ISEE

Acceleration of Convolutional Neural Network Based on CPU-GPU Hybrid Programming

薛盛可[1]，叶慧敏[2]

College of Information Science and Electronic Engineering

Zhejiang University

Spring, 2016

[1]ID: 11531015, e-mail: xueshengke@zju.edu.cn
[2]ID: 11531016, e-mail: 1042430841@qq.com

Abstract

Objective:
We focus on accelerating the computation of an algorithm, namely Convolutional Neural Network (CNN), which is prevalent and widely employed in the field of computer vision, image classification, recognition, etc.
With the help of power of Graphics Processing Units (GPUs), we obtain considerable results on efficient computation.

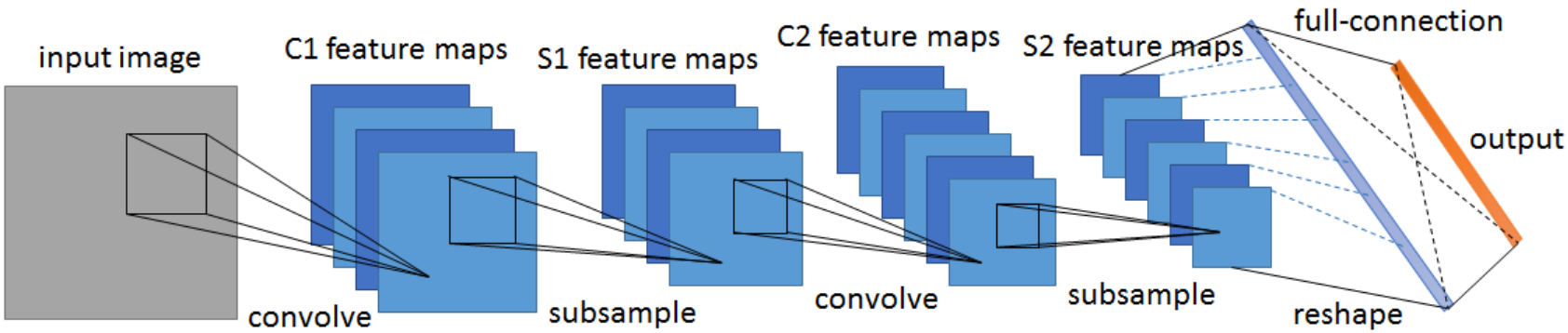
Our presentation is organized as follows:

