure: 2.23). It is one of the preferred way to control overfitting in the model caused by using high number filters.

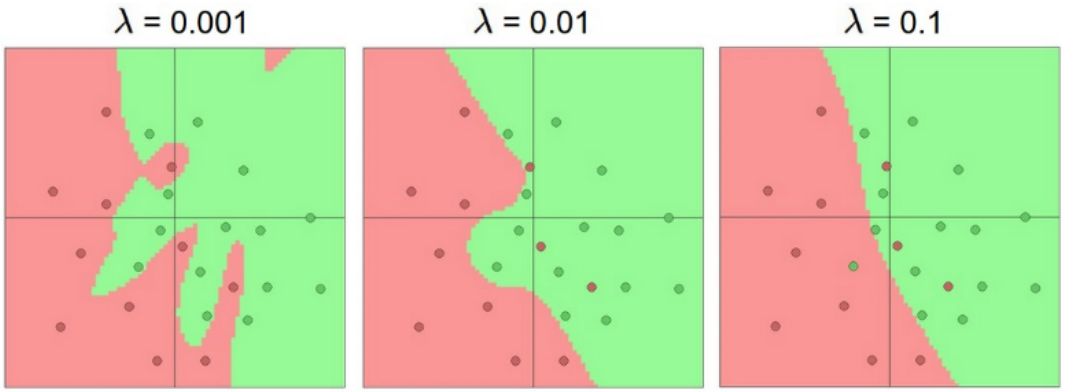


Figure 2.23: The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization.[2]

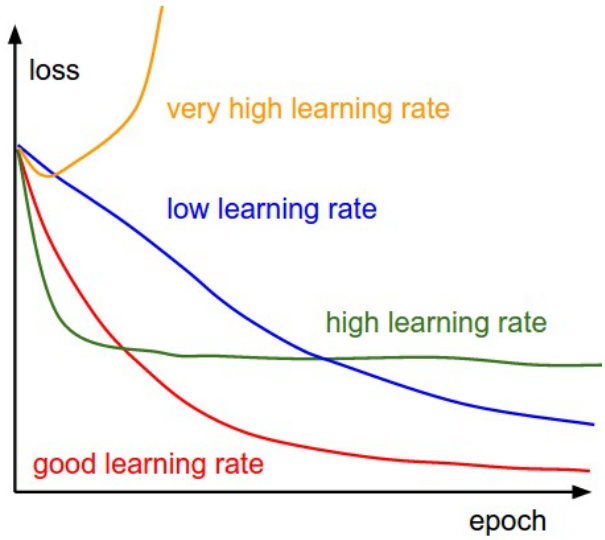


Figure 2.24: A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line).[2]

6. Validation/Training Accuracy

   The gap between training and validation accuracy can be useful to track overfitting in the model. If the gap is increasing then there is more overfitting in the model and there is need of change hyperparameters or network topology. It also arises need for more data. So, using validation set in training is an important methods to know more about model during training. In this way, we can tune our model hyparameters in effective way.

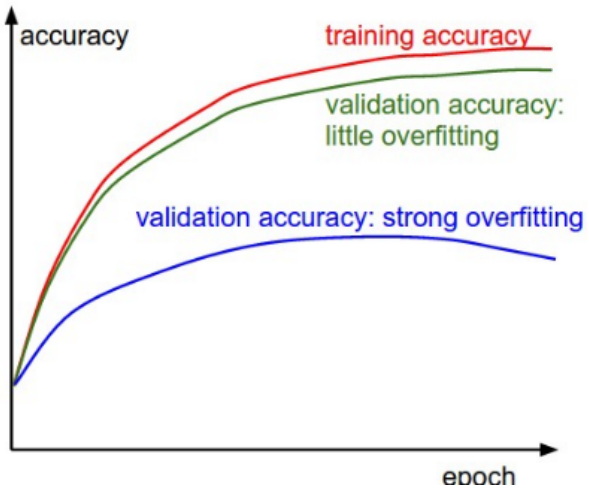


Figure 2.25: The gap between the training and validation accuracy indicates the amount of overfitting.[2]

Optimization Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x. One particular case of optimization: finding the parameters θ of a neural network that significantly reduce a cost function $J(\theta)$, which typically includes a performance measure evaluated on the entire training set as well as additional regularization terms[1]. Typically, the cost function can be written as an average over the training set, such as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

Figure 2.26: Cost function[1]

where L is the per-example loss function, $f(x;\theta)$ is the predicted output when the input is x, $pdata$ is the empirical distribution[1].

There are two most used loss function which are discussed below.

1. Mean Square Error (MSE)

   It is a multi class loss formerly used to train neural networks.

   $$Loss(x, y) = \sum_i |x_i - y_i|.$$

   with x be the vector denoting values of n number of predictions. Also, y be a vector representing n number of true values.n.

2. Cross Entropy

   It is usually preferred more than MSE for neural networks training.

$$H(p, q_\theta) = \mathbb{E}_p[-\log q_\theta]$$

Figure 2.27: Cross Entropy formula

where p is the true distribution and q is the model distribution. As before, the model q is parameterized with θ (i.e. these are the network weights).

There is chance that initialization of the parameters result in the network being decisively wrong for some training input. The output neuron would saturate near to 1 instead of 0 or vice versa. The MSE loss will usually slow down learning. But cross entropy do not get any impact from this.

Optimization algorithms in machine learning does not usually halts at local minimum. Instead, It minimizes a surrogate loss function but halts when early stopping is satisfied. Optimization algorithm that uses entire training set is called batch gradient methods and if it uses more than one training examples but not all example at once, then it called as minibatch stochastic methods. Most of time, stochastic gradient methods are applied in deep learning models for optimization. There are few advantages of using stochastic methods. Large batches provide a more accurate estimate of gradient, considerable reduction in runtime and also it utilizes the multicore architectures of computers by using all core to process the batches in parallel. But large batches requires more memory and large batches can not offer regularizing effect like smaller batches. So, optimum size of minibatch should be used for training deep learning models.

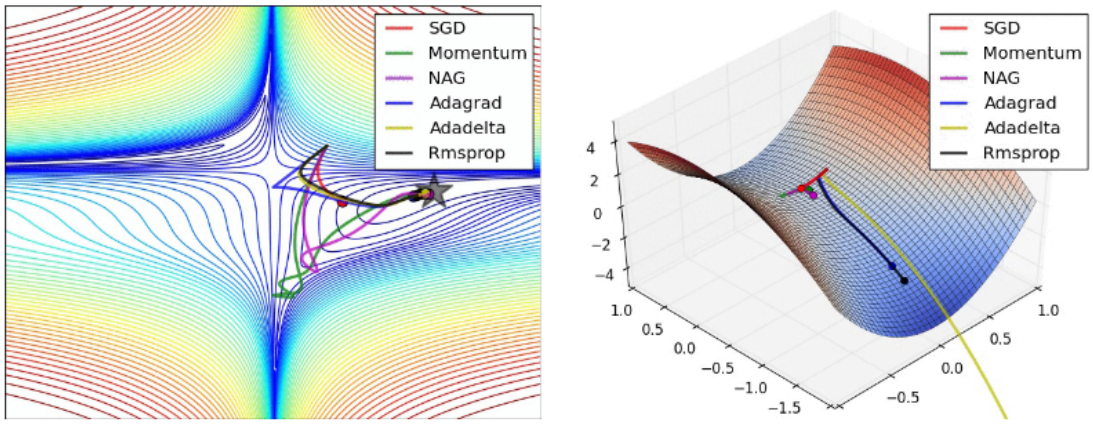


Figure 2.28: Animations that may help your intuitions about the learning process dynamics. Left: Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. Right: A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed. Images credit: Alec Radford.

There are many optimization algorithms like Stochastic gradient descent, Momentum, Nestrov momentum. Most popular one is stochastic gradient descent. There are also algorithm which supports adaptive learning algorithm and perform quite well compare to algorithm having absolute learning rate. AdaGrad, RMSProp and Adam are most popular optimzation algoritm with adaptive learning rates(figure: 2.28). The RMSProp

algorithm modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average[1].

Transfer Learning Training neural networks is computationally expensive as well require lots of data. It is not convenient for everybody to invest time and resources to train the neural network. Due to this, it is good idea to somehow use pre-trained model from other people with small changes. Transfer learning enable us to use pre-trained model to accelerate our solution. Pre-trained model is created by someone else to solve similar problem. For example, there are many pre-trained model available based on imagenet data of 1.2 million images of 1000 categories. These pre-trained models have ability to generalize to images outside the ImageNet datasets via transfer learning. The modification on pre-existing model has been done by fine-tuning the model. Two ways of fine-tuning the mode have described below.

1. Feature extraction

   Just removing the output layer(the one which gives the probabilities for being in each of the 1000 classes) and then use the entire network as a fixed feature extractor for the new data set.

2. Use the Architecture of the pre-trained model

   We can only use the architecture only. Just initializes the weights and train the model on new data.

