Implementation of *LeNet-5* Architecture using self created Image Processing Library

KARAN TANWAR AND HARSH YADAV

Corresponding Repository: Design-Practices-COP290

Compiled March 5, 2019

© 2019 Design Practices in Computer Science

http://www.cse.iitd.ernet.in/~rijurekha/teaching.html

1. INTRODUCTION

Here we developed different convolution filters with and without padding using many different tools like INTEL MKL, OpenBLAS, and pthreads. We compared the performance of all those and results were not linear to what was expected. Next, we created other different functions like Pooling, Sigmoid, Softmax, Relu and tanh activation. [All these funcitons are in the folder imgProcess]

This image processing lirary was used to implement LeNet-5 function in which all 6 layers are implemeted including fully connected layers.[Pre trained extraction from text file done using takeFilters.h]

2. COMMANDS FOR EXECUTION OR USING THE SOURCE FILES

A. ** [LeNet-5 Library] Make sure to edit the intel mkl path according to the machine you are operating (By default its in /home/karan/). Also run export MKLROOT=~/intel/mkl/ **

>make: Takes **input image name** for recognition (should be in png format) and gives three numbers with highest softmax probability.

B. ** [Image Algebra Library] For using image processing library functions goto imgProcess subfolder **

make: compile the main.cpp file

run: ./main.out [function] matrix1.txt [numrows] matrix2.txt [numrows] [stride/{max/avg}]

Examples:

./main.out convolution_withpadding matrix1.txt 5 matrix2.txt 2

./main.out convolution_withoutpadding_matrixmult matrix1.txt 5 matrix2.txt 2 pthread/openblas/intelmkl/simple(*simple means using topleitz method *)

./main.out pooling avg/max matrix1.txt 5 [stride]

./main.out relu_activation matrix1.txt [matrix1 size]

./main.out softmax matrix1.txt [matrix1 size] [stride]

3. FUNCTION DEFINITIONS

1. Conv_1(Vector_Matrix_Float image): Takes 28*28 image 2D matrix and apply first convolution filter resulting in 24*24 image with output channel being 20.

2. Pool_1(Vector_Tetris_Float v): Input: 24*24*20 | Output: 12*12*20 | Pooling: max | Stride: 2 **3. Conv_2(Vector_Tetris_Float v)**: Input: 12*12*20 | Filter: 5*5*20 | Output: 8*8*50

3. Pool_2(Vector_Tetris_Float v): Input: 8*8*50 | Output: 4*4*50

4. FC_1(Vector_Tetris_Float v): Input: 4*4*50 | Output: 1*1*500

5. FC_2(Vector_Tetris_Float v): Input: 1*1*500 | Output: 1*1*10 | Addn. Information: Until now image has been operated on with different convolution filters.

6. mainController(string Filename): Takes image name as input and convert it into matrix; then return its corresponding softmax probabilty of all the digits 0-9 using all of above layers.

7. maxExpectations(mainController(ïmage.txt)̈): results in displaying 3 digits with highest softmax probabilities using the text file created by another python script preprocess.py [in lenet subfolder].

4. REFERENCES

1. https://software.intel.com/en-us/product-code-samples?value=20825value=20802 2. https://tex.stackexchange.com/questions/
border-style 3. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf 4. https://engmrk.com/lenet-5-a-classic-cnn-
architecture/ 5. https://github.com/xianyi/OpenBLAS/wiki/User-Manual 6. https://stackoverflow.com/questions/35569545/linl
a-program-with-both-openblas-and-intelmkl

5. [PERFORMANCE COMPARISION] FIGURES AND TABLES

Intel MKL was proved to be most useful for the implementation of LeNet-5 on the images particularly 28*28. In fact, INTEL MKL was very fast even at the size of matrices being in the range of 20,000. But MKL was not as much fast in very small values like 10*10 in which openBLAS was the only method useful.

