Midterm Update of Image Classification based on CNN with Cloud Platform

Tianyang Liu
Electrical and Computer Engineering
University of Florida
Gainesville, US
dotasheepliu@ufl.edu

Yicheng Wang
Electrical and Computer Engineering
University of Florida
Gainesville, US
yichengwang125@ufl.edu

Su Pu
Electrical and Computer Engineering
University of Florida
Gainesville, US
Soupsoup88@ufl.edu

***Abstract*— ImageNet provides a huge dataset of images in different categories. While local machines do not have enough computational and storage resource to process them, cloud computing has the ability. In the past weeks we studied the deep Convolutional Neural Network image classification framework as well as implementation details in TensorFlow. Research has been done on the training speed of different platforms and devices including CPU / GPU. In case of performance we modified various learning parameters of CNN middle layers, trained models with CIFAR-10 then evaluated models using recognition rates, parameter counts and converging epochs. Future work since midterm will be on the construction of models for ILSVRC2012 image classification using GPUs on AWS and HiperGator.**

Keywords—Midterm Update; ImageNet; TensorFlow;CIFAR-10;GPU;

I. INTRODUCTION

Analyzing and classifying pictures is quite a heating topic in nowadays computer vision field. This idea which evolved from artificial network explains how the machine is going to understand and learn the objects from the pictures. The associated database ImageNet which is leaded by Feifei Li is the most prestigious and largest research resources. The Large Scale Visual Recognition Challenge (ILSVRC) competition is based on this database and is held every year since 2010. The result from every year competition is improving very fast and the overall accuracy can reach more than 95%. However, the requirement of the computation resources is very critical, and both calculation and storage take up great amounts of time and space. Fortunately, the cloud computing is quite mature to help the researchers deal with high performance calculation. Cloud computing relieves the pressure of computing resources when processing big data and big algorithm. Cloud computing saves money for the companies from buying servers and provides them with enough disk space for high demand computation. By using third party data centers like AWS and gcloud, users and enterprises from different fields can share the same resources and dynamically change their requests.

Convolutional Neural Network (CNN) is a second boost to ILSVRC. In machine learning area, CNN is a kind of feed forward artificial neural network where the connection characteristics among neurons origins from the arrangement of animal visual. Each visual neuron would respond to overlapping regions affecting the visual field, which mathematically could be explained as a convolution operation, the basic working scheme of CNN. CNN consists multiple layers of receptive convolution fields, and may include local and global pooling layers combining the outputs of neural clusters. The convolutional layer features parameter sharing and local connectivity, which are the two important characteristics determining its potential advantage. Due to its superior classification ability, CNN has become a superstar in ILSVRC competition since the year 2011. In addition, deeper CNN and an increasing count of neurons should bring an even higher correct rate, therefore we can see that most CNNs today are deep and fat.

Based on these technologies, this project plans to realize an even more accurate image classification using CNN neural network method from a previous paper proposed by Alex research group. In this report the following part explains some previous work on images processing, then next the overview of architecture. It will also provide a detailed system and subsystem design and anticipated performance evaluation methods.

II. REALTED WORK

Convolutional Neural Network (CNN) was born in 1990s and is growing super-fast in this decade: since 2011 CNN has been taking a dominant position in ImageNet LSVRC. CNN simulates how human brains are working and therefore have great potential in the field of computer vision. Article [1] is an important attempt to apply CNN in ImageNet classification where 60M parameters were used to construct five convolutional layers, several max pooling layers and three fully connected layers, resulting in a winning top five score. Moreover, it verified the feasibility of CNN and the importance of depth of convolution.