Outline

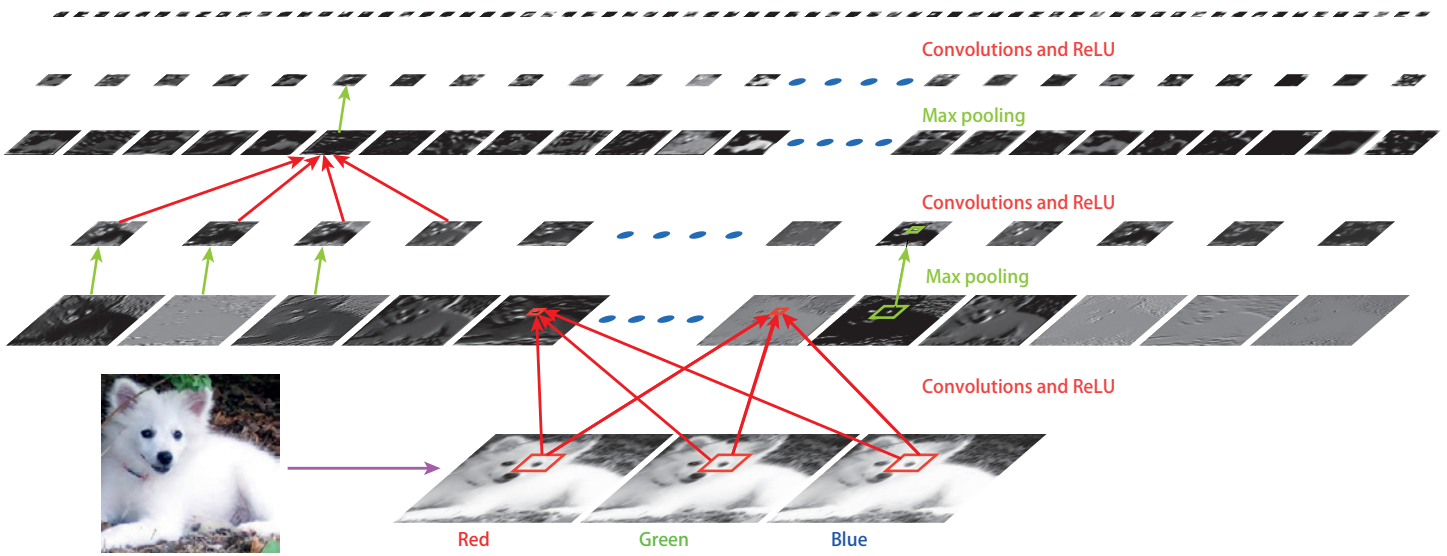
Outline

Convolutional Neural Network

CNN, which is a special artificial neural network, is designed to process data that come in the form of multiple arrays, for example a color image composed of three 2D arrays containing pixel intensities in the three color channels. The rough model is given below,



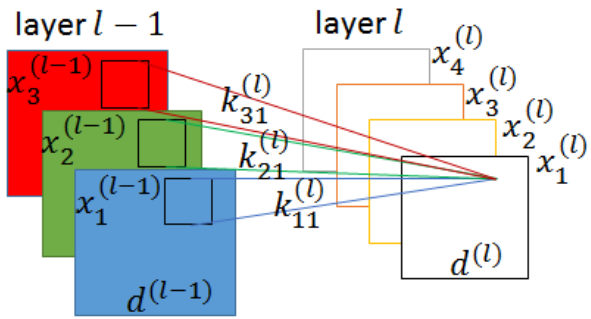
Architecture

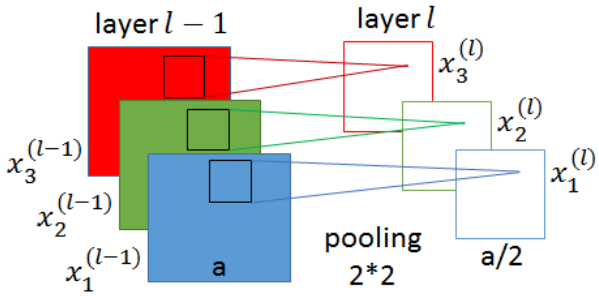


CNN is structured as a series of stages: convolutional layers and pooling layers with fully-connected layers at last. Each unit in a feature map is connected to the