Implementation of Convolution Neural Network on the CIFAR-10 Dataset

Name: Arjun Chakraborty

ECE 8903

Project Report

Dr. Marilyn Wolf

Acknowledgement

I would like to express my heartfelt thanks and gratitude to Dr. Marilyn Wolf, School of Electrical and Computer Engineering, Georgia Institute of Technology for all her guidance and mentorship throughout the duration of the project. It was an enriching experience that has enhanced my academic credentials and has exposed me to new and exciting Machine Learning concepts and tools.

INDEX

1. Introduction

Computer Vision has become ubiquitous in our society, with widespread applications in image understanding, apps, drones and self-driving cars. Many of these applications are visual recognition tasks such as image classification, localization and detection. Advances in deep learning approaches have advanced the performance of these visual recognition systems. The second phase of the project aims at implementing image classification algorithms using Convolution Neural Network on the CIFAR-10 dataset. This is a multi image classification task with more than two classes. This means that one image needs to be labeled by one category from a set of categories based on analyzing the numerical properties of the image. The CIFAR-10 dataset is an ongoing research effort to provide researchers around the world an easily accessible large-scale image database. The project aims to master the deep learning model using CIFAR-10, since it is a typical multi image classification problem.

2. Dataset

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images in each class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly selected images from each class. The training batches contain images in a random order, but some training batches may contain more images from one class than the other. The classes are completely mutually exclusive. There is no overlap between each class.

3. Tools Used:

Caffe is the deep learning framework, which has been used for this project. It is a framework, which has been made with expression, speed and modularity in mind. The models and optimization are defined by configuration without hard coding. The switch between CPU and GPU can be done by setting a single flag to train on the GPU machine. The speed and implementation of Caffe makes it perfect for deployment in research and industry.

4.Timeline:

Task	Time Duration
Familiarity with Machine Learning Concepts	January 2015
Reading of Papers on Deep Learning and ULFDL Tutorial	February 2015
Implementation of basic Deep Learning	March 2015

concepts on MNIST dataset using Theano	
Study of Convolution Neural Networks in detail using Stanford CS 231 Lecture Notes	March-April 2015
Familiarity with Caffe and Implementation on CIFAR-10 dataset	April 2015

4. Theoretical Background:

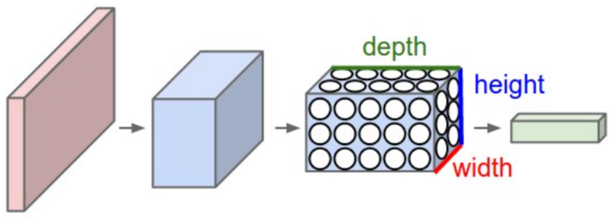
4.1 Basic Idea

Convolution Neural Networks are very similar to ordinary neural networks. They are made up of neurons that have learnable in weights and biases. Each neuron receives some inputs, performs a dot product and follows it up with a non-linearity. This leads to the whole network expressing a single differentiable score. This is from the raw image pixels at one end to the class scores at the other.

4.2 Structure of a Convolution Neural Network

Convolution Neural Networks take advantage of the fact that the input consists of images. This leads to a constraint in the architecture in the most sensible way. The convolution neural network has neurons arranged in three dimensions: width, height and depth. The CIFAR-10 dataset, for example has an input volume of 32x32x3 (width, height and depth). The neurons in a layer will only be connected to a small layer in the region above it, instead of all the neurons in a fully connected manner. The final output layer for CIFAR-10 would have dimension 1x1x10. The full image would be reduced to a single vector of class scores, arranged along the depth.

If we use the method of parameter calculation for regular neural networks, a fully connected neuron in the first hidden layer would have 32x32x3=3072 weights. The amount seems manageable, but if an image of dimensions 200x200x3 were the input, the number of weights for each neuron of the hidden layer would be 120000.



Structure of a Convolution Neural Network

4.2 Layers used to build a Convolution Neural Network

Every layer of the Convolution Neural Network is used to transform one value of activation to another using a differentiable function. Generally, the three main types of layers used to build Convolution Neural Network architectures are: Convolutional Layer, Pooling Layer and Fully Connected Layer. An example of a structure of a Convolutional Neural Network would be:

- INPUT: This layer would hold the raw pixels of the image, in this case, an image of a certain width, height and the number of channels.
- CONV: The Convolutional layer would compute the output of the neurons that are connected to the local regions in the input. Each computing a dot product between their weights and the region connected to in the input volume.
- POOL: This layer will perform a downsampling along the dimensions resulting in a smaller volume.
- FULLY CONNECTD LAYER: The fully connected layer will compute the class scores. As with ordinary neural networks, each neuron in the layer will be connected to all the numbers in the previous volume.

