Lightweight Convolutional Neural Network Framework

Simeng Liao(simengl), Zhejin Huang(zhejinh)

URL: https://simeng96.github.io/Lightweight-CNN-Framework/

SUMMARY

We are going to implement a Lightweight Framework for Convolutional Neural Network training and inferencing using C++ and CUDA.

BACKGROUND

Convolutional Neural Network is a class of deep neural network used for visual or speech related tasks. A detailed explanation can be found in http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf

The most tricky part of the Convolutional Neural Network is Convolution Layer. The algorithm of performing a forward calculation and backward propagation on a tensor (a high dimension matrix) is shown as follows.

Pseudo code for forward passing (by Prof. Bhiksha Raj)

```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for j = 1:D_l
     for x,m = 1:stride(l):W_l-1-K_l+1 # double indices
        for y,n = 1:stride(l):H_l-1-K_l+1
          segment = y(l-1, :, x:x+K_l, y:y+K_l) #3D tensor
          z(l,j,m,n) = W(l,j).segment #tensor inner prod.
          Y(l,j,m,n) = activation(z(l,j,m,n))

Y = softmax( {Y(L,:,:,:)} )
```

Pseudo code for backward propagation (by Prof. Bhiksha Raj)

```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for j = 1:D_l
     for x = 1:W-K+1
        for y = 1:H-K+1
          dz (l,j,x,y) = dY(l,j,x,y) f'(z(l,j,x,y))
          dY(l-1, :, x:x+K_l, y:y+K_l) +=
              dz(l,j,x,y)W(l,j) #tensor scalar prod.
          dW(l,j) += dz(l,j,x,y) Y(l-1,:,x:x+K_l,y:y+K_l)
```

From the above pseudo code, we can see that there exists potential of parallelism in both channel dimension and spatial dimensions. In this project, we aim to fully explore the parallel strategy to let the convolutional calculation more efficient with respect to variable tensor size, which is common in real-world Convolutional Neural Network.

THE CHALLENGE

1. The computation requirement and memory requirement vary a lot with respect to tensor size and the number of parameters. It is hard to come up with a universal strategy to accommodate for different tasks.
2. Efficient memory handling is tricky when the amount of computation varies. We will use different levels of memory (Cache and Main Memory) when faced with different kinds of tasks.
3. To implement a framework for CNN, we have to implement all the components which are necessary to run a model, including image preprocessing functions provided in OpenCV, data-loader and optimizer(such as SGD and Adam). If time is limited, we are going to focus more on components that are related to parallel programming and the system should not necessarily be end to end.
4. Error handling is tricky when there are nesty memory-related or cache-related bugs in CUDA, for sometimes we cannot get the appropriate error message.

RESOURCES

1. Andrew Machine GPU and GCP
2. Pytorch source code (may be tricky, we should conduct experiments and play around it)

GOALS AND DELIVERABLES

75%: Implement a runnable CNN framework, which performs 50% as efficient as PyTorch on average(we'll build reasonable and explainable test cases).
100%: Achieves 75% efficiency of Pytorch on average.
125%: Outperforms Pytorch in some cases, while maintaining 75% PyTorch efficiency on average.

PLATFORM CHOICE

We are going to implement tensors in C++ on CPU side and the computation related to tensors in CUDA on GPU side.

SCHEDULE

Week	Task
Nov w1	PyTorch source code study
Nov w2	1. Implement CPU version of 2D Convolutional Layer 2. Midterm report
Nov w3	Implement GPU parallel version of 2D Convolutional Layer
Nov w4	Optimize efficiency
Dec w1	1. Optimize efficiency 2. final report