Numerous works upon [1] were published. Szegedy et al. [2, 3] concentrated on increasing the width and depth of CNN while maintaining the computational budget constant. It came from the thinking that although the increased model size tends to translate

to immediate quality gains in most cases, the efficiency and low parameter count are still crucial in cases like mobile vision, and was implemented through factorized convolution and aggressive regularization. Eventually this 22 layers structure won the first place in 2014. Deep CNN inevitably accompanies a large number of hyperparameters, and therefore would be difficult to train. He [4] presented a residual learning framework to resolve the issue of training deep CNN: it explicitly transformed the layers as learning residual functions with regard to inputs instead of the unreferenced functions. As a result, this method could increase performance dramatically especially in those deep nets. CNN has multiple features identical to the visual part of the human brain, but is different in that CNN has a feed forward architecture when brains have lots of recurrent structures. Article [5] researched on this idea and proposed a variation called Recurrent CNN through connecting convolutional layers with recurrent modules. The pros of this structure include the utilization of context information as well as using a small number of parameters to achieve a high recognition rate, i.e. 0.67M parameters versus 7.37 percent mistake in CIFAR-10. While [1] illustrated the importance of depth versus performance, all three variations above increased just a small amount of depth but adopted various techniques to increase the efficiency of each neuron.

Face detection is the most promising subdivision that CNN is working in. Compared to image classification it is primarily facing the problem that faces are always not in perfect condition, which is the common case in real life. In [6] Joshua constructed 10 face detectors different in size and the time of iterations using Caffe and tested with AlexNet. Models were trained on Core i7 plus GT 430 up to seven days to complete 5000 iterations. As a result, it proved the positive relationship between the size of training data versus the performance of CNN based face detection. [7] in depth considered influences like pose, expression and lighting. It first pointed out that accurate discriminative models tend to be computationally prohibitive, then proposed a cascaded CNN operating at multiple resolutions, introducing a calibration stage and dealing with candidates with various strategies. In cases above the authors used two convolutional layers, which should be enough for simple face detection. In potential complex situations like face recognition and image classification, cloud will be utilized to do operations on big data and big algorithm [8], [9], [10].

III. System Architecture

A. *TensorFlow*

TensorFlow is an open source library for machine learning first started by google brain and released on November 9th.

A very special structure of TensorFlow is it is using data flow graphs for numerical computation. The user can adjust the data flow graphs and created its own calculation process. Besides, TensorFlow can also work on different platforms, GPU can increase the speed of computation, and the mobile platform stimulates the development of the industry for machine intelligence field. TensorFlow 0.8 even provide the distributed system, which mean the machine learning process can be work on parallel node and increase the speed.

There are also many good open source machine learning libraries such as Caffe and Mxnet. Since the TensorFlow has more learning resources and can support variety of neuron networks, we choose this library as our final library.

B. *GPU (AWS instance)*

The first phase experiment was implement on single GPU. the GPU we use to generate data is g2.2xlarge on AWS. In the future test, we are going to enlarge the number of GPUs to 2, we also plan to use the HiperGator to realize the multiple GPUs implementation.

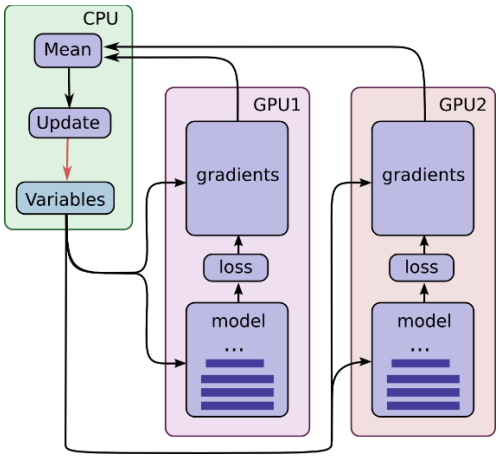

Fig. 1.Structure of Multiple GPUs on Training

C. *CIFAR-10 dataset*

1) *The overview of cifar10*

The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex, Vinod Nair, and Geoffrey Hinton. Alex is also the author who produce the convolution network algorithm in 2012 ImageNet competition.

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

There are also cifar-100 dataset, since we are working on the overall process of the training and the ImageNet database is the final database, we didn't choose the CIFAR100 dataset.

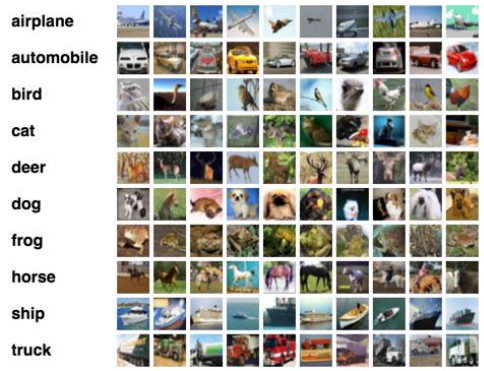

Fig. 2.Sample of CIFAR-10 database

The reason we choose cifar10 is because it contains smaller dataset than ImageNet and has label on every image. After

proving the structure work. We are going to train it with ImageNet database.