local patches in the feature maps of previous layers through a set of weights called a filter bank.

Algorithm

In the architecture above, there are hundreds of thousands of parameters inside. Hence massive computation is involved for updating them in training procedure.

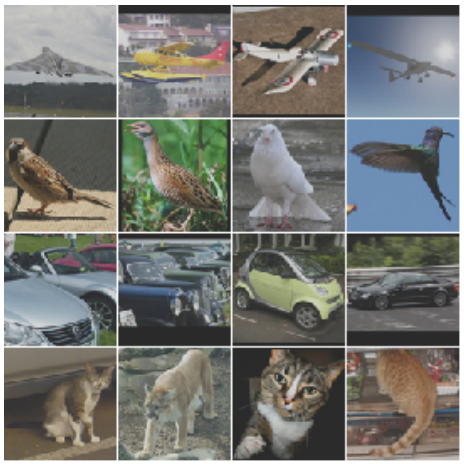


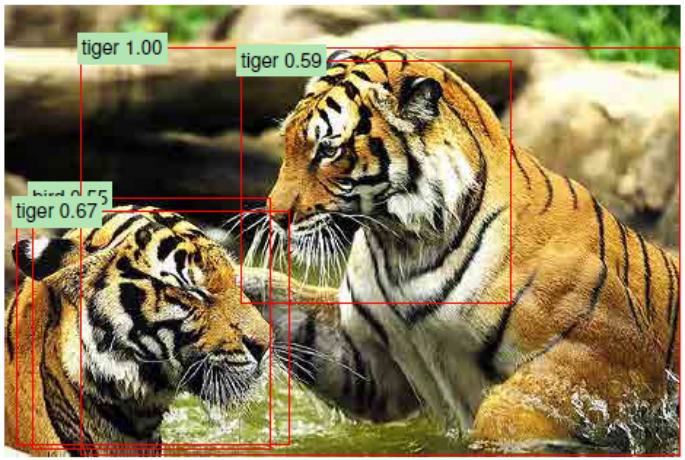


In fact, we compute each feature map in a layer one by one, which is an usual way to do. So the training process of CNN is time-consuming. But what will happen if we find out an approach that parallels the computation without altering the entire training procedure?

