# Image Processing Library

Vasu Jain Ritvik Vij

17/02/19

## 1 Introduction

The image processing library is a basic implementation of the neural net architecture LeNet with an input image as a .png file of size 28×28

The task was divided into three stages

- implementing linear algebra operations such as convolution and matrix multiplication as well as applying activation functions on vectors and matrices
- improving operations such as matrix multiplication using multithreading as well as using libraries such as openblas and intel mkl
- taking an image as input for the LeNet architecture to make a digit recognition software.

The data provided to us already trained using caffe framework,it includes the weights and the bias.The weights matrix is a 4 dimension matrix with each dimension depicting the height, width, number of filters and number of input channels respectively

The convolution implemented was of a matrix image with a volume or box in the function BoxPooling.Sub sampling has been done to apply maxpooling on the volume using the function BoxPooling.

We will be describing each subtask in detail ahead.

## 2 Subtask1

The operations involve convolution which used the functions convolution() and convolution using matrix multipliction which used the function convolutionMM() using the principles of toeplitz matrix

Activation functions implemented involved Relu and tanh for a matrix and Sigmoid and Softmax for a vector.

Arguments in these functions include the input Matrix/Vector, padding if any and stride during sub sampling of the Matrix.The input was taken as a command line input.

### 2.1 Functions used

- For convolution - vector$< vector < float >> convolution(intpad, vector < vector < float >> UnpaddedInput, vector < vector < float >>$ kernel)
- For convolution using matrix multiplication - vector$< vector < float >> convolutionMM(intpad, vector < vector < float >> UnpaddedInput, vector < vector < float >>$ kernel)
- For ReLU vector$< vector < float >> MatrixReLU(vector < vector < float >>$ input)
- For tanh vector$< vector < float >> MatrixTanh(vector < vector < float >>$ input)
- For Sigmoid - For Sigmoid vector$< float > sigmoid(vector < float >$ input)
- For Average Pooling vector$< vector < float >> AveragePooling(intpad, intPoolSize, vector < vector < float >>$ UnpaddedInput)

- For Max Pooling vector$< vector < float >> MaxPooling(intpad, intPoolSize, vector < vector < float >>$ UnpaddedInput)
- For Matrix view void printMatrix(vector$< vector < float >>$ Matrix)
- For Vector view void printVector(vector$< float >$ Vector)

## 3 Subtask 2

This subtask involved improving our Linear Algebra operations such as matrix multiplication using pthreads and commercial libraries such as multi-threading, OpenBlas and Intel MKL.

- Using pthreads - if we have a N×N Matrix, suppose we use k threads for the matrix multiplication every k rows is applied to the same pthread, a thread contains a row which can be multiplied with the other matrix. The pthread is a much faster implementation than the normal matrix multiplication. Using the toeplitz matrix, the matrix multiplication can be used to implement the convolution of a matrix with a filter.
- Using OpenBlas - OpenBlas is a linear algebra library for fast implementation of linear algebra operations, sgemm function was used to implement matrix multiplication and is considerably fast for larger sizes of matrix upto approx 800×800.
- Using Intel MKL - MKL linear algebra library is built upon the OpenBlas library but is considerably faster than OpenBlas and Pthreads,approximately 19 times faster for a matrix of size 200×200, the function used is the same sgemm(for operations on float matrices).

A GNU Plot was made to compare the time taken to perform this operation using pthreads, OpenBlas and Intel MKL.

Further usage of convolution in the LeNet architecture for the digit recognition software has been done using our very own pthread implementation.

## 4 Subtask 3

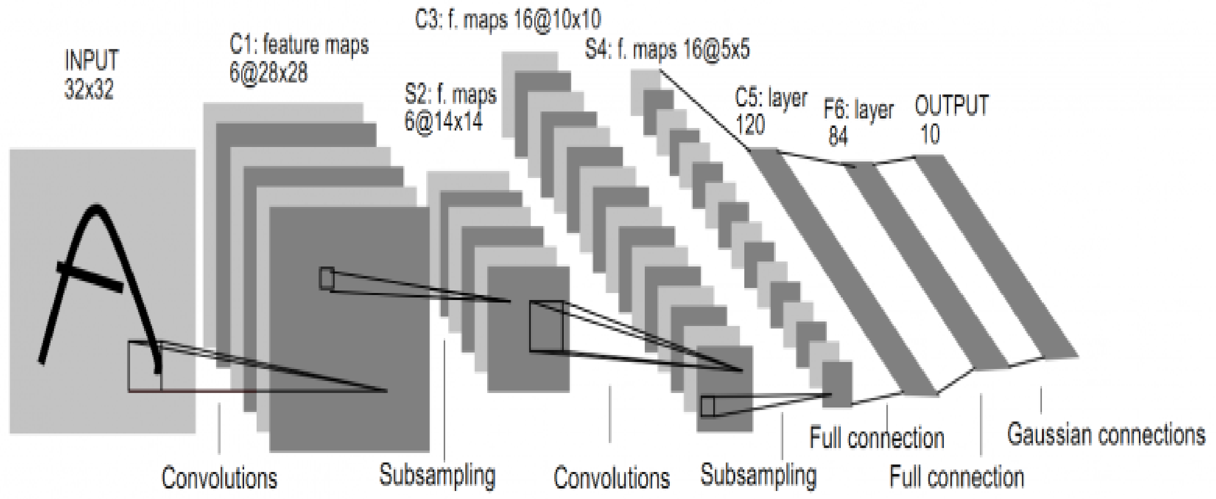

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Our implementation has a slight variation from the image given above

We have 3 important functions that we have used

- BoxConvolution() - This function implements convolution of an Input Box with a 4 dimensional matrix of Weights along with correction with bias. The output matrix Y = W×X + B where X is the input matrix, W is the weight matrix and B is the bias matrix.
- BoxPooling() - This function implements the subsampling and maxpool on the input matrix. Maxpool is used in this implementation over average pool, if the image is zoomed out maxpool keeps the image clearer than average pooling with a higher intensity for the gray scale image.

- FullyConnectedBox() - Applies Relu on the input Matrix, an input argument in each layer will be provided which mentions if they layer has ReLu applied to it or not.

The implementation has a total of six layers which takes an input image and first applies convolution, then max pooling, then again convolution and average pooling and then a fully connected convolutional layer with feature maps each of size 1×1. Another layer is a fully connected layer and then finally a softMax layer which corresponds to probabilities for digits from 0-9.

In the Week3 implementation, the code till Subtask 2 has been used using a header file.

This implementation of LeNet can be used using g++ on a linux machine with version c++11.

The command to run it is ::

```
g++ -stdc==c++11 -pthread Week3.cpp -o Week3.out
./Week3.out
```