[Acceleration with pthreads]

pthreads was indeed the most difficult one to implement because the syncing of all threads and was taking more time for thread handling than matrix convolution for fairly small image sizes like 20*20. Thus it was decided to go with mkl in implementing lenet-5 architecture.

Image Matrix Size	INTEL MKL (millisecond)	openBLAS (millisecond)	pthread (millisecond)
40*40	4.78E-05	1.28E-06	2.42E-04
1000*1000	1.515E-03	1.32E-03	2.281E-03
2000*2000	3.8E-03	4.51E-03	9.53E-03

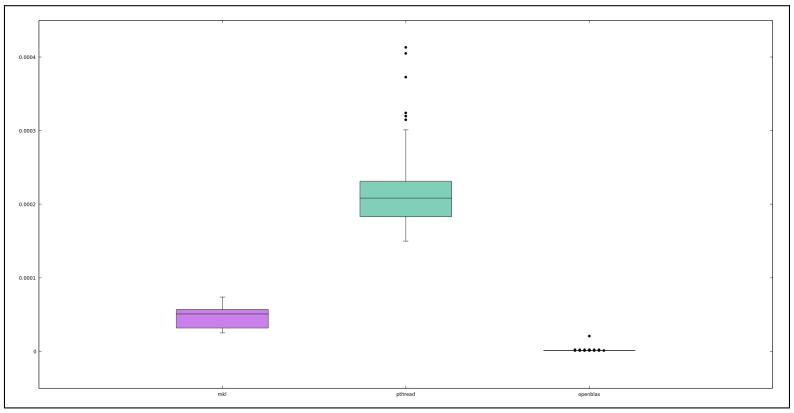

Fig. 1. Comparision of different convolution libraries for image matrix of size 40*40

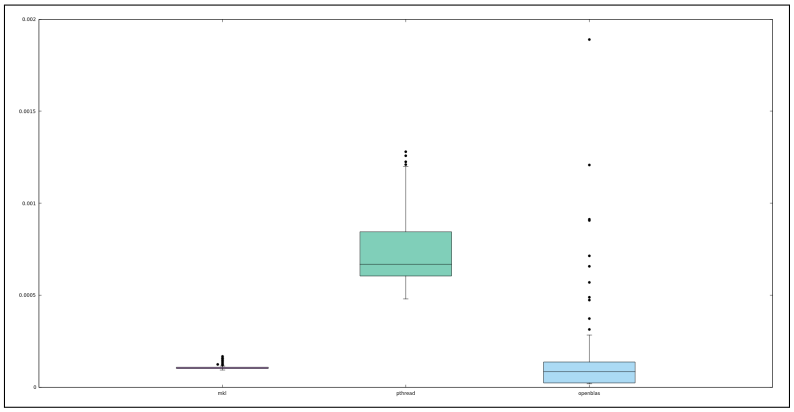

Fig. 2. Comparision of different convolution libraries for image matrix of size 400*400

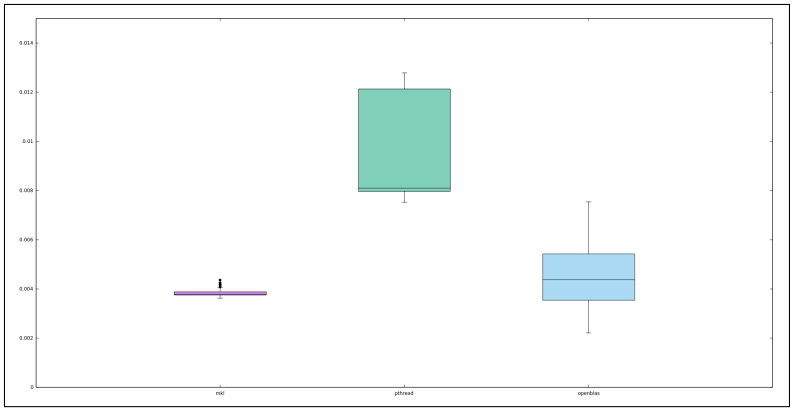

Fig. 3. Comparision of different convolution libraries for image matrix of size 40*40