Application

- Image Classification [Russakovsky et al., 2015]
- Object Detection [Girshick, 2015]
- Face Recognition [Sun et al., 2014]
- Semantic Segmentation [Long et al., 2015]





Current researches are leaded by academic institutions (New York University, University of Montréal, University of Toronto, UC Berkeley, Standford University, etc.) and enterprises (AI research of Google, Facebook, Microsoft, etc.).

Outline

CUDA

CUDA [3] is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by using the power of the graphics processing units (GPUs). With millions of CUDA-enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for GPU computing with CUDA.

Tesla, Quadro and GeForce series GPUs are capable to deploy CUDA.

[3] http://www.nvidia.com/object/cuda_home_new.html

Installation

CUDA is well supported and cross-operating system. For we have CentOS 7 Linux running already, we download CUDA 7.5 Toolkit and install it without much effort.

Our Graphics card is NVIDIA Quadro K4200,

CUDA cores	1344
GPU Memory	4GB GDDR5
Memory Interface	256 bit
Memory Bandwidth	173 GB/s

GPU Compute Capability [4]:

GTX 1080	6.1
Quadro K4200	3.0
GTX960m	5.0

[4] https://developer.nvidia.com/cuda-gpus

Outline

Caffe

Caffe [Jia et al., 2014] is a deep learning framework made with expression, speed, and modularity in mind.

It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors.

Yangqing Jia created the project during his PhD at UC Berkeley.

Caffe

- Expressive architecture: Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine.
- Extensible code: In Caffe's first year, it has been forked by over 1,000 developers and had many significant changes contributed back.
- Speed: Make Caffe perfect for research experiments and industry deployment. Caffe can process over 60M images per day with a single NVIDIA K40 GPU. That's 1 ms/image for predicting and 4 ms/image for learning.
- Community: Caffe already powers academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia.

Installation

- read the tutorial;
- download open-source code;
- install prerequisites and dependences (CUDA, BLAS, Boost, protobuf, glog, gflags, hdf5,lmdb, leveldb, Python, OpenCV, $\cdots$);
- compile all Caffe source code (about 3 hours compilation fully running on CPU);
- finish. Ready for use.

Configuration

At first, Caffe deployment requires Linux operation system such as Ubuntu, RHEL, CentOS and Fedora. It goes through more complicated procedure although Caffe is compatible with Windows at present. Therefore, we choose CentOS as our target.

Caffe is able to run on CPU certainly, which is a great waste. NVIDIA Graphics Card supporting CUDA with compute capability $\geqslant 3.0$ will reinforce the power of Caffe. Fortunately, we have Quadro K4200 available to use.

Installation process is not as easy as we think, for numerous problems (packages not found, compilation with different parameters, the order of installation, difference of directory, etc.) which are not mentioned in tutorial, occurring in this period. In our case , it takes us four days struggling with it.

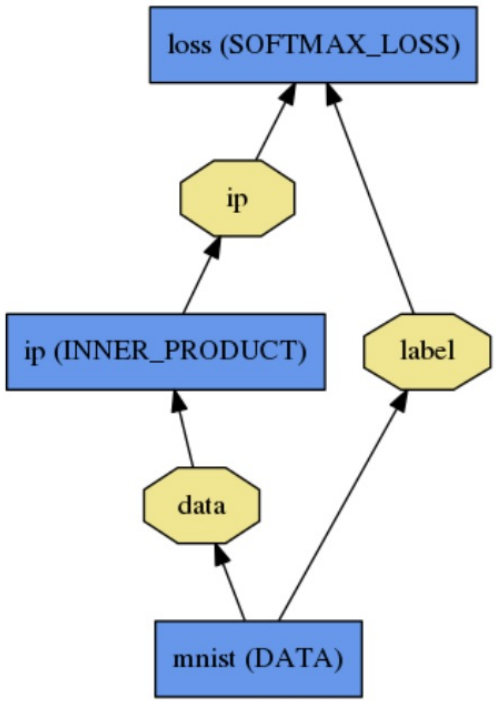
Usage

After Installation and compilation successfully, Caffe is prepared to use.
For instance, we want to a CNN algorithm for image classification.
We go through the following steps:

1. download corresponding data set in lmdb format;
2. define a specific CNN model and write the code with Google Protobuf (a mixed language data criterion);
3. set parameters (learning rate, weight decay term, momentum, training iterations and CPU or GPU mode);
4. train the CNN model with Caffe;
5. test the performance of the generated CNN model.