2) The reason why we choosing cifar10

The reason we choose cifar10 is because it contains smaller dataset than ImageNet and has label on every image. After proving the structure work. We are going to train it with ImageNet database.

D. Structure of CNN in CIFAR-10 Sample

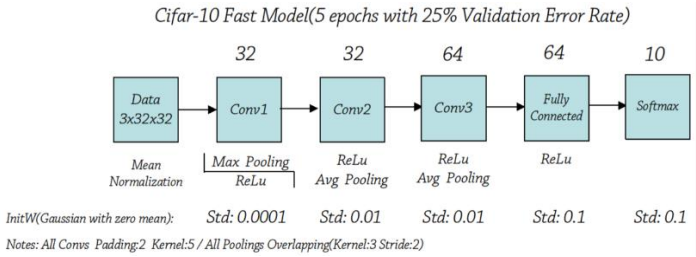

Fig. 3.Structure of CNN in CIFAR-10

The architecture for CIFAR-10 contains three convolution layers each following a pooling process. And the initial weights of each layers are designed based on standard deviation.

IV. THE ALOGRITHM

While traditional multilayer perceptron (MLP) shown in Fig.3, models were successfully used for image recognition, the full connection algorithm would lead to the curse of dimensionality. For example, if the size of input image is 200*200*3 (200 columns, 200 rows, 3 color channels), the weights for input layer to first hidden layer would be 120,000*n (n is the number of neurons in the first hidden layer). While as for these kind of input images, the numbers of hidden layers and neurons in each hidden layer are tremendous.

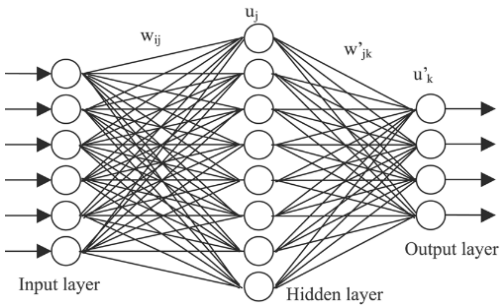

Fig. 4. Structure of MLP

The full connectivity of neurons is wasteful in the framework of image recognition, and the huge number of parameters quickly leads to overfitting. A simple ConvNet mainly contains 3 layers: Convolutional Layer, Pooling Layer, and the Fully-Connected Layer.

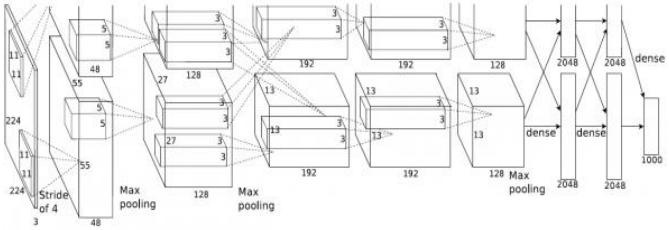

Fig. 5.Structure of CNN

A. Convolutional Layer

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

which means elementwise multiplication and sum of a filter and the signal.

The layer's parameters consist of a set of kernels that extend through the full depth. So the kernels could be regarded as 3-dimensional filters, and each filter gets a 2-dimensional activation map. For example, in Fig.4, the author applied 96 filters to the first layer, the depth of the output is 96.

B. Pooling Layer

The idea behind this layer is that once the feature is found, the exact location is not as important as the rough location relative to other feature. So this is a non-linear down sample layer, which makes the representations smaller and more manageable. For example, the pooling algorithm in Fig.5 is max pooling, which means to segment the whole image into non-overlapping region, and output of each sub-region is the maximum inside. It operates over each activation map independently, so the depth will not change.