2.3 Convnets Architectures

There are several architectures based on convolutional neural networks. Some of architectures which is quite common and influential are listed below.

LeNet(1990s) LeNet was one of the first CNN which used usually for character recognition tasks such as reading zip codes, digits etc. LeNet architecture consists of alternate convolutional layers always immediately followed by pooling layer. Convolutional, pooling and non-linearity has been used in the architecture. Least mean square is employed as loss function in LeNet.

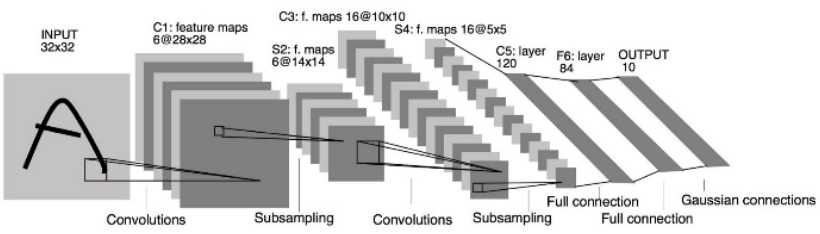


Figure 2.29: LeNet-5: First convolutional architecture for digit recognition. Courtesy: `https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba`

AlexNet (2012) The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton in 2012. It significantly outperformed other hand crafted machine learning algorithm in ImageNet ILSVRC 2012 challenge. It is a more deeper , wider and featured consecutive convolutional layers version of LeNet. ReLU has been used in the model for non-linearity. Stacked convolutional layers and overlapping max pooling are applied in the model.

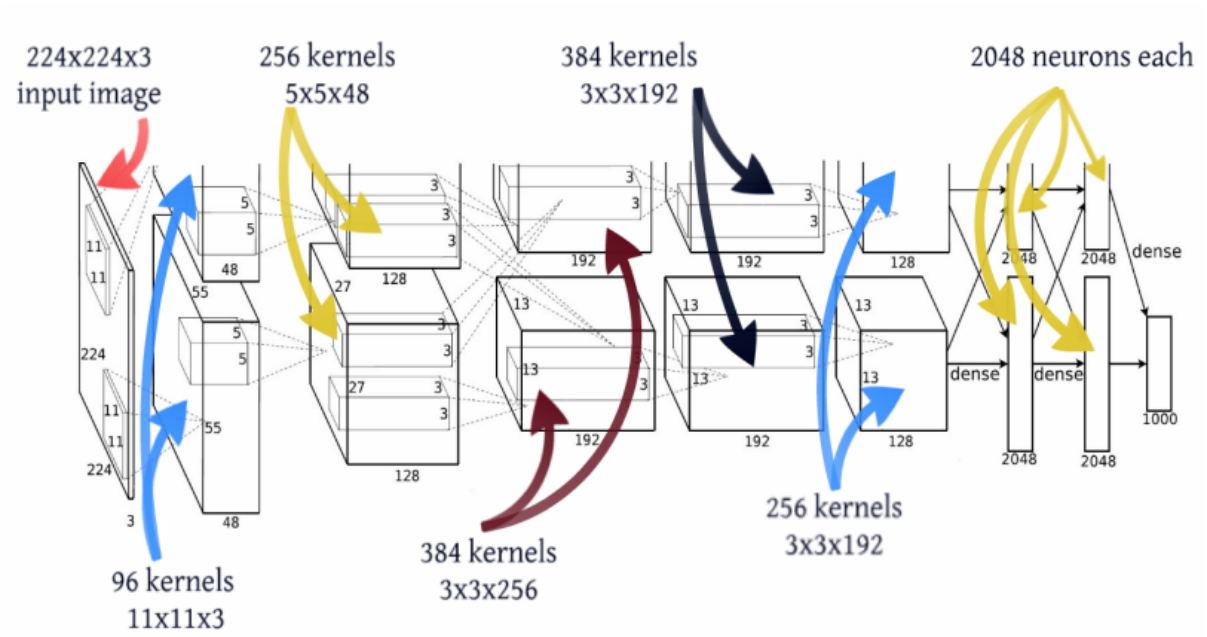


Figure 2.30: CNN Architecture: AlexNet. It contains 7 hidden layers, 650 thousand neurons, 60 million parameters and 650 million connections. Courtsey: `http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf`

GoogLeNet (2014) GoogleNet was winner of ILSVRC 2014. Its main contribution was development of inception module which reduces number of parameters to 4m from 60m(AlexNet).

VGGNet(2014) VGGNet was runner up of ILSVRC 2014. Its main contribution was to show that depth of network is an important factors for the overall performance of CNN model. It consists of 16 conv/FC layers and performs 3*3 convolution and 2*2 pooling across the layers.

DenseNet(2016) DenseNet is a network architecture where each layer is directly connected to every other layer in a feed-forward fashion (within each dense block). For each layer, the feature maps of all preceding layers are treated as separate inputs whereas its own feature maps are passed on as inputs to all subsequent layers. This connectivity pattern yields state-of-the-art accuracies on CIFAR10/100 (with or without data augmentation) and SVHN.

2.4 Applications and Evaluation Methods of Deep Learning

2.4.1 Applications

Now a days, Deep learning are being applied in many fields such as computer vision, speech recognition, Natural language processing and other applications related to commercial uses.

Computer Vision Computer vision is interdisciplinary field which deals on how computers process, analyze, understand and create images or videos. Most of computer vision such as image recognition is quite easy for humans but challenging for computers. Deep learning is giving quite promising results in the field of computer vision such as object recognition, object segmentation, annotating an image or transcribing symbols from an image. Convolutional neural networks are usually applied to solve computer vision problems.

Speech recognition Speech recognition is interdisciplinary field in which computers can translate or recognize the spoken words into text. It is also called as automatic speech recognition or computer speech recognition. The task of speech recognition is to map an acoustic signal containing a spoken natural language utterance into the corresponding sequence of words intended by the speaker[book]. Recently deep learning models like deep LSTM RNN is giving quite good results compare to previous machine learning algorithms.

Natural Language Processing Natural language processing (NLP) is the use of human languages, such as English or French, by a computer. NLP is more about extracting meaning out of spoken words. Machine translation is one of application of NLP. It converts one human language to another human language using computers. Sometimes, very generic neural networks can be applied on to natural language processing. However, to achieve good performance in some NLP problems like machine translation, Deep RNN could be applied.

Other Applications There are many applications of deep learning like cancer detection, autonomous vehicles, music generation, photo description, real time analysis of human behavior etc.

2.4.2 Evaluation methods

Performance Measure Performance measure is a measurement which make predictions by a trained model on test dataset. It is specialized for different problems like classification, regression, and clustering. Classification accuracy which sum of all positives divided by total prediction is used for classification problems. Sometimes, we need to tweak or breakdown performance measure to gain better insights into data.

Accuracy is defined by total number of true positives and true negatives divided by total population(figure: 2.31). In general Positive = identified and negative = rejected. Therefore:

1. True positive = correctly identified
2. False positive = incorrectly identified
3. True negative = correctly rejected
4. False negative = incorrectly rejected

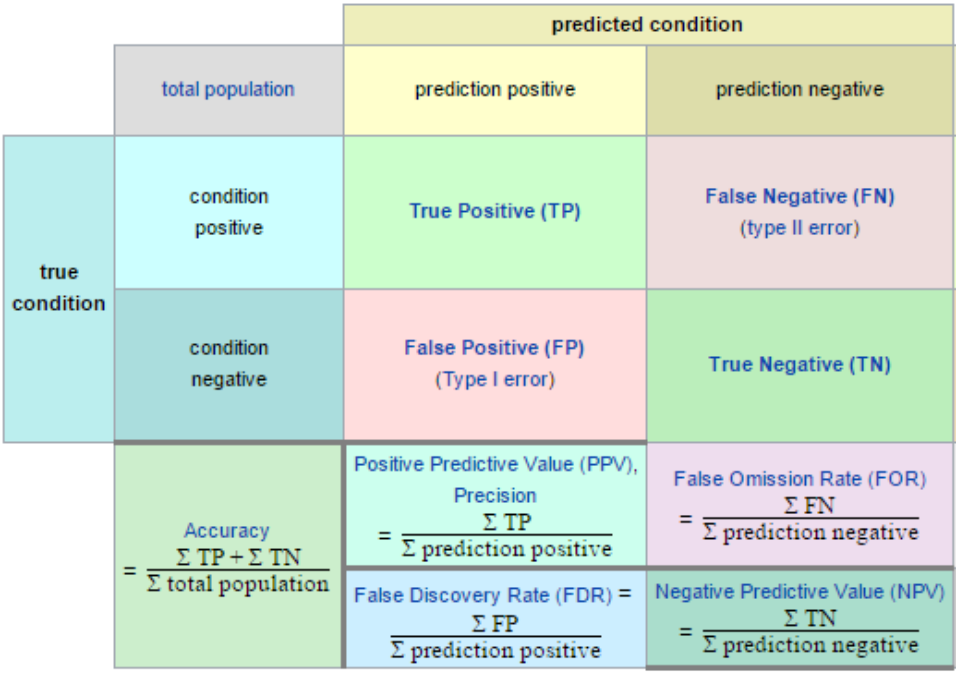


Figure 2.31: Confusion Matrix : Based on different scenario, One can choose performance measure for better insights from this confusion matrix. Courtesy: Wikipedia

An automatic verification dataset Automatically splitting the training data into validation dataset then evaluating the performance of the model on that validation dataset at each epoch. So, we would get four outputs such as cross entropy loss, accuracy, validation loss and validation accuracy at every epoch. Based on both the accuracies and its difference, one can decide about stopping the training when accuracy is maximum and its difference is less. This automatic verification dataset evaluation methods has beeen realized by using python library called Keras for my experiment.