Protobuf

It is invented by Google Inc., at first. Then it is contributed to open-source community and becomes very popular. It looks like this,

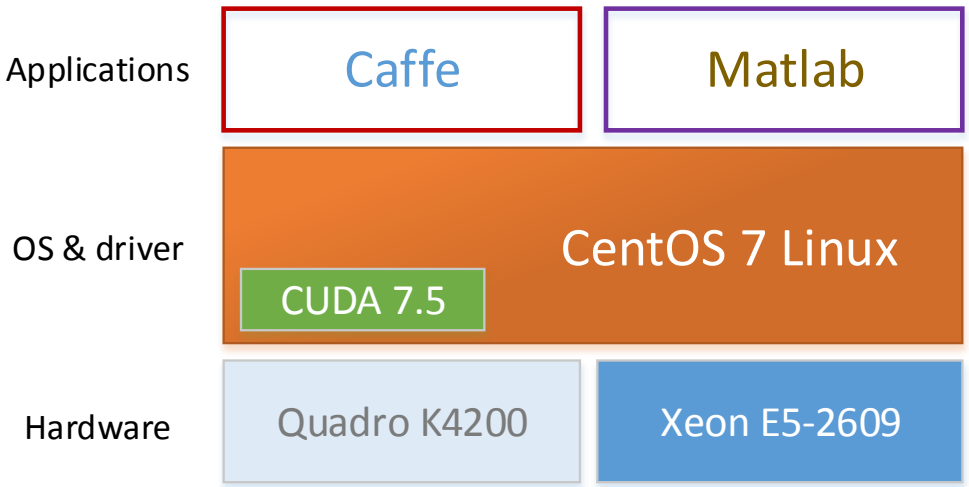



```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Outline

Environment

Linux Kernel: 3.10.0-327.4.4.el7.x86_64
Operating System: CentOS 7 Distribution
Memory: 128 GB
CPU: Intel Xeon(R) E5-2609 @ 1.90 GHz × 12
Graphics: NVIDIA Quadro K4200, 4096 MB

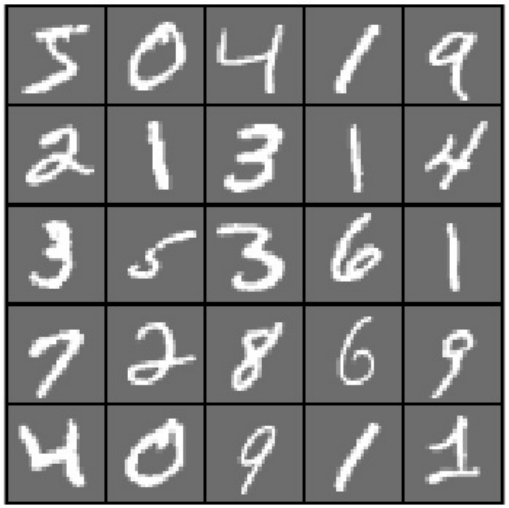


Experiments

We do experiments about image classification of CNN based on two well-known public datasets, i.e. MNIST and CIFAR-10. Each one of datasets are tested both on CPU (implemented by Matlab) and GPU (implemented by Caffe), respectively.

MNIST

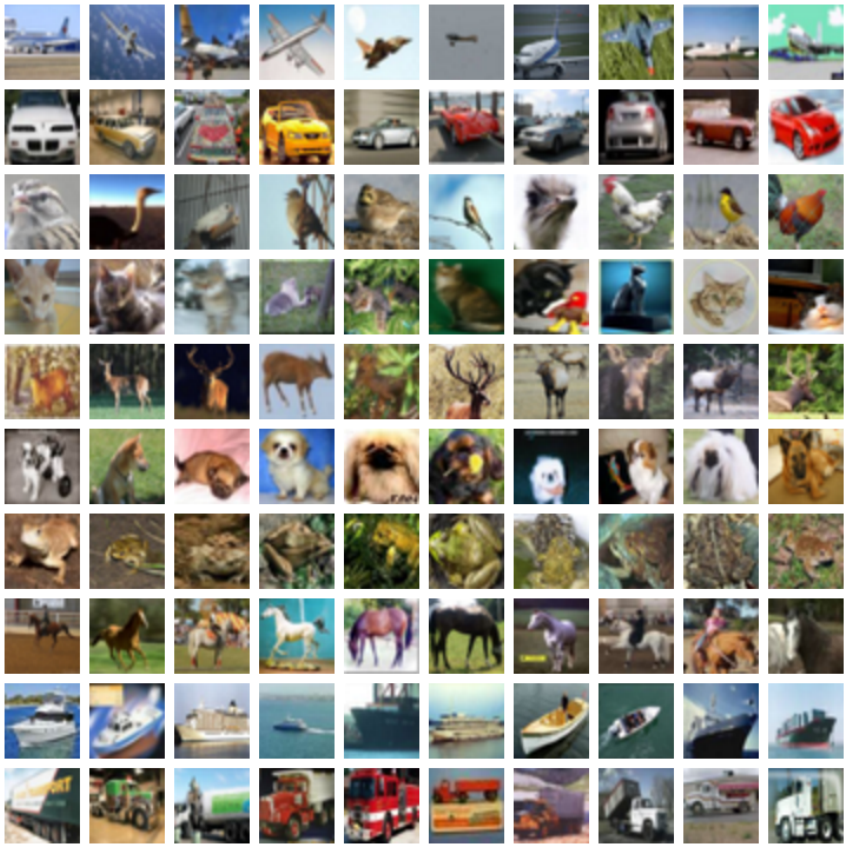
MNIST [5] is a handwritten digits dataset composed of 60,000 training images and 10,000 test images with a 28x28 size. It has been used for examining various algorithms and the state-of-the-art performance is held by CNN with the classification accuracy 99.77 % [Ciresan et al., 2012].



[5] http://yann.lecun.com/exdb/mnist/

CIFAR-10