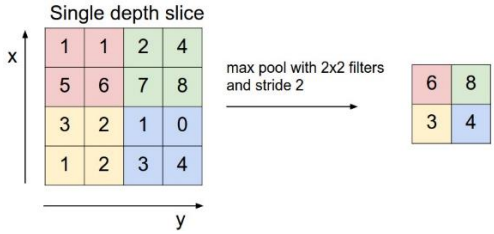

Fig. 6. Max pooling

C. Fully-Connected Layer

Just like the connection method in ordinary NN, the neurons of one layer connect all the neurons in previous layer, which means there are N*M parameters (N is the number of neurons in present layer, and M for the previous layer).

D. ReLU

An activation function called Rectified Linear Units is used in this structure, comparing with traditional sigmoid function, ReLU naturally leads to sparse networks and is closer to biological neurons' responses in their main operating regime. With rectifier units as nonlinear activation function, the deep neural networks don't need pre-training, which is faster and more efficient than sigmoid.

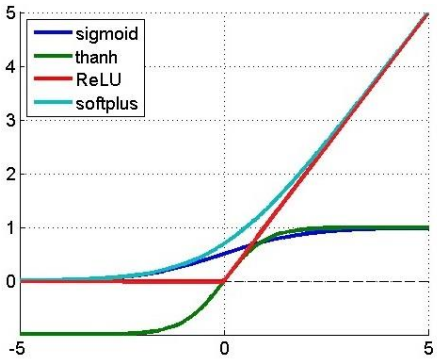

Fig. 7.Different Activation Function

E. Detailed Implement of CNN

1) Input

The input model separates the images into different batch, and the parameters of weights and bias will change after each batch go through the neuron network. There are 50000 images for training, and 10000 for evaluation, and the classes is 10. In the TensorFlow model. The size of the picture is 24*24, which is deferent from the size of 32*32 in the original dataset need to be cropped.

The lines in the **cirfar10_input.py** define the information we discussed before.

```
# Global constants describing the CIFAR-10 data set.
NUM_CLASSES = 10
NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN = 50000
NUM_EXAMPLES_PER_EPOCH_FOR_EVAL = 10000
```

```
# Basic model parameters.
tf.app.flags.DEFINE_integer('batch_size', 128,
                            """Number of images to process in a batch.""")
```

Fig. 8.Partial input code

2) Training

Using the model in TensorFlow, we can easily form the structure in the above picture, TensorFlow creates some variables about weights and bias.

Some important lines in **cifar10.py** to create the training process. Create the convolution network.

```
with tf.variable_scope('conv1') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 3, 64],
                                       stddev=5e-2,
                                       wd=0.0)
  conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
  bias = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(bias, name=scope.name)
```

```
# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool1')
# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm1')
```

Fig. 9.Partial Neuron network code

By the above method, the script also generates three layer of convolution network.

After generate the fully connected layer, the script go through the SoftMax regression.

```
softmax_linear = tf.add(tf.matmul(local4, weights), biases, name=scope.name)
_activation_summary(softmax_linear)
```

Fig. 10. SoftMax Code

To train the parameters, the script calculates the average cross entropy loss across the batch.

```
cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits, labels, name='cross_entropy_per_example')
cross_entropy_mean = tf.reduce_mean(cross_entropy, name='cross_entropy')
tf.add_to_collection('losses', cross_entropy_mean)
```

Fig. 11. Loss calculation

Then the script computes the gradient and update the parameters.

```
# Compute gradients.
with tf.control_dependencies([loss_averages_op]):
  opt = tf.train.GradientDescentOptimizer(lr)
  grads = opt.compute_gradients(total_loss)
```

Fig. 12. Calculate the gradient

3) Evaluation

By defining the flags in the TensorFlow, the interval time of evaluation can be changed, at each interval time, TensorFlow will retrace the checkpoint record and calculate the accuracy.

The following is code in the **cifar10_eval.py.**

```
# extract global_step from it.
global_step = ckpt.model_checkpoint_path.split('/')[-1].split('-')[-1]
```

Fig. 13. Extract data from check point.

```
while True:
  eval_once(saver, summary_writer, top_k_op, summary_op)
  if FLAGS.run_once:
    break
  time.sleep(FLAGS.eval_interval_secs)
```

Fig. 14. Run the evaluation every evaluation second interval.

