



Midterm Presentation:

Image Classification based on CNN

T11 PangCloud: Tianyang Liu, Yicheng Wang, Su Pu



Outline

- Architecture
- CNN Algorithm
- Experiments
- Team Coordination
- Future work



Progress

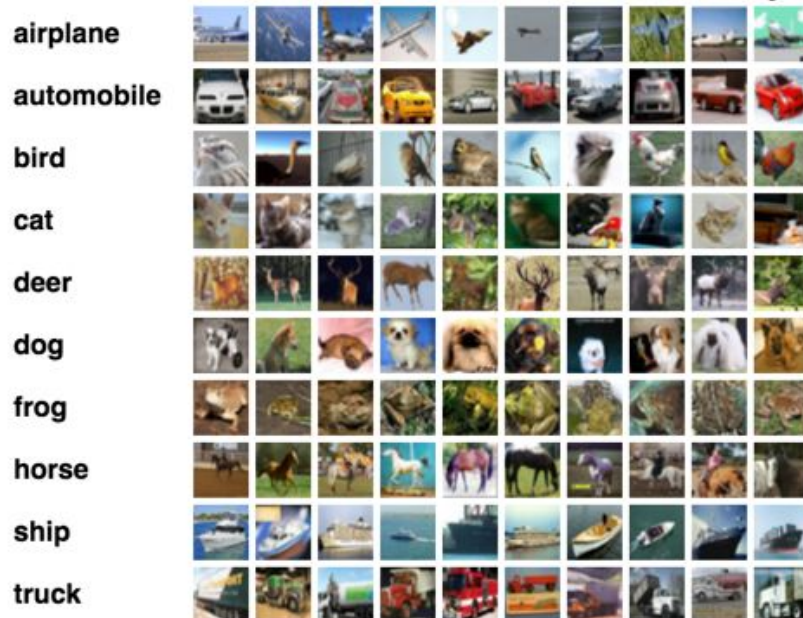
- Learn the neural network model.
- Learn the TensorFlow structure.
- Train CIFAR-10 example in TensorFlow.
- Change different CPU / GPU combinations to find the best arrangement.
- Adjust structure and parameters of CNN for CIFAR-10

Basic Component

TensorFlow: An open source machine learning framework.

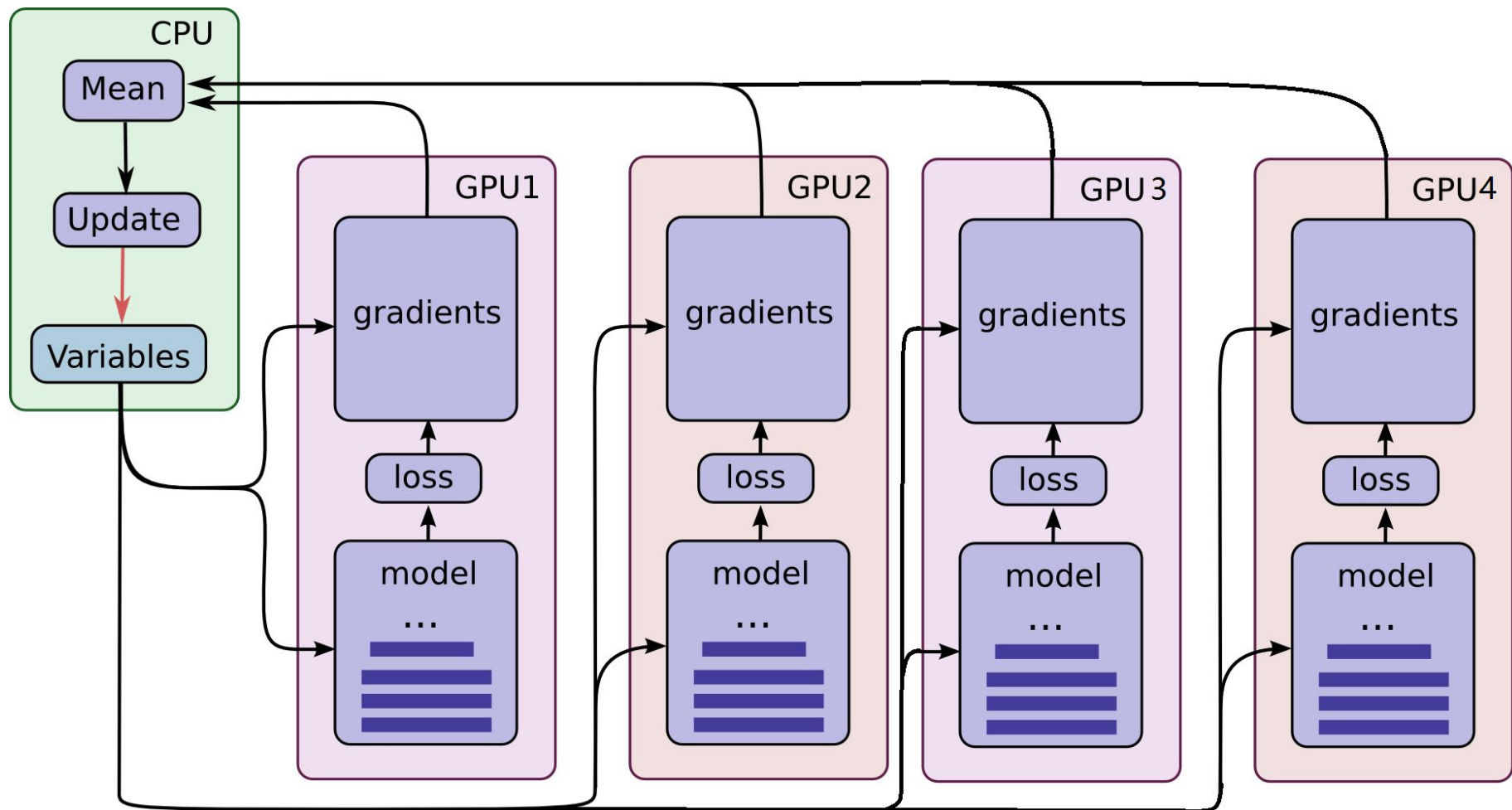
GPU / CPU: Parallelized computing platform.