CIFAR-10[6] is collected by Alex Krizhevsky and Geoffrey Hinton at University of Toronto. The dataset consists of 60,000 32x32 color images in 10 classes with 6,000 images per class. There 50,000 training images and 10,000 test images.

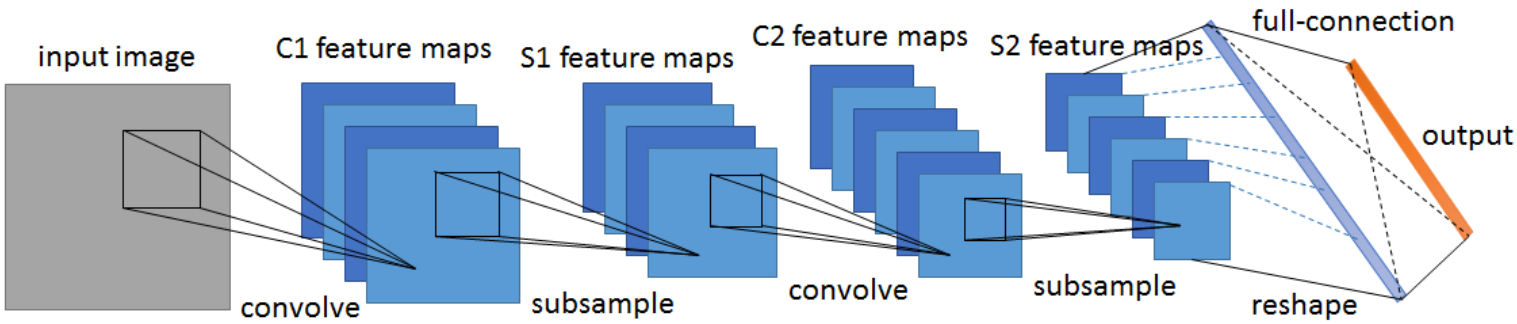


[6] http://www.cs.toronto.edu/~kriz/cifar.html

Design

We design the following two CNN models for MNIST and CIFAR-10, respectively.
For MNIST:

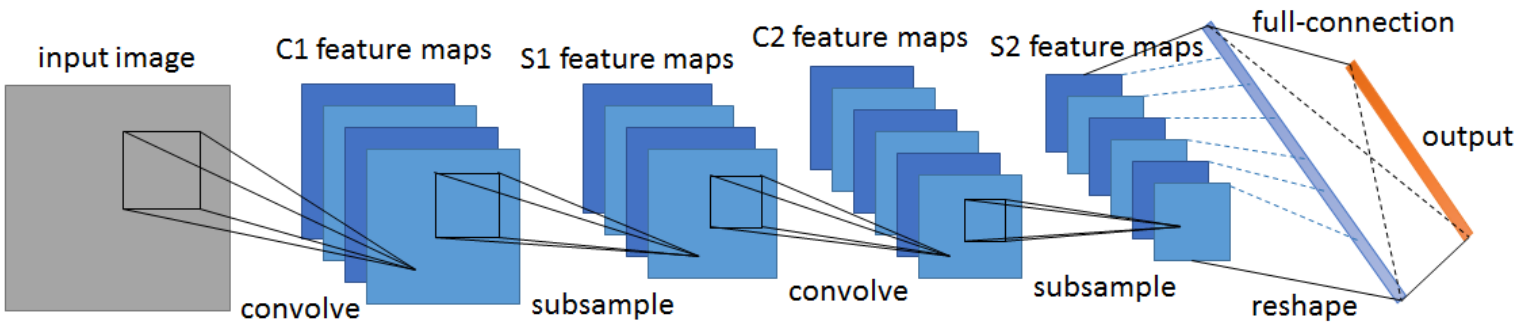
conv1	5x5x20	fc1	500
pool1	2x2	fc2	10
conv2	5x5x40	iteration	10000
pool2	2x2	learning rate	0.01



ISEE

For CIFAR-10:

conv1	5x5x32 pad: 2	fc1	64
pool1	3x3 stride: 2	fc2	10
conv2	5x5x32 pad: 2	iteration	4000
pool2	3x3 stride: 2	learning rate	0.001
conv3	5x5x64 pad: 2		
pool3	3x3 stride: 2		



Running on CPU

We implement our CNN algorithm through Matlab. The script includes these steps,

1. load dataset;
2. initiate the CNN model;
3. train our model using training set;
4. test our model using test set;
5. record the performance (elapsed time and accuracy).

Running on GPU

We implement our CNN algorithm through Caffe. The code fitting to protobuf only contains the definitions of exact parameters of CNN given previously, for the entire structure is accomplished and clear enough to modify. With a single command, all procedures run automatically.