V. Performance Evaluation

A. Experimental Setting

In the following experiments the compute engines would use AWS GPU instances G2.2XLARGE and G2.8XLARGE. G2.2XLARGE controls 8 virtual CPUs of Intel Xeon E5-2670 and a GPU of Nvidia GRID K520 with 1536 CUDA cores and 4GB visual memory; G2.8XLARGE is four times the performance of it. In case of CPU we would use combinations of virtual CPUs on GCloud. To control the budget, we would also utilize several Nvidia Tesla K80 GPUs within UF HiperGator 2.0 to train part of the data. All operations would Software supports include CUDA 7.5 Toolkit and cuDNN v5 controlled by TensorFlow 0.11.

B. Performance Metrics

In case of CNN training speed, we have programmed a clock to calculate pictures to be processed per second. In case of performance of CNN, we will use CIFAR-10 and ILSVRC2012 datasets as benchmarks to evaluate models we trained, and compare with identification rates of published works. We also need to consider the number of hyperparameters in case of neuron efficiency, and the loss at certain epoch number in case of training efficiency.

C. *Experimental Results*

We have conducted two and will conduct six experiments in total to evaluate multiple aspects of the training process and the final models.

1) *Speed of Different Platforms*

```
2016-10-07 05:07:01.741812: step 500, loss = 3.20 (383.0 examples/sec; 0.334 sec/batch)
2016-10-07 05:07:05.399037: step 510, loss = 3.11 (381.8 examples/sec; 0.335 sec/batch)
2016-10-07 05:07:08.734987: step 520, loss = 3.21 (378.5 examples/sec; 0.338 sec/batch)
2016-10-07 05:07:12.072165: step 530, loss = 3.05 (385.8 examples/sec; 0.332 sec/batch)
2016-10-07 05:07:15.409250: step 540, loss = 3.02 (372.2 examples/sec; 0.344 sec/batch)
2016-10-07 05:07:18.724467: step 550, loss = 3.11 (393.5 examples/sec; 0.325 sec/batch)
2016-10-07 05:07:22.043128: step 560, loss = 3.18 (399.7 examples/sec; 0.320 sec/batch)
2016-10-07 05:07:25.389089: step 570, loss = 3.19 (382.5 examples/sec; 0.335 sec/batch)
2016-10-07 05:07:28.710200: step 580, loss = 2.99 (394.7 examples/sec; 0.324 sec/batch)
2016-10-07 05:07:32.027653: step 590, loss = 3.15 (397.1 examples/sec; 0.322 sec/batch)
2016-10-07 05:07:35.366276: step 600, loss = 3.07 (389.5 examples/sec; 0.329 sec/batch)
```

```
2016-10-07 05:48:11.429976: step 200, loss = 3.89 (82.2 examples/sec; 1.558 sec/batch)
2016-10-07 05:48:29.109686: step 210, loss = 3.61 (80.5 examples/sec; 1.590 sec/batch)
2016-10-07 05:48:44.880040: step 220, loss = 3.84 (82.1 examples/sec; 1.559 sec/batch)
2016-10-07 05:49:00.484082: step 230, loss = 3.71 (81.9 examples/sec; 1.563 sec/batch)
2016-10-07 05:49:16.059764: step 240, loss = 3.65 (83.0 examples/sec; 1.542 sec/batch)
2016-10-07 05:49:31.697198: step 250, loss = 3.70 (82.0 examples/sec; 1.561 sec/batch)
2016-10-07 05:49:47.196845: step 260, loss = 3.73 (82.1 examples/sec; 1.558 sec/batch)
2016-10-07 05:50:02.828827: step 270, loss = 3.49 (82.0 examples/sec; 1.562 sec/batch)
2016-10-07 05:50:18.289893: step 280, loss = 3.63 (83.5 examples/sec; 1.532 sec/batch)
2016-10-07 05:50:34.027672: step 290, loss = 3.62 (80.7 examples/sec; 1.585 sec/batch)
2016-10-07 05:50:49.545020: step 300, loss = 3.71 (82.9 examples/sec; 1.544 sec/batch)
```

```
2016-10-07 05:57:58.867688: step 600, loss = 2.92 (634.0 examples/sec; 0.202 sec/batch)
2016-10-07 05:58:01.180618: step 610, loss = 2.95 (636.7 examples/sec; 0.201 sec/batch)
2016-10-07 05:58:03.231749: step 620, loss = 3.11 (615.2 examples/sec; 0.208 sec/batch)
2016-10-07 05:58:05.298993: step 630, loss = 3.01 (619.0 examples/sec; 0.207 sec/batch)
2016-10-07 05:58:07.372742: step 640, loss = 2.86 (614.6 examples/sec; 0.208 sec/batch)
2016-10-07 05:58:09.444589: step 650, loss = 2.98 (611.6 examples/sec; 0.209 sec/batch)
2016-10-07 05:58:11.499563: step 660, loss = 2.79 (618.1 examples/sec; 0.207 sec/batch)
2016-10-07 05:58:13.567201: step 670, loss = 2.87 (611.8 examples/sec; 0.209 sec/batch)
2016-10-07 05:58:15.631679: step 680, loss = 2.93 (615.7 examples/sec; 0.208 sec/batch)
2016-10-07 05:58:17.729369: step 690, loss = 2.83 (578.7 examples/sec; 0.221 sec/batch)
2016-10-07 05:58:19.784727: step 700, loss = 2.86 (661.1 examples/sec; 0.194 sec/batch)
```