4.2.1 Convolutional Layer

The convolution layer's parameters consist of a set of learnable filters. Every filter consists of a set of learnable filters, but extends through the full depth of the input volume. During the forward pass, each filter is slid across the width and height of the input volume, producing a 2 dimensional input activation map of the filter. As we slide across the filter, the dot product between the input and the entries of the filter are computed. The network will learn the filters that activate when they see a specific feature at some spatial position in the input. These activation maps for all filters are stacked along the depth dimension along the full output volume.

The input to the convolutional layer is a high dimensional image. It is impractical to connect neurons to all the neurons in the previous volume. Instead, each neuron is only connected to a local region of the input volume. This is known as the receptive field of the neuron.

4.2.2 Pooling Layer

The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and to control over fitting. The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form of poling is a pooling layer with filters of size 2x2.

4.2.3 Fully Connected Layer

Neurons in a fully connected layer have full connections to all the activations in the previous layers, as seen in regular neural networks. The activations can be computed with a matrix multiplication followed by a bias offset.

4.3 Spatial Arrangement of Neurons:

This section deals with the connectivity of each neuron in the convolution layer to the input volume. Three hyper parameters control the size of the output volume:

- Depth: It controls the number of neurons in the convolutional layer that connects to the same region of the input volume. The set of neurons that are all looking at the same region of the input as the depth column.

- Stride: The space with which we allocate depth columns around the width and height. When we allocate a new depth column of neurons to spatial positions only 1 unit apart.

- Zero Padding: Zero padding is used to pad the input with zeros spatially on the border of the input volume.

5. Classification using CIFAR-10 dataset

Caffe has certain BVLC trained models for unrestricted, out of the box use. Lots of researchers have made Caffe models for different tasks with all kinds of architecture and data. These models are learned and applied for various types of visual recognition. This project discusses two kinds of architectures. The first is the *LeNet* model and the second is Alex Krizhevsky's Cuda-convnet model. They were both run for the CIFAR-10 dataset using Caffe on the Jinx cluster. Access to the Jinx cluster was given to me because of the ECE 6254 Statistical Signal Processing class.

5.1 Hardware Used

The Jinx cluster consists of 30 nodes, with 3 distinct hardware configurations:

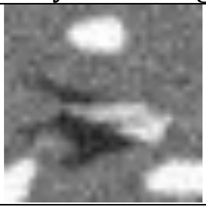
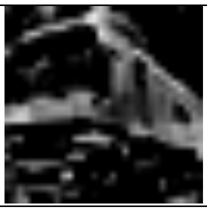
- 24 HP sl390s nodes, each with: 2 Intel Xeon X5650 6-core processors, 24GB memory, GPU accelerated, 380GB local scratch storage.

- 6 Dell PowerEdge R710 rack-mounted servers, each with: 2 Intel Xeon X5570 quad-core processors, 48GB memory, 2TB high-speed local scratch storage

5.2 Preprocessing:

- **Conversion into grayscale**: The *LeNet* model has been originally constructed for the MNIST dataset. Hence, for that model the images in the CIFAR-10 dataset were converted to grayscale using MATLAB.

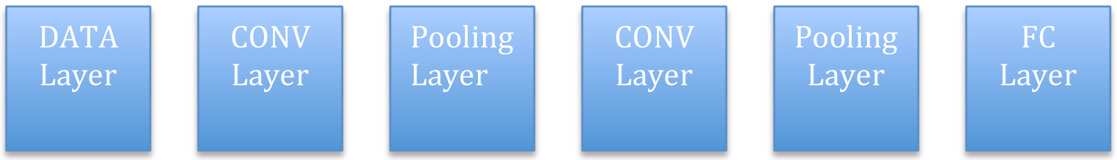
- **Whitening**: PCA is a dimensionality reduction algorithm that can be used to speed up the feature-learning algorithm. The understanding of PCA helps to implement whitening. In order for PCA to work well, the features have to have approximately zero mean and have similar variances to each other. PCA is in fact invariant to scaling of data and will return the same eigenvectors, regardless of the scaling of the input. The goal of whitening is to make the data less redundant. The features should be less correlated and have the same variance. In this project, ZCA whitening has been implemented. ZCA whitening is a form of pre-processing that maps from x to $x_{ZCAwhite.}$ It is a rough model of the retina processes images. The retina performs a decorrelation operation, which is similar to the one performed by ZCA.

Original Image	Grayscale Image	ZCA Whitened Image
		

The table above shows the original image, the grayscale image and the ZCA whitened image. As it can be seen, the grayscale image is very unclear for a 32x32 image. The LeNet network would struggle to recognize the finer features of such an image. It is hypothesized that the filters would only learn a set of edge filters.