CIFAR10: A smaller dataset compare to ImageNet.

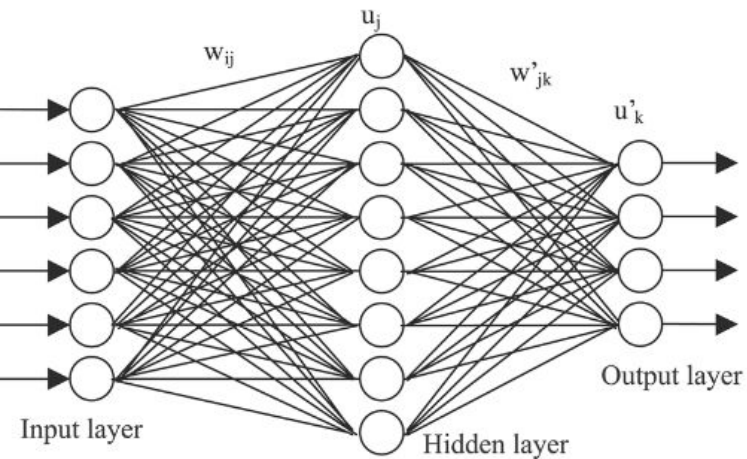


Consists of 60000
32x32 colour images
in 10 classes.

Multiple GPUs Architecture

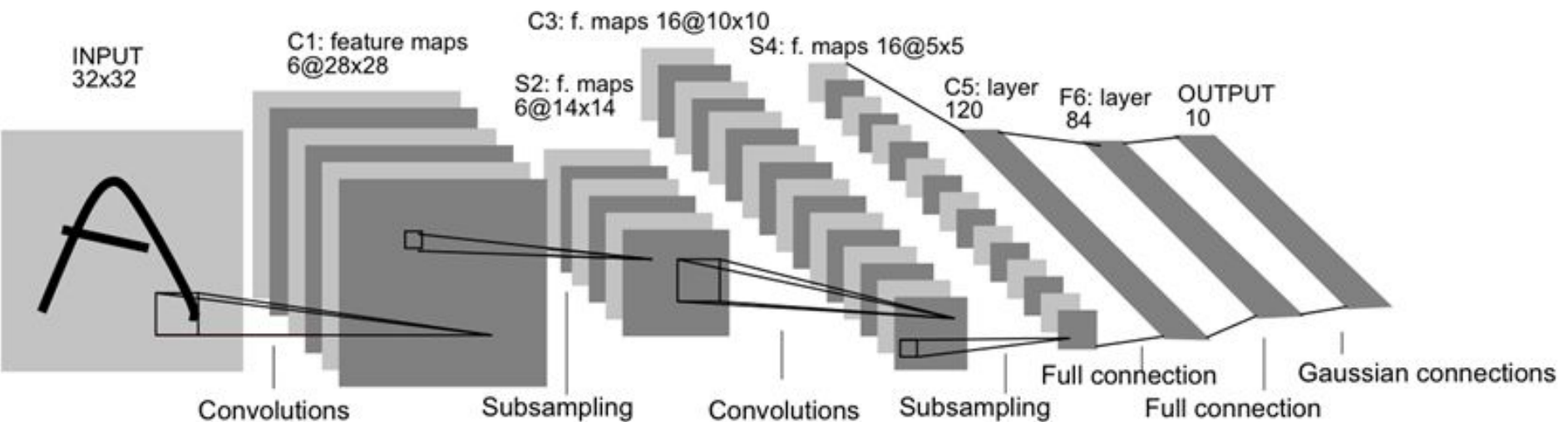


Core algorithm: CNN