Fig. 15. G2.2XLARGE / 1vCPU / 16vCPUs computing CNN

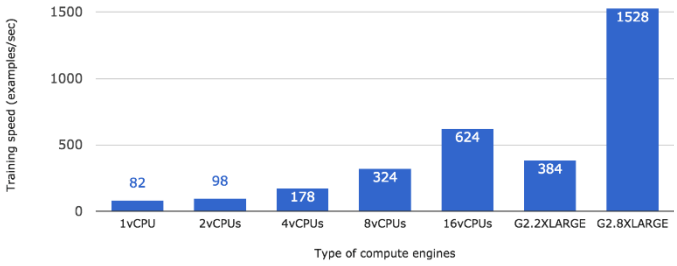

Fig. 16. Comparison of speed of various machines

The first experiment is on the training speed upon various compute engines. We trained a CNN model for CIFAR-10 classification using G2.2XLARGE / G2.8XLARGE / 1vCPU / 2vCPUs / 4vCPUs / 8vCPUs / 16vCPUs respectively then compared the speed as well as the ratios. The speed of initial 200 steps are not stable and therefore should not be considered.

Fig. 15 exhibits the training process of TensorFlow CNN on CIFAR-10; Fig. 16 shows the average speed of each type of compute engines. It is interesting to see from Fig. 2 that the CPU parallelization is not always ideal and the increase of the virtual CPU count from 1 to 16 could just increase the speed by 1.2 / 1.8 / 1.8 / 1.9 all lower than two. The increase of GPU cores from 1 to 4 however, results in 3.9 almost full acceleration. When compare between CPU and GPU, we may say GPU has a much higher speed though it may also cost more budget. Overall we decided to move on with GPU to do future model training, possibly using several Nvidia Tesla K80 cards on HiperGator.

2) *Overlapping pooling*

In traditional CNN, the pooling layer is not overlapped. Not overlapping means no influence between adjacent pixels, but it will decrease the accuracy of the net. And in Alex's paper, they found overlapping pooling can somehow avoid overfitting.

So when test with CIFAR-10, we did the following test. We set the kernel size as Conv1: (3,3), Conv2: (4, 4), Conv3:(5, 5) and the pooling size is all (2,2). The batch size here is 100, one epoch means 500 steps. As the table below shows, when we used overlapping pooling, learning rate drops down rapidly at epoch 9.

Fast 10 epochs Validation Error Rate										
Epoch/Model	1	2	3	4	5	6	7	8	9	10
Overlapping	44%	37%	34%	30%	30%	30%	30%	30%	25%	25%
Not Overlapping	52%	44%	39%	37%	35%	34%	33%	32%	29%	29%

So with the overlapping pooling structure we can get better accuracy and faster learning speed.