5.3 Architectural overview:

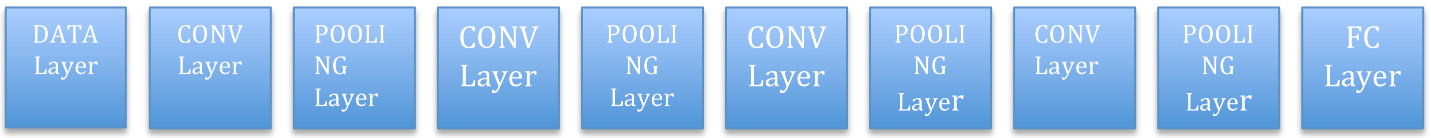
LeNET Architecture:



Parameters involved:

Layer	No of outputs	Kernel Size	Stride	Zero Padding
CONV Layer 1	20	5	1	0
Pooling Layer 1	NA	2	0	0
CONV Layer 2	50	5	1	-
Pooling Layer 2	NA	2	0	0
Inner Product 1	500	-	-	-
Inner Product 2	10	-	-	-
Softmax Loss	-	-	-	-

Cuda-Convnet Architecture



Parameters involved:

Layer	No of outputs	Kernel Size	Stride	Zero Padding
CONV Layer 1	32	5	1	2
Pooling Layer 1	-	2	3	0
CONV Layer 2	32	5	1	0
Pooling Layer 2	-	3	0	0
CONV Layer 3	64	5	1	2
Pooling Layer	-	2	0	0
Inner Product 1	64	-	-	-
Inner Product 2	10	-	-	-
Softmax Loss	-	-	-	-

6. Results

In this project, we used 2 different architectures to run the CIFAR-10 dataset. We used the pre-trained LeNet and Cuda-Convnet architectures for this purpose. The Cuda-Convnet architecture s able to process RGB images, hence no preprocessing is required. In the case of the LeNet architecture (which has been trained only for grayscale images), some preprocessing was required. Results were compared for 2 cases: the first case was for grayscale images and the other for the ZCA whitened images. There were improvements in the second case.

The Cuda-Convnet architecture was developed by Alex Krizhevsky and produces much superior results on RGB images.

6.1 LeNET Architecture:

No of Iterations: 10000
Batch Size: 100
Learning rate=0.01
Weight Decay=0.005

Input Image	Testing Accuracy
Grayscale	53.80%
Grayscale and Whitened	64.10%

There are some things that need to be kept in mind while fine-tuning a dataset on a pre-trained network. The overall learning rate for the LeNet was kept at 0.01. It was important to decrease the learning rate to a smaller number, so that the whole model would get more time to adapt to the new dataset. Also, the learning rate of the newly introduced layer should be increased, so that it is able to learn the higher features better.

In the case of only the grayscale images being used , the network only learn the edge detectors as expected and hence produced very poor results. In case of the grayscale-whitened images, the results were much better. However, more work has to be done in understanding the constraints involved in ZCA whitening, rather than treating it as a stand-alone pre-processing procedure. There is the possibility of getting better results with this architecture.

6.2 Cuda Convnet:

No of Iterations: 5000
Batch Size: 100
Learning rate=0.0001
Weight Decay=0.005

Input Image	Testing Accuracy
RGB	73.98%

The Cuda-Convnet produces excellent results with the RGB images, much superior to the LeNet network. This is because it has been fine-tuned to work with this dataset and the weights have been pre-trained accordingly. I did try running the same architecture with different filter sizes and number of feature maps, but it did not change the results very drastically.

6. Conclusion and Future work:

The second phase of the project gave me a deeper understanding of Convolutional Neural networks. It also gave me hands on experience working with Caffe, which is an extremely popular Deep Learning framework. The flexibility offered by Caffe allowed me to experiment with different networks. In case of the CIFAR-10 dataset, the Cuda Convnet architecture produced much superior results. However, the experience of implementing the same dataset and fine-tuning it on the LeNet architecture proved to be very helpful. The whole project, over the course of 6 months gave me a very solid understanding of Deep Learning concepts and Convolutional Neural Networks in particular. Over the two phases of the project, I programmed in Theano as well as Caffe. Hence, giving me a holistic experience with respect to Deep learning frameworks. I also managed to gain valuable experience running my network on the Jinx cluster.

Future work involves trying to run these networks on the Jetson TK-1 Development Kit. Caffe has already been configured to work with the Jetson TK-1 Development Kit and it would be an interesting insight into how mobile computing configures with the relatively new concept of Deep Learning architectures.

7. References:

(1) Theano Development Team. http://deeplearning.net/tutorials/contents.html

(2) A.Ng, J.Ngiam,C.Y.Foo and C Suen.,UFLDL tutorial. http://ufldl.stanford.edu

(3) CAFFE Development Team, http://caffe.berkeleyvision.org

(4) Alex Krizhevsky,Iilya Sutskever, Geoffrey Hinton, ImageNet classification with Deep Convolution Neural Networks.

(5) Yoshua Bengio,Learning Deep Architectures for AI.