Manual k-Fold Cross Validation Training datasets has been split into k subsets. Each subset is called fold. The algorithm is trained on k-1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross validation, we end up with k different performance values then average of those values has been calculated and it would be final performance value for that model.

2.5 The Tensorflow Framework

TensorFlow is an open source software library for numerical computation using data flow graphs developed by Google. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow can be used for deep learning algorithm quite efficiently as there are lots of pre-defined deep learning libraries are available. It also provides state flow diagram for deep learning models so that anyone can visualize it in better way. Tensorflow supports multiple programming languages such as Python, Java, C++, R, etc. CuDNN, which is GPU-accelerated library primitives for deep neural networks also supports tensorflow. Keras is high-level library which run on top of Tensorflow. It provides more intuitive programming style for deep learning model.

2.6 Deep Learning in Computer Vision: CIFAR10 Images Data

Deep learning model, AlexNet had popularized CNN in the field of computer vision when it became winner in ILCVRC 2012 with only around 16 percent error rate. The second runner up have almost 26 percent error rate. Now there are also deep learning models which surpasses the human level accuracy[4]. In the past years at ILCVRC competition, there is further improvement in the accuracy on CIFAR10 image data using modified convolutional networks by researchers. Those convolutional networks have been used for different applications of computer vision like image segmentation, annotating image with text or learning to play game just by analyzing screen pixel.

2.7 Architectures Solving CIFAR10 and Their Accuracies

There are various CNN architectures had been applied on CIFAR10 datasets. There are few most important architectures are discussed below. The first CNN architectures, AlexNet(2012) which become widely popular in getting significantly better accuracy on identifying pictures using CIFAR10 than any other machine learning algorithm.

ImageNet Classification with Deep Convolutional Neural Networks, 2012 AlexNet was one of the first CNN architecture which gained attention by its comparatively good accuracy results on image data. This architecture was named by Alex Krizhevsky whose group was winner of ILCVRC 2012. Architecture consists of five convolutional layers and three fully connected layers. Dropout had been used in first two fully-connected layers. ReLU and max pooling used in convolutional layers. After data augmentation in CIFAR10 image data, AlexNet model has achieved 89 percent accuracy[12].

Spatially-sparse Convolutional Neural Networks, 2014 CNN architecture performed significantly well when it processes spatially-sparse inputs with some modification in its architecture. A character drawn with a one-pixel wide pen on a high-resolution grid looks like a sparse matrix. If the picture is not sparse, then it can be made sparse by adding padding. Slow max-pooling retains more spatial information; this is better for handwriting recognition, as handwriting is highly structured. So applying pooling slowly, using 2*2 pooling rather than using small number of 3*3 or 4*4 pooling with small convolutional filters, performed better than later. Architecture have alternating convolutional layers with pooling layers. Architecture can be denoted by DeepCNet(l, k) where $l + 1$ is number of convolutional layer ,separated by l layers of pooling 2*2 pooling layers and nk is number of convolutional filters in nth convolutional layer. Last convolutional layers is fully-connected output layer with SoftMax activation function. If the input of size N*N , then N is equal to 3*2 to the power l. For example, DeepCNet(5,200) is architecture from input layer size N = 96 with 5 convolutional filter layers and 5 pooling layers. The spatial size of first convolutional filter is 3*3 , and then 2*2 for subsequent layers. The lower levels of a DeepCNet seem to be fairly immune to overfitting, but applying dropout in the upper levels improved test performance for larger values of k. Apply a deep convolutional neural network with sparsity has resulted 6.28 percent error rate in CIFAR10 small picture dataset[13].

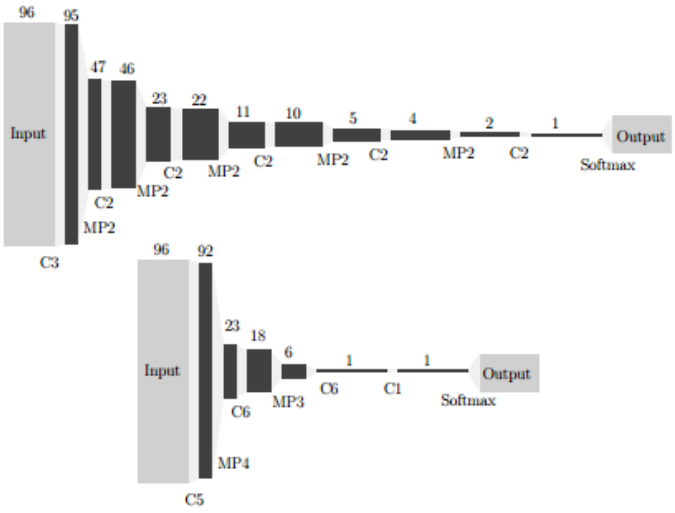


Figure 2.32: Model : input-100C3-MP2-200C2-MP2-300C2-MP2-400C2-MP2-500C2-output. The hidden layer dimensions for l = 5 DeepCNets, and LeNet-7. In both cases, the spatial sizes decrease from 96 down to 1. The DeepCNet applies max-pooling much more slowly than LeNet-7[13].

STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET, 2015 Most CNNs architecture pipeline consists of convolutional layers, pooling layers and fully connected layers. All convolutional net architecture, convolutional layer replaced the pooling layers, dropout layer as well as fully connected layers without loss of accuracy. Architecture solely built on convolutional layers, with occasionally dimensionality reduction by using stride of 2 and yields competitive performance on image recognition datasets. This architecture trained vanilla gradient descent with momentum reaches state of the art performance without the need for complicated activation functions, any response normal-

ization or max-pooling. Architecture pipeline consists of 5 3∗3 convolutional layers and at 3rd and 5th position, there is 3∗3 convolutional layer having strides of 2. Usually at those position, it is common to have pooling layers but in All-Conv-C , is replaced by convolutional layer with strides of 2 so that next layer covers the same spatial region as before. All-CNN-C architecture had achieved error rate of 9.08 percent without augmentation and accuracy of 95.59 percent with image augmentation in CIFAR10 dataset[5].

Densely Connected Convolutional Networks, 2016 Densely connected convolutional networks (DenseNets), which connects each layer to every other layer in feed forward fashion. Conventional convolutional networks with l layers have l connections. DenseNets with l layers have $(l+1)/2$ connections. For every layer, feature maps of all preceding layers are used as inputs, and its own feature maps are used as inputs into all subsequent layer. DenseNets alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. DenseNets performed significantly better than most of convolutional networks, whilst requiring less memory and computation to achieve high performance. DenseNets(k=24) have achieved error rate of 3.74 percent on CIFAR10 with data augmentation[14].

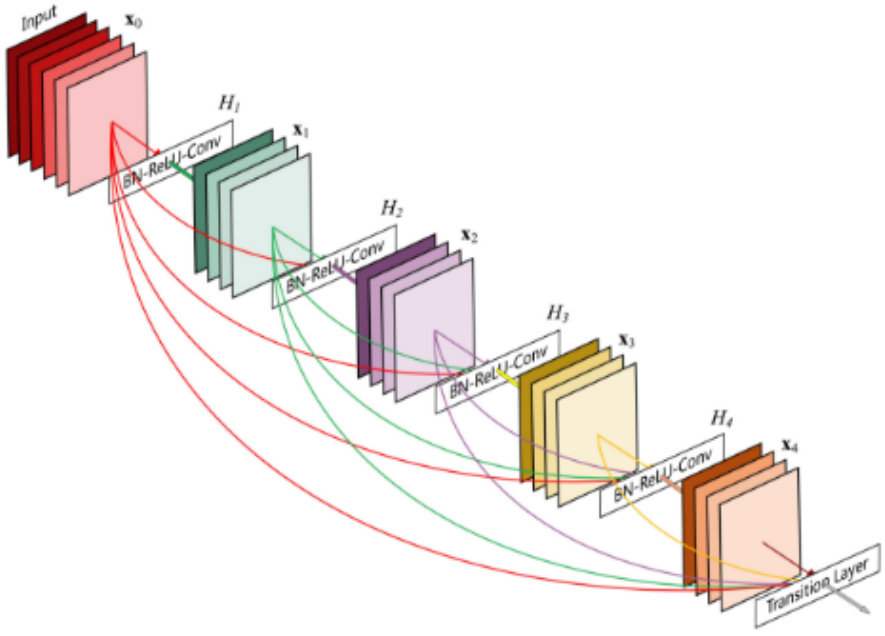


Figure 1: A dense block with 5 layers and growth rate 4.