```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
```

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

Outline

Performance

We record the accuracy of two experiments. No surprisingly, algorithms distributed on CPU and GPU reveal the identical performance, as the figures illustrate,

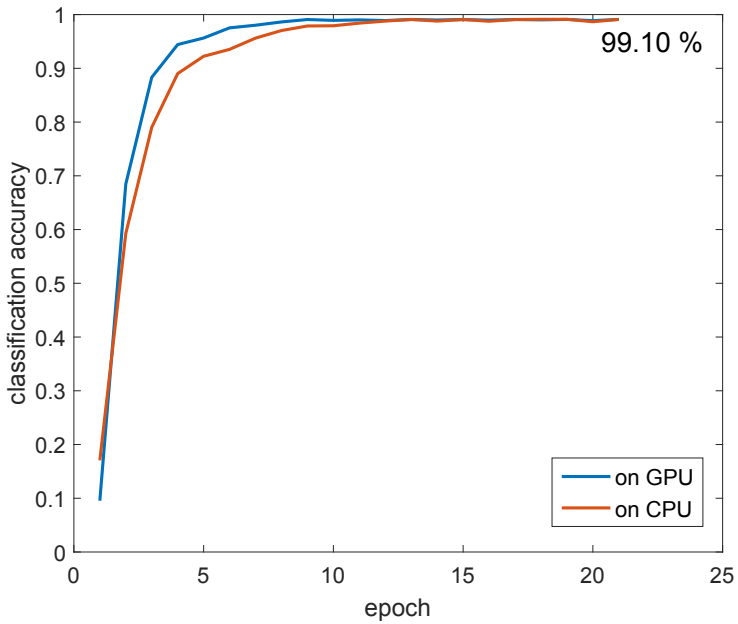


Figure 1: Accuracy of CNN on MNIST

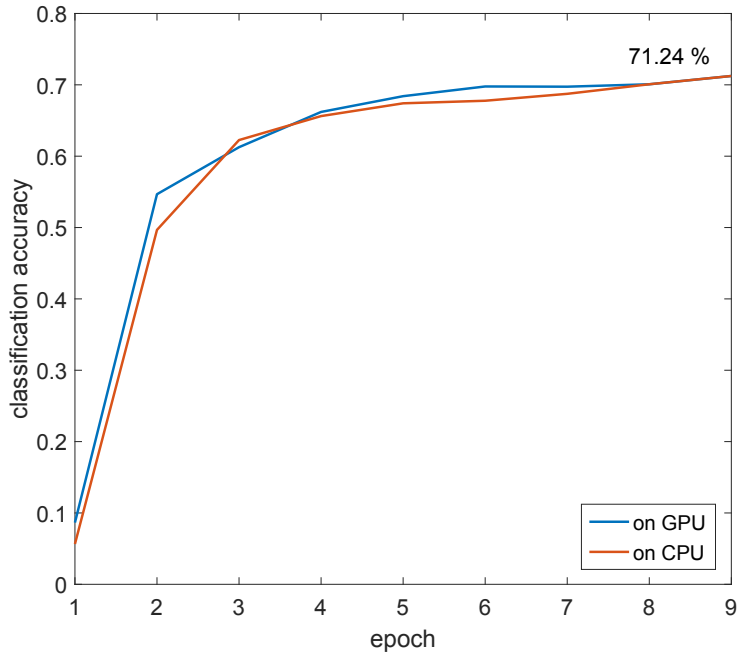


Figure 2: Accuracy of CNN on CIFAR-10

Time Expenditure

MNIST running on CPU takes 6554.47 seconds totally. Convolution occupies the majority of time, that is, 6190.27 seconds.
CIFAR-10 running on CPU takes 19125.40 seconds totally. Convolution occupies the majority of time, that is, 12220.82 seconds.

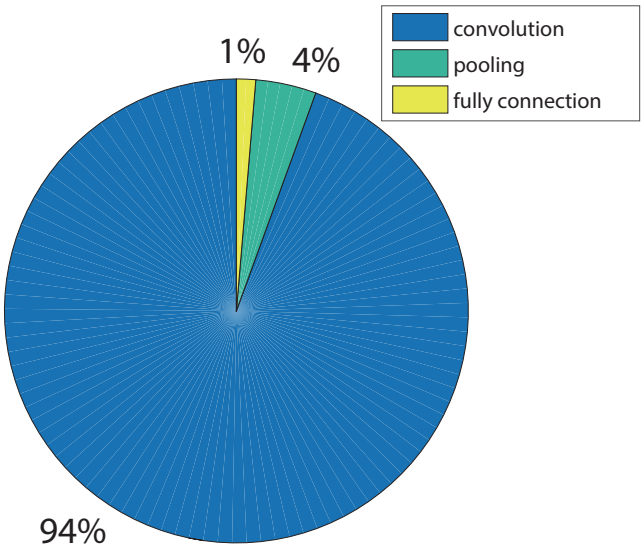


Figure 3: Elapsed time of CNN on MNIST

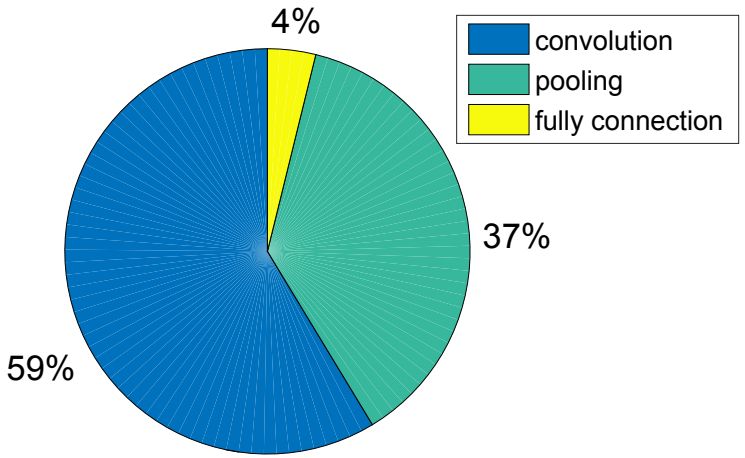


Figure 4: Elapsed time of CNN on CIFAR-10

Time Expenditure

However, with help of GPU involved in computation, MNIST running on CPU takes 242.28 seconds totally, which is quite efficient compared to CPU.
CIFAR-10 running on GPU takes 468.76 seconds, much faster than on CPU, too.

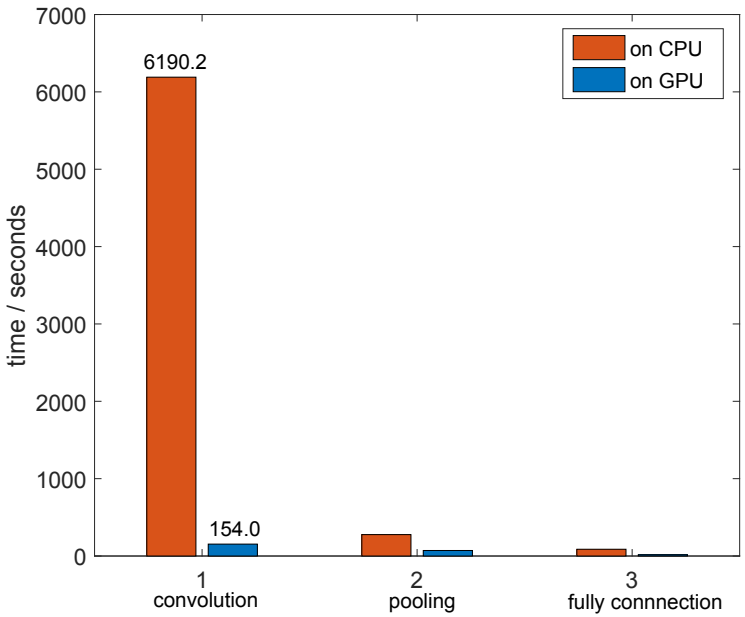


Figure 5: MNIST: comparison between CPU and GPU

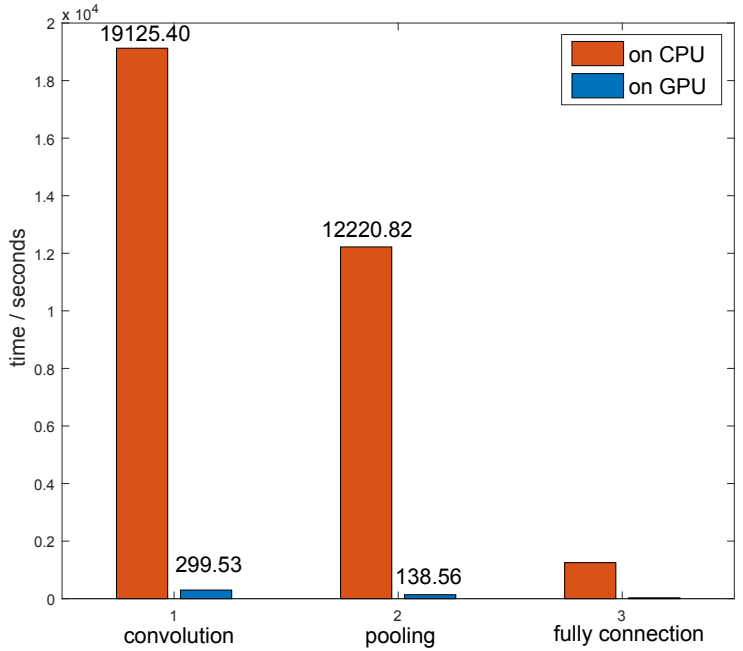


Figure 6: CIFAR-10: comparison between CPU and GPU

Outline

Conclusion

Current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of large CNNs with dozens of times faster than preceding one on CPU.

The acceleration of computation in CNNs using GPUs has been proved to be a successful leap, saving valuable time on researches.

Outline

References I

Ciresan, D., Meier, U., and Schmidhuber, J. (2012).
Multi-column deep neural networks for image classification.
In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE.

Girshick, R. (2015).
Fast r-cnn.
In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014).
Caffe: Convolutional architecture for fast feature embedding.
arXiv preprint arXiv:1408.5093.

Long, J., Shelhamer, E., and Darrell, T. (2015).
Fully convolutional networks for semantic segmentation.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

References II

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015).
ImageNet Large Scale Visual Recognition Challenge.
International Journal of Computer Vision (IJCV), 115(3):211–252.

Sun, Y., Wang, X., and Tang, X. (2014).
Deep learning face representation from predicting 10,000 classes.
In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

End

Thanks for your listening.

感谢！