3) *Future experiments*

The third experiment will be on various number of convolutional layers versus the first place identification rate. This experiment as well as the following two would be conducted on CIFAR-10 because it is comparatively small while useful. It will give us ideas on how to build large models for ILSVRC2012. The fourth experiment is on the properties of the fully connection layer where we can adopt techniques from published papers like jump over and recursion. The fifth experiment is on parameter count versus identification rate on various CNN models conducted base on ILSVRC2012. And the last experiment will be on loss versus epochs on different CNN models conducted base on ILSVRC2012.

VI. Current Progress and Project Management

A. *Current Status*

So far we have done works on getting familiar with the ML library environment and analyzing the algorithm on AWS and GoogleCloud. We used CIFAR-10 database to verify the CNN structure, and learned to change parameters likes kernel sizes, pooling types and numbers of layers etc. Some milestones include:

1)

Test the default TensorFlow model on various CPU and GPU types on AWS and GoogleCloud. As we trained on G2.2XLARGE / G2.8XLARGE / 1vCPU / 2vCPUs / 4vCPUs / 8vCPUs / 16vCPUs respectively, we arrived at the conclusion that GPU on AWS has a much higher speed than multi CPUs on gcloud though it may also cost more budget. An economic way is to use several GPUs on UF HiperGator 2.0.

2)

Learn the theoretical knowledge of CNN. We researched on the basic idea of CNN then looked at papers of several models from ILSVRC2012 to ILSVRC2015 like AlexNet, GoogleNet and ResNet. We analyzed the reasons why these models can achieve superior results, whose concepts could also be applied in our own model.

3)

Learn to change structures and parameters of CNN. Based on CIFAR-10 database, we compared the error rate and learning rate of different numbers of layers and pooling methods. So

somehow get an idea on how to achieve higher accuracy but this is only limited to CIFAR-10 i.e. a relative small dataset. When working on an enormous database like ImageNet, the problems could be more complicated.

B. Team Coordination

We have multiple tasks to finish and will assign tasks among three team members. Tianyang the coordinator will continue working on CIFAR-10 with IEEE CNN papers to research on possible ways to increase model performance. Yicheng will work on the ILSVRC2012 dataset and will implement results from Tianyang to train optimized deep CNN models. Su will learn CUDA programming to utilize GPU to train models and also assist Yicheng in model training. All three of us will work on the remaining part like result analyzing and thesis composing.

C. Milestones, Weekly Plans, and Deliverables

The next step is to implement our own CNN model, and will try it on HiperGator with multiple GPUs. Following the success of AlexNet and GoogleNet, we can empirically learn the structure and apply skills to our model. Several future milestones include:

1)
Repeat experiment results from other papers and rethink advantages of these methods and techniques.

2)
Choose then combine stacks of technology into our models through modifying TensorFlow CNN layers.

3)
Boost training with CUDA parallelization with reasonable assignment of tasks among kernels and multiple GPUs.

4)
Compare overall aspects of the models that we will have trained and optimize one as the final result then compose the final report.

REFERENCE

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[2] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[3] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." arXiv preprint arXiv:1512.00567 (2015).

[4] He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015).

[5] Liang, Ming, and Xiaolin Hu. "Recurrent convolutional neural network for object recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[6] van Kleef, Joshua. "Towards Human-like Performance Face Detection: A Convolutional Neural Network Approach." (2016).

[7] Li, Haoxiang, et al. "A convolutional neural network cascade for face detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[8] Çatalyürek, Üm i˙ T. V., Cevdet Aykanat, and Bora Uçar. "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe." SIAM Journal on Scientific Computing 32.2 (2010): 656-683.

[9] Howard, Andrew G. "Some improvements on deep convolutional neural network based image classification." arXiv preprint arXiv:1312.5402 (2013).

[10] Howard, Andrew G. "Some improvements on deep convolutional neural network based image classification." arXiv preprint arXiv:1312.5402 (2013).

[11] https://www.tensorflow.org/versions/r0.11/tutorials/index.html

[12] https://github.com/tensorflow/models/tree/master/inception

[13] http://image-net.org/

[14] https://aws.amazon.com/

[15] http://www.cnblogs.com/neopenx/p/4480701.html