Figure 2.33: DenseNet Architecture

Chapter 3

Comparing different CNN Architectures

3.1 Description of CNN Architecture

There are many CNN architecture in recent times which performs quite well in the field of image recognition such as AlexNet, VGGNet, GoogleLeNet, DenseNets. In these architecture, there are several different regularization techniques such as Dropout, Batch normalization, Data augmentation, which has been introduced to prevent overfitting in the model. For non-linearity in the model, rectifiers like ReLU have been employed. It also reduces overfitting in the model.

In this experiment, I try to apply some techniques and methods of different architectures to find out their effectiveness on the model for CIFAR10 data. There are four CNN architecture for solving CIFAR10 in this paper. Each CNN architecture is slightly different from others in terms of positions of dropout layers and number of convolutional layers in the model.

I have been able to reproduce the effect on model from a few research papers[4][7][8] that dropout reduce overfitting and increase the number of convolutional layers to enhance model accuracy. I have applied multiple network topologies in CNN architecture to see if it affects the accuracy of experiment. Besides, there have been use of different regularization strategies like ReLU, Pooling, Dropout layers for better accuracy. I have used VGGNet as base model and then, performed some changes in it to improve accuracy with limited computational resources and relatively less runtime. Number of convolutional layers has been reduced and more dropout layers are added in the architecture. So, experiments have been performed on four different CNN architectures similar to VGGNet on CIFAR10 datasets. All the four architectures have distinct network topologies with respect to positions of various layers and number of various distinct layers in CNN architecture.

I train three models(namely A, B and C) on CIFAR10 dataset described below in the figure 3.2, 3.3 and 3.4 in the page 31. The fourth model D(3.1) is replica of CNN model provided in tutorial by tensorflow. All other three models are modified version of this architecture. I tried to reproduce or improve the results produced by this model. All architecture are based on VGGNet with modifications on network topology and adding of

dropout layers. Input size of CIFAR10 data is $32 \times 32 \times 3$. Number of parameters in model does not depend on input size. It depends only on depth, kernel size and depth of previous layer for CNN models. Details about CNN model such as dimension and number of parameters have been provided in the followed table 3.1. There are 16 convolutional layers, which are being used in actual VGGNet model. As it is very computationally expensive to use 16 convolutional layers, so I have reduced the number of convolutional layers in the architecture. Dropout layer also has been added which is not present in VGGNet. Dropout reduces overfitting and improve accuracy on test data set. It is also inexpensive methods for regularization. Model A has three convolutional layers and two dropout layers. Model B and C have four convolutional layers each. Detailed network topologies of these models have been described in the below figure on page 31. The positions of dropout, convolutional and pooling layers are different in the three architectures. They are changed in order to see whether it would impact in the accuracy of the model. The main aim of building different architectures is to realize if any models would be able to get better results than model D. And how different layers affect the accuracy of model? Which layers would be effective to reduce overfitting?

Table 3.1: Architecture details of CNN models for training CIFAR10

Models	CNN Architectures (Network Topology)	Number of Parameters
A	**CONV2D-[32x32x64]-Pooling** **CONV2D-[16x16x64]-Pooling-Dropout(0.5)** **CONV2D–[8x8x128]-Dropout(0.5)** **FC-[1x1x384]** **FC-[1x1x192]** **Softmax**	**(3*3*3)*64 = 1,728** **(3*3*64)*64 = 36,864** **(3*3*64)*128 = 73,728** **(8*8*128)*384 = 31,45,728** **(192*192) = 36864** **Total $\sim$ 3.2M**
B	**CONV2D-[32x32x64]-Pooling** **CONV2D-[16x16x64]-Pooling** **CONV2D-[8x8x64]-Pooling** **CONV2D-[4x4x128]-Pooling** **FC-[1x1x384]** **FC-[1x1x192]-Dropout(0.5)** **Softmax**	**(3*3*3)*64 = 1,728** **(3*3*64)*64 = 36,864** **(3*3*64)*128 = 73,728** **(3*3*128)*128 = 147,456** **(2*2*128)*384 = 786432** **(192*192) = 36864** **Total $\sim$ 1M**
C	**CONV2D-[32x32x64]** **CONV2D-[32x32x64]-Dropout(0.3)-Pooling** **CONV2D-[16x16x64]** **CONV2D-[16x16x128]-Pooling** **FC-[1x1x384]-Dropout(0.5)** **FC-[1x1x192]** **Softmax**	**(3*3*3)*64 = 1,728** **(3*3*64)*64 = 36,864** **(3*3*64)*128 = 73,728** **(3*3*128)*128 = 147,456** **(8*8*128)*384 = 31,45,728** **(192*192) = 36864** **Total $\sim$ 3.2M**
D	**CONV2D-[32x32x64]-Pooling** **CONV2D-[16x16x64]-Pooling** **FC-[1x1x384]** **FC-[1x1x192]** **Softmax**	**(3*3*3)*64 = 1,728** **(3*3*64)*64 = 36,864** **(8*8*128)*384 = 31,45,728** **(192*192) = 36864** **Total $\sim$ 3.2M**

All the four architectures have pictorial representation in the below figure. These pictures show how different layers are stacked among each other in every architectures. All these architecture have been tested on CIFAR10 data. Rectifiers, Pooling, Convolutional and dropout layers are used in different fashions in model A, B and C.

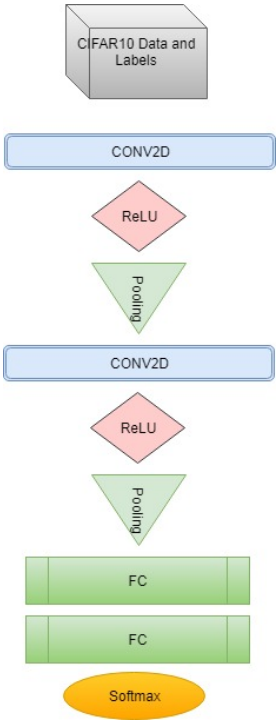


Figure 3.1: D: CNN model by tutorial of tensorflow on CIFAR10[14]

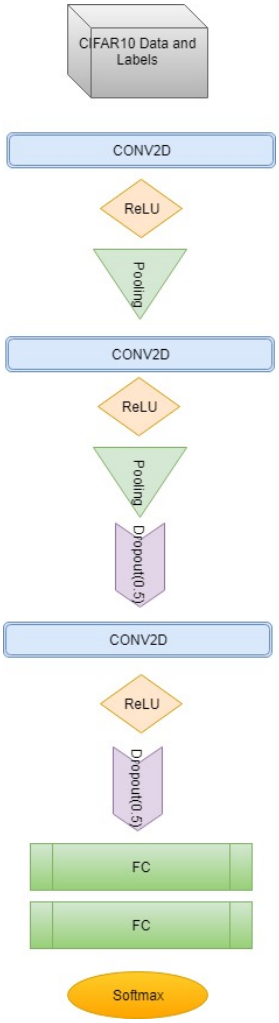


Figure 3.2: A: 1st Architecture

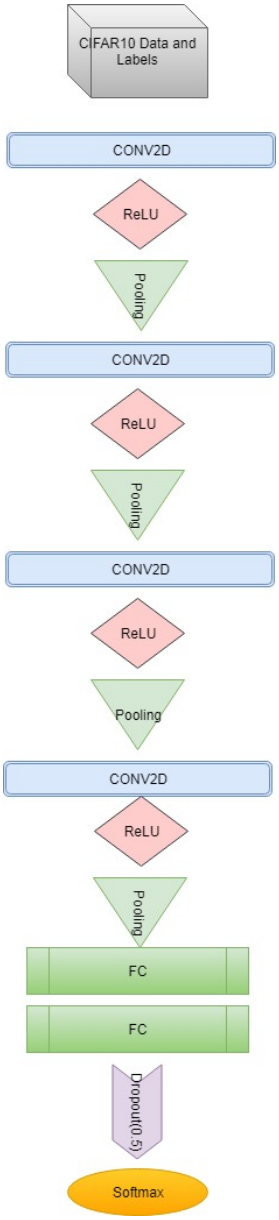


Figure 3.3: B: 2nd Architecture

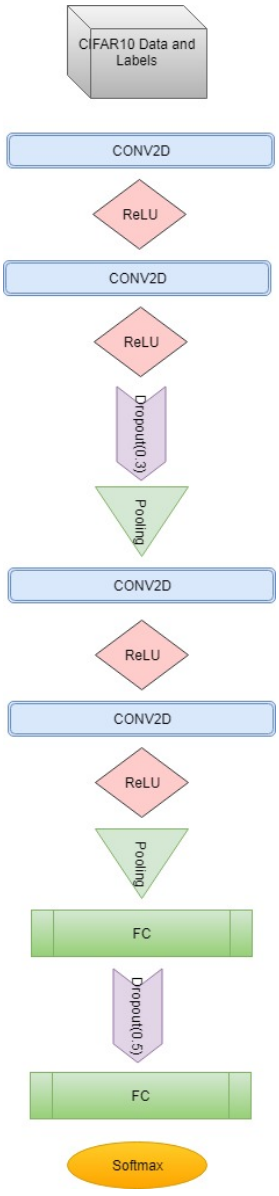


Figure 3.4: C: 3rd Architecture

3.2 Setting of the Deep Learning Environment

To get started for Deep learning research, one should first set up required environment for deep learning projects. I have set up the environment on Ubuntu Server 16.04 LTS as its operating system, and NVIDIA GRID K520 with memory size of 8 GB as its Graphical processing unit(GPU). GPU has needed because neural networks require lots of memory and computation power to perform operations such as matrix multiplication. GPU usually

has thousands of cores while a CPU usually have no more than 12 cores. Therefore, GPU facilitates parallel processing of these computations and provides required memory power using their cores effectively than CPU.

There are a few software packages, libraries and frameworks which are necessary for deep learning experiment such as Python, CUDA, cuDNN, Tensorflow and Keras. Python 2.7 used as programming language and CUDA is NVIDIA API for programming on graphics card. One can write high performance program using CUDA language on GPU. cuDNN is a library for deep neural networks using CUDA. It provides high performance GPU acceleration and highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. Tensorflow is a deep learning framework by Google which use abstraction built on cuDNN , and it also accelerates tensorflow framework. Keras is high level neural network API which runs on top of Tensorflow. It is written in Python. It provides user friendly API, and easy and fast prototyping which are possible for CNNs and RNNs using Keras.

There could be different ways to set up deep learning environment depending on programming languages, deep learning framework provided by many other vendors, operating systems, libraries, etc. Deep learning frameworks such as Caffe, Theono, Torch, Tensorflow, Keras, Deeplearning4j, etc. could be used for deep learning programming. Each framework does support multiple programming languages such Python, Java, C++, Scala, GO, R, MATLAB. etc. Programming languages could be used if that particular language is supported by the framework. Keras is a special deep learning framework which works under theono or tensorflow as it backend. Keras supports only python as of now. MATLAB is also quite handy for deep learning projects, especially for computer vision tasks. It is a powerful tool for image processing related operations, such as shortening pixel sizes, image resizing of pictures, image augmentation, etc.

In this experiment, for simplicity and to avoid conflicts with other libraries installed in the system, it is neccessary to use ”virtualenv” to create python environment and set up CUDA, cuDNN, Tensorflow, Keras, and IPython Notebook inside it. Virtualenv is a tool to create isolated python environment. Virtualenv creates a folder which contains all the necessary executables to use packages which python project would need.

3.3 Configuring and Installing in AWS Cloud

As it is quite easy and economical to configure and install deep learning environment in cloud infrastructure. So, this project utilizes AWS Cloud. It provides flexibility in selecting GPU and running it as per user'demand. Therefore, instead of buying some GPU hardware for deep learning projects, one can rent it on AWS Cloud whenever there is a need of GPU. It is one of most inexpensive and efficient ways to perform small deep learning experiments.

To configure and set up deep learning environment, the first step is to create new EC2 GPU instance, one needs to select the OS image and GPU type. For this project, I choose OS image Ubuntu Server 16.04 LTS and g2.2*large as its GPU type which contains NVIDIA K520 GPU with memory size of 8 GB. After launch of EC2 instance, CUDA, cuDNN, Tensorflow and Keras need to be installed on that EC2 instance. While

installing above software such as tensorflow and keras, one need to aware about versions. This is quite new frameworks and libraries so its version keep on changing frequently. Old code from previous software versions sometimes throw error if its run on newer version softwares. Tensorflow libary has been used for CIFAR10 training but I have used Keras for "RPSG16" data. Keras run on top tensorflow. Keras provides easier way to efficiently program for deep learning model required for deep learning projects. Below are figures of starting a machine in AWS. In the figure 3.8, there is a need to increase the storage capacity of root in OS as CUDA and CuDDN requires more than 8 GB of space.

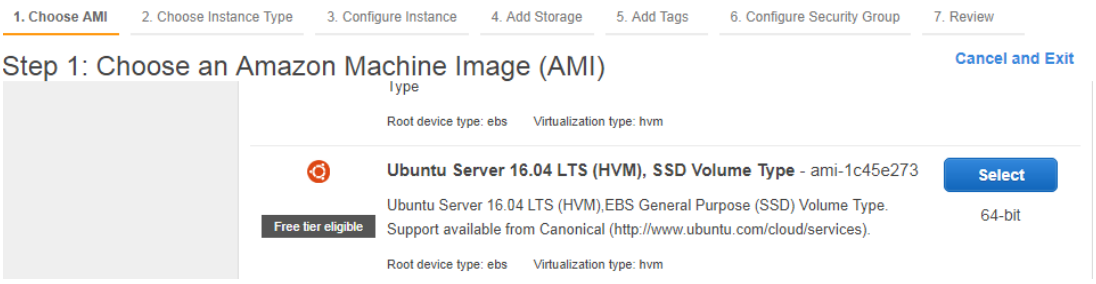


Figure 3.5: First step to choose the OS

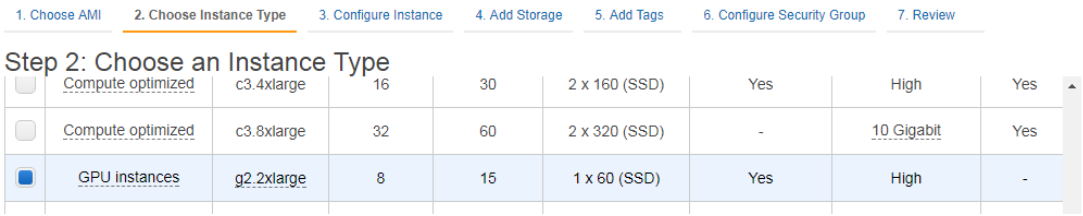


Figure 3.6: Choosing GPU in AWS

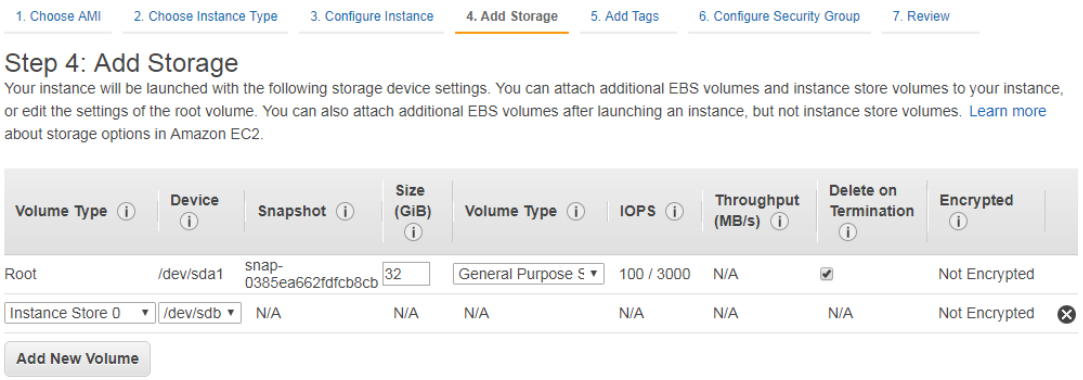


Figure 3.7: Storage of root in OS

3.4 Analyzing outputs for Layover of ConvNets

After the training, we can visualize outputs at every layer. When a image of a dog is feed into CNN then first convolutional layer output 64 different images shown below. It learns edges at first layer. In next layers, there is another 64 different images generated but it is

difficult to understand what exactly model tries to learn. All 64 images in 2nd layer have random black and white spots.

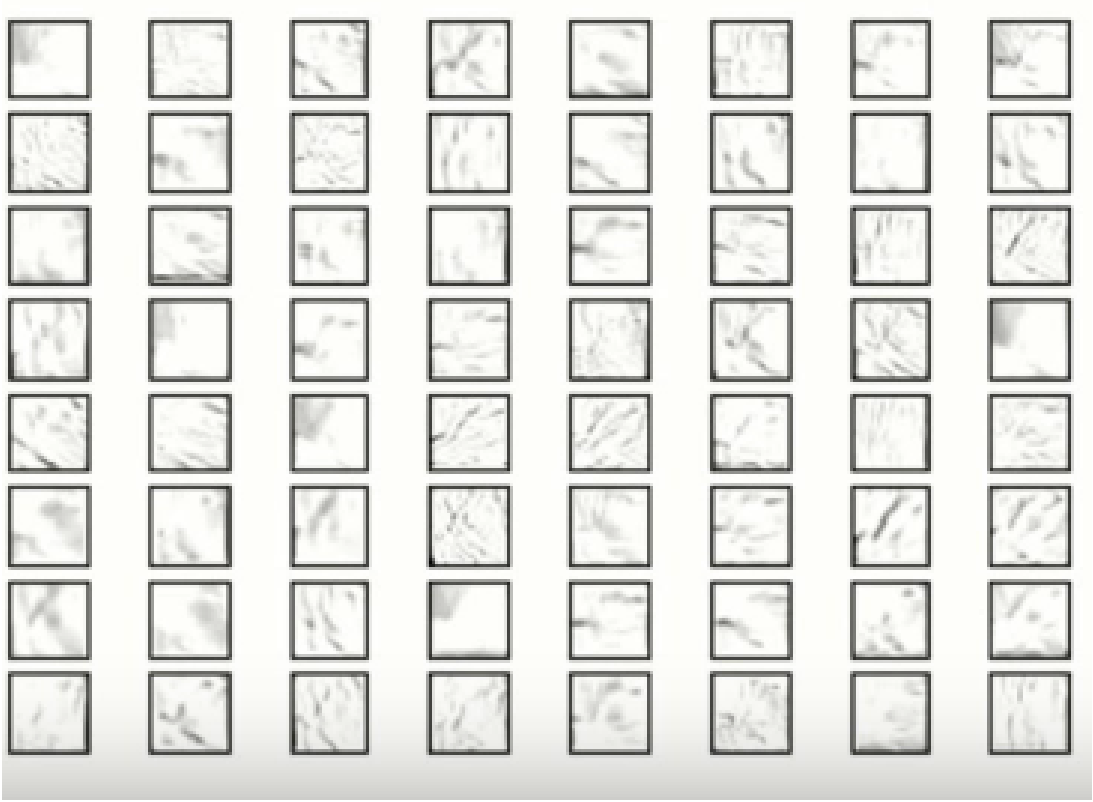


Figure 3.8: 64 images generated from 1st convolutional layer

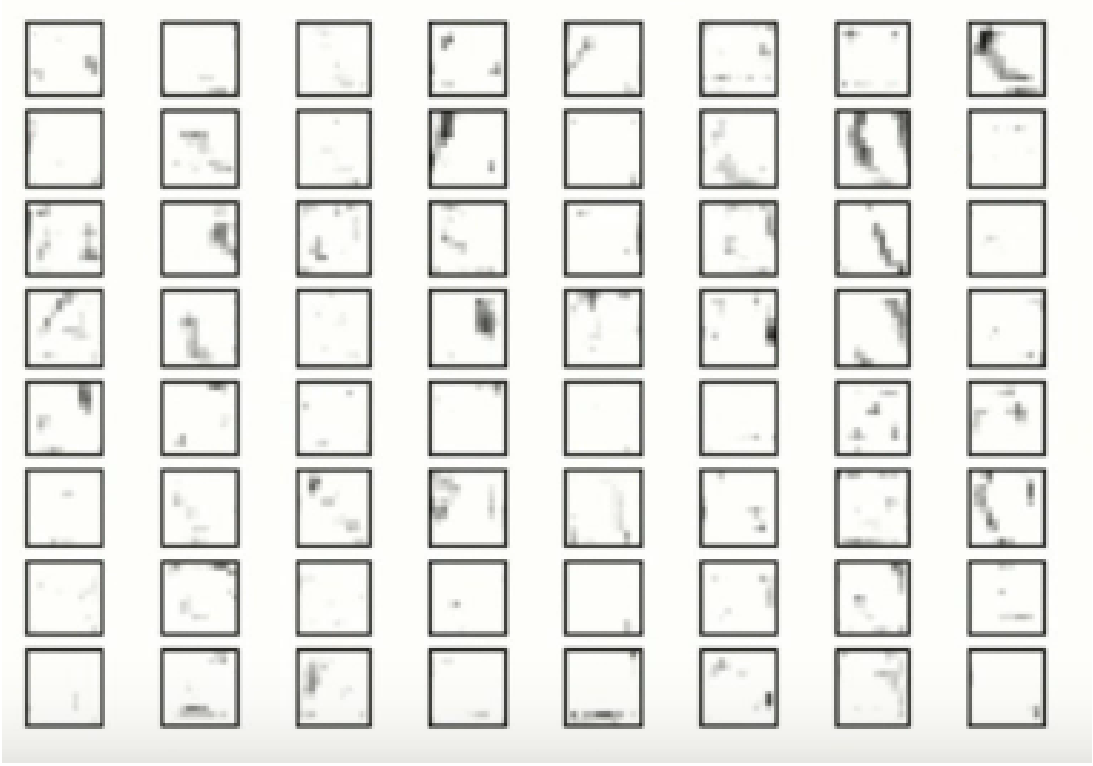


Figure 3.9: 64 images generated from 2nd convolutional layer

3.5 Results on CIFAR10

In the below table 3.3, there are results for all three models of the experiment. I have done cross-validation in this case as I just have to compare my results with results from tensorflow's tutorial and they have used same test set as what I used for training all three models. If the number of convolutional layers increased by in model A compared to model D and two dropout layers added, then it produces less accuracy than model D. It is obvious that excessive use dropout does not lead to improve test error even we increase convolutional layers. Model C performed 0.5% better than model D. It has used 4 convolutional layers with two dropout layers placed at the 3rd and 8th position in architecture. We also could see that the use of pooling layers after at every convolutional layer and dropout layer at the end just before softmax in the model B. The accuracy of model B is less than model C but more than model A. Therefore, by looking at these results of empirical study, one can infer that increasing the number of convolutional layers and adding of dropout layer at proper place significantly impact the accuracy. However, how exactly these layer positions affects the accuracy, is not clear from this empirical study. It is reaffirmed[5] from these results that dropout layer do enhance the accuracy on test data as it is one of the powerful regularization technique for CNN models.

Table 3.2: Training details of 4 models

Models	**Runtime details**	**Resource**
A	**7 hours - 60k steps (128 batch size)**	**NVIDIA K520 8 GB**
B	**10 hours - 60k steps (128 batch size)**	**NVIDIA K520 8 GB**
C	**12 hours - 60k steps (128 batch size)**	**NVIDIA K520 8 GB**
D	**5 hours - 60k steps (128 batch size)**	**1 Tesla K20m**

Table 3.3: Results of Training using four models on CIFAR10 data

Models	CNN Architectures (Network Topology)	Accuracy
A	**CONV2D-64-ReLU-Pooling** **CONV2D-64-ReLU-Pooling-Dropout(0.5)** **CONV2D–128-ReLU-Dropout(0.5)** **FC-384** **FC-192** **Softmax**	**83.1%**
B	**CONV2D-64-ReLU-Pooling** **CONV2D-64-ReLU-Pooling** **CONV2D-64-ReLU-Pooling** **CONV2D-128-ReLU-Pooling** **FC-384** **FC-192-Dropout(0.5)** **Softmax**	**85.2%**
C	**CONV2D-64-ReLU** **CONV2D-64-ReLU-Dropout(0.3)-Pooling** **CONV2D-64-ReLU** **CONV2D-128-ReLU-Pooling** **FC-384-Dropout(0.5)** **FC-192** **Softmax**	**86.5%**
D	**CONV2D-64-ReLU-Pooling** **CONV2D-64-ReLU-Pooling** **FC-384** **FC-192** **Softmax**	**86.1%**

Chapter 4

A Deep Convolutional Neural Network for Hand Posture Classification

There is a huge deep CNN effectiveness in the field of computer vision due to recent state of the art performance in the image recognition using deep learning algorithms. It is now become virtually de facto to apply deep learning in image recognition tasks. There are many variants of CNN played important role in increasing accuracy for image classification tasks. I have used CNN model similar to VGGNet architecture. Model has dropout layer in its network which is not present in VGGNet. Also, Model has less number of convolutional layers than VGGNet.

4.1 Playing Rock-Paper-Scissors with an Artificial Agents

There is an artificial agent named NAO at university. It has been programmed to play Rock-Paper-Scissors game with humans. There are three hand postures used in the game. So, NAO must identify correct hand postures of opponent. For identifying hand postures, NAO need to use some kind of machine learning algorithm. As human hand postures are different in colors, sizes or posture orientations based on different people and bakground color of hand postures. Therefore, it is quite challenging for conventional machine learning algorithm to be able to accurately classify them.

Deep learning algorithm like CNN is good in feature extraction therefore it is able to perform well for classifying the images with different colors, sizes or orientations. Due to these reasons, I use CNN in classifying different hand postures of game. We have collected images with the help of university staff members and students. We have taken a number of photos of people's hand postures(three postures of game) in three different background using each of two cameras. NAO'camera is one of two cameras used for taking photos. Other one is mobile phone cameras. We took photos from almost 35 university students and staff members.

4.2 Creating Training Data for Rock-Paper-Scissors Hand Postures

After the collecting images from different cameras, we get almost 800 images of all the three hand postures. Every hand postures (three hand images of rock-paper-scissors game) of each person is taken by two cameras in three different orientation having three distinct background.

There are 3 distinct orientations for every postures. There are also 3 different background for all the images taken from mobile camera. Only one background is used while taking picture from NAO'camera.

Image count from Mobile'camera: 3 (Hand postures) x 3 (different orientation of same hand postures) x 3 (distinct background) x 38 (total number of students and staff members) $\approx$ 1000 images

Image count from NAO'camera: 3 (hand postures) x 38 (total number of students and staff members) $\approx$ 350 images

Total image count: $\approx$ 1350 images

Total count of all the images from people whose image are taken in the experiment, is around 1300 as shown in the above mathematical calculation.So, there are total 12 images of every hand postures of single person. Every image has arbitrary pixel values and size as they are taken with different angle and distance. Total images then split into three sets of training, validation and test in 80:20:20 ratio. It is standard proportion used for machine learning experiments. Each set further split into three subsets of Rock, Paper and Scissors. It would act as label for images. Three hand postures are kept into their corresponding folders of label named Rock, Paper and Scissors. I named data as **"RPSG16"**.

4.2.1 Image Data Preprocessing

Data preprocessing is one of the most important parts for almost any deep learning experiment. Image data preprocessing includes converting these images into gray scale, pixel shortening, normalization, etc.

Images have been taken from arbitrary distance and direction. Therefore, pixel values and size of images are random to each other. There are images whose pixel size is more than 1200*1200. It is very computationally expensive to perform matrix multiplication in CNN model. It is also quite inefficient for deep learning model to work on these images, which have variety of pixel sizes. So, In this experiment, I rescale all the images to 128*128 pixel size. Then, it needs to be changed into gray scale images so that input matrix for CNN would consists of single channel (128*128*1). Using single channel in input matrix makes model less computationally intensive hence significantly improves the training speed. Color makes learning complex and difficult for features extraction. Most of the applications in image processing, color information would not help to learn important features. Due to all these reasons, I gray scale all the images and shorten the pixel values to 128*128. After that, I have programmed in MATLAB, which is one of the

most powerful image preprocessing tools for shortening pixels and converting images to gray scale.

After all images are resized and gray scaled, they all have pixel values between 0 to 255. If these values feed into model, then it may lead to numerical overflows. And multiplication of weights with pixel values force neuron to saturate. So, we should apply some techniques to overcome this problem. There is per-channel normalization technique, first zero center the data, then normalized them. It is described below.

Per-channel normalization

$X- = np.mean(X, axis = \theta)$;

$X/ = np.std(X, axis = \theta)$

However, I have used simple rescaling techniques for compressing pixel values between 0 and 1(Rescale $\bar{1}./255$). This is implemented by Keras which is a high level python library.

4.2.2 Image Data Augmentation

Images data which we gathered for the experiment are not sufficient for deep learning training. Deep learning models do require significant amount of data so that all its kernel weights would be fully optimized. If there is more data available for training, then there are less possibilities of overfitting. Hence, it would eventually boost performance. Data augmentation could also be inexpensive and powerful method of regularization in deep learning model. However, data augmentation is not enough to prevent overfitting since generated samples are still highly correlated.

I created artificial/fake image samples from existing images by a number of random transformations. New Image data has been randomly generated by rotating, horizontal flipping, shifting width and height of existing image data. These parameters have been applied to all images randomly and created 3 new images from each image. Therefore, the total images are almost 9000. These techniques provide entirely new sample for inputs. Due to rotation and shift, there is fill mode strategy applied to fill in newly created pixels. These generated images are mot highly correlated from each other as they have different orientation, width, height, etc. All images looks as distinct as inputs.

Keras is the library which I used for data augmentation task. As it has already built-in functions to do transformations such as rotating, flipping, shifting weight, etc. It randomly apply various transformations together on every image at each iteration. As a result, CNN model would never see twice the same picture. This helps prevent overfitting and helps model generalize better. These are the below parameters used for random transformations. Based on these parameters, all images are generated from old ones. I have generated around 3 artificial images from each image.

Datagen = ImageDataGenerator(

rotation_range=40,

width_shift_range=0.2,

height_shift_range=0.2,

rescale=1./255,

shear_range=0.2,

zoom_range=0.2,

horizontal_flip=True,

fill_mode='nearest')

Below is details about transformations and its parameters:

- **Rotation_range**: It is a value in degrees (0-180), a range within which to randomly rotate pictures.
- **Width_shift and Height_shift**: These are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally
- **Rescale**: It is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. factor.
- **Shear_range**: It is for randomly applying shearing transformations
- **Zoom_range**: It is for randomly zooming inside pictures
- **Horizontal_flip**: It is for randomly flipping half of the images horizontally relevant when there are no assumptions of horizontal assymetry (e.g. real-world pictures).
- **Fill_mode**: It is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

We collected around 1400 images. Each of three classes have almost 420 images. Using MATLAB, I mirror all the images. So, there are around 840 images for each group. Then, I have applied above mentioned transformations to generate 3 images from every image. Finally, there are total 8 images for every original image(which taken by cameras).

Figure 4.1: Original gray images

Figure 4.2: 1st generated image

Figure 4.3: 2nd generated image

Figure 4.4: 3rd generated image

4.3 Realizations of Deep learning project

Convolutional neural network, a pillar algorithm of deep learning, are used for perceptual problems such image classifications. As there is small dataset available to perform deep learning algorithm so data have been augmented by programming. Data augmentation is not enough to prevent overfitting. The main focus to prevent overfitting should be entropic capacity of model, how much model is allowed to store. If a model stores a lot of information then it would be likely to learn more features and some of features might be irrelevant. But if it stores only a few features, then it will have to focus on the most significant features. Hence, it is more relevant and generalize better. One way to modulate entropic capacity, is the choice of parameters, i.e the number of layers and size of each layer.

Considering all the above reasons, I design a small convnet compare to advanced industrial one. Dropout layers also being used. Dropout layers in addition to data augmentation help to prevent overfitting.

Technical details

This experiment has been done by remote machine provided by AWS. I rented a machine in which all the required software frameworks and libraries are configured. CUDA, CuDNN, Tensorflow and Keras are already installed in the OS.

Operating System: Linux/Unix, Ubuntu 14.04, 64-bit Amazon Machine Image (AMI)

AWS Instance details:

1. **Instance Name** : g2.2xlarge
2. **GPU** : 1 NVIDIA GRID K520 GPU, each with 1,536 CUDA cores and 4 GB of video memory
3. **Internal Memory**: 32 GB
4. **CPU**: 32 vCPUs

Software framework and Libraries:

1. **NVIDIA CUDA and CuDNN :** CUDA is NVIDIA's API for programming on the graphics card. cuDNN is a library for deep neural nets built using CUDA. Cuda Toolkit Version 7.5 and CuDNN 5.1.
2. **Tensorflow :** Tensorflow 1.0.0
3. **Keras :** Keras 2.0.0 as deep learning library run on top of tensorflow.
4. **MATLAB :** MATLAB R2017a has been used for data preprocessing.

4.3.1 Convnet Architecture

In this experiment,The available dataset for training is smaller. So, I build relatively a small CNN architecture. Small networks have lesser entropy capacity so it learns only important features. Therefore, the chance for overfitting is lesser. It would be a good idea to build architecture with optimally calculated number of layers. The layers are 4 convolution, 2 fully connected, 1 dropout, 3 ReLU and 3 max-pooling layers. Dropout randomly removes nodes with a given probability. In this case, I have kept 0.5 as its probability. A dropout layer is placed between two fully connected layer. An excessive use of dropout layers may result a lower accuracy as the available dataset is smaller and entropy capacity is lesser for a smaller model. With every consequent layer,the number of parameters increase drastically. It has been shown in the my previous experiment with CIFAR10 data using Model C that a dropout layer would be more effective near the end of convolutional network in a smaller architecture. Considering all this, it would be efficient to keep a dropout layer between two fully connected layers. The network topology of this experiment is bit similar to Model C which was used for training CIFAR10.

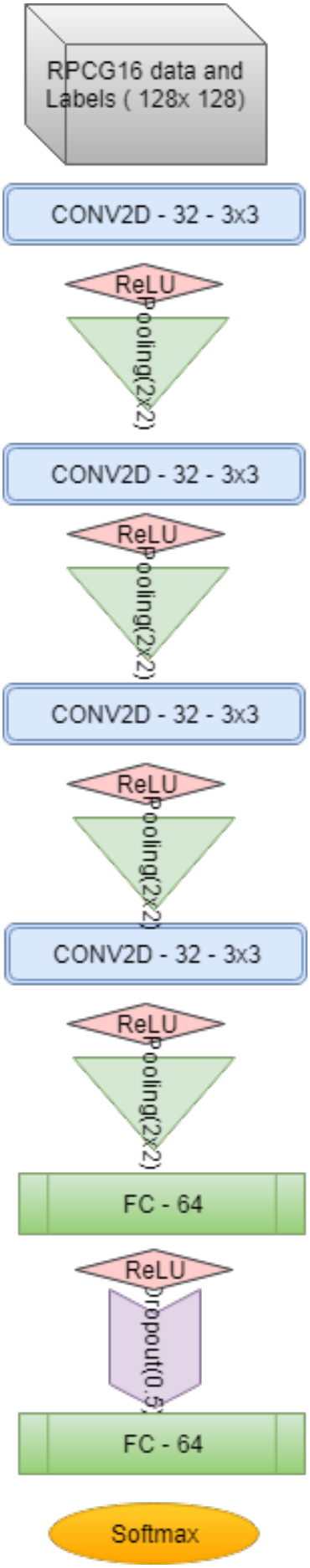


Figure 4.5: Convnet Architecture for RPSG16 data

4.3.2 Application of the CNN Architecture

There are many applications of CNN model. This model can be applied to any image classification tasks. To perform any such tasks, there is need to change few hyperparameters and image preprocessing is also required.

4.3.3 Code implementation

The libraries used for this experiment are Tensorow, Keras, SciPy, PIL, etc. The programming language used is python. For image preprocessing tasks, MATLAB is also been used as programming tool. MATLAB is used for pixel shortening, mirroring image and converting image to gray scale. For data augmentation, keras library in python has been used. Model is programmed in python using keras library. Tensorow run in backend of keras. So, Keras exploits all the supports from tensorow such GPU acceleration. Some of the code snippet are shown below.

Network consists of 4 convolutional layers. Hypermeters such as pool size, kernel size are same for all layers but depth size is different among layers. Two of convolutional layers have 32 and other two have 64 as its depth size. Max-pooling and rectifier layer have been used just after every convolutional layer.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape, dim_ordering='tf'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Two fully connected layer has been placed at the end of model. A fully connected layer is a kind of convolutional layer with 1convolutional kernel as per Yann LeCun. A dropout has been placed between two fully connected layers. As the dataset is small for deep learning experiment even after data augmentation,the number of parameters increase exponentially from input to output. It is a good idea to keep near the end as lesser parameters gets dropped from the model. So, it would impact comparatively lesser parameters if we put it near the end but we need the dropout because it would help to prevent overheating. In this model, I have used probability 0.5 for dropout layer. Below are code snippets.

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(3))
model.add(Activation('softmax'))
```

4.3.4 Training: Epochs and Early Stopping

Validation set has been used to find accuracy and loss at every epoch while training. It also helps to decide, when I can stop the training based on error on validation set. If validation error is increasing, then it is a sign of overfitting. Also, when gap between test accuracy and training accuracy starts to increase(Figure: 2.25), then that is also sign of overfitting and there is need to stop training and its called early stopping(as discussed in 2nd chapter). So, training has been stopped after 50 epochs as training accuracy have started to decrease gradually after epochs.

A validation set has been used to track accuracy and loss at every epoch while training. It also helps to decide for when can i stop the training based on error in a validation set. If validation error is increasing, then it is a sign of overheatting. Also, when gap between test accuracy and training accuracy starts to increase(Figure: 2.25), then that is also a sign of overheatting and there is a need to stop the training and it's called early stopping(as discussed in 2nd chapter). So, the training has been stopped after 50 epochs as the training accuracy started to decrease gradually after it.

There are five training and test data sets for a 5-fold cross-validation. There is no intersection of image data between training and test sets used for a particular training. There are around 2400 images for each classes in training set and 300 images for each classes of a test set. There is only one validation set used for all five training. Validation set consists of around 400 images for each classes(Rock, Paper and Scissors). So, total training set contains 7277 images, validation set contains 1301 images and test set have 836 images for every round of 5 trainings. I have set up batch size as 32. It means that the algorithm takes 32 samples of a training dataset and trains the network. Then, it takes another 32 samples and trains the network again. The Optimization algorithm is RMSProp for this training and a softmax funtion is placed at the end as classifer. It took around 20 minutes to train model until 50 epochs on a EC2 machine(AWS).

4.4 Results and Evaluation

Initially, there was 3 convolutional layers but results were not satisfactory. Accuracy was around 76% from that model after 5-fold cross-validation. Then, I have increased one more convolutional layer in network and results significantly improve to around 82% test accuracy. Below is the table of results obtained from 5-fold cross-validation.. Exact test accuracy after cross-validation is **81.53%**(figure: 4.1).

Table 4.1: Test Accuracy from 5-fold cross-validation

Training/Test sets	**Test Accuracy**
1st Training	84.61%
2nd Training	85.68%
3rd Training	75.76%
4th Training	77.48%
5th Training	84.14%
Total	**81.53%**

As we can see in the below table(4.2), the gap between training accuracy and validation accuracy is quite low. This infers that there is very less overfitting in the model(Refer Graph 2.25). Also, the difference between validation and test accuracy is around 10%. It is common for scenario when there is less data availability and model is relatively small.

Table 4.2: Training and Validation Accuracy : 5-fold cross-validation

Training/Test sets	**Training Accuracy (After 50th Epoch)**	**Validation Accuracy (After 50th Epoch)**
1st Training	90.49%	94.41%
2nd Training	91.03%	91.17%
3rd Training	90.14%	89.20%
4th Training	90.05%	89.28%
5th Training	90.60%	97.01%
Total	**90.46%**	**92.21%**

Below are screenshots of results for first training set. It taken from EC2 instance terminal:

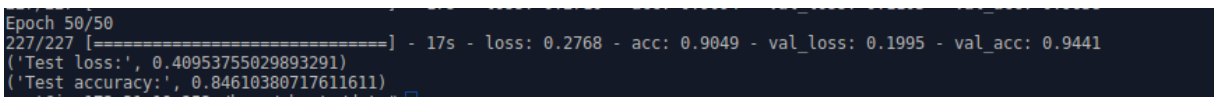


Figure 4.6: 1st result : test accuracy

Chapter 5

Conclusion and Future work

In this thesis, I have covered an overview of deep learning and discussed methods and techniques for convolutional neural networks. Those techniques and methods like Batch normalization, Dropout, ReLU, Pooling are applied on CIFAR10 data to see its effectiveness on performance. Also, multiple deep learning model has been trained with corresponding methods and techniques. The architecture which performs well compared among models (with techniques and methods) is selected for my final experiment for Rock-Paper-Scissors. In addition, the model has been trained on the data after data preprocessing and data augmentation. The achieved accuracy rate was around 82 percent on data and this figure can further be improved by increasing data and using more convolutional layers in the architecture. It is also noticeable that training accuracy gradually decrease after certain epochs. In this case, one should use strategy called Early Stopping. Both Early Stopping and Dropout are inexpensive way to prevent overfitting in the model.

One can collect some more images from different people. Fake data generated from real ones are still correlated to each other and this may lead to overfitting. Hence, collecting more images and feeding it into large CNN model do enhance the test accuracy significantly. We have a robot named NAO which can plays "Rock-Paper-Scissors" game with human. As we now have deep learning algorithm which can identify the hand postures of game with more than 80 percent accuracy, we can incorporate this algorithm into NAO in the future to see the change in performance of robot when playing the game with different people.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[2] Andrej Karpathy's course: `http://cs231n.stanford.edu/`.

[3] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. *Deep Learning REVIEW*. NATURE, MAY 2015.

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

[5] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox and Martin A. Riedmiller. *Striving for Simplicity: The All Convolutional Net*. abs/1412.6806, 2014.

[6] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. abs/1502.03167, 2015.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. JMLR, 2014.

[8] Haibing Wu and Xiaodong Gu. *Max-Pooling Dropout for Regularization of Convolutional Neural Networks*. abs/1512.01400, 2015.

[9] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. abs/1311.2901, 2013.

[10] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. abs/1409.1556, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. *Going Deeper with Convolutions*. abs/1409.4842, 2014.

[12] Yoshua Bengio, Aaron C. Courville and Pascal Vincent. *Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives*. abs/1206.5538, 2012.

[13] Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks*. NIPS, 2012.

[14] Benjamin Graham. *Spatially-sparse convolutional neural networks*. abs/1409.6070, 2014.

[15] Gao Huang, Zhuang Liu and Kilian Q. Weinberger. *Densely Connected Convolutional Networks*. abs/1608.06993, 2016.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg and Fei-Fei Li *ImageNet Large Scale Visual Recognition Challenge*. abs/1409.0575, 2014.

[17] Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner. *Gradient-based Learning Applied to Document Recognition*. PROC. of the IEEE, 1998

[18] Tensorflow's tutorial: `https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10`

[19] Francois Chollet's documentation: `https://github.com/fchollet/keras-resources`

Appendix A

Content of the DVD

The DVD contains a zip file with the following content:

1. a folder that contains all the original images(RPSG16) taken from two cameras.
2. a folder that contains all the 5 training sets and test sets used(RPSG16 preprocessed data).
3. a file which have all technical detail to run and train the models.
4. a another folder that contains all files for the training, including the .py files for CIFAR10 and RPSG16 training.
5. the "ReadMe" file with some additional information that one might notice.
6. the Master's thesis in PDF format.

Ich erkläre hiermit gemäß §17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bamberg, den 14.06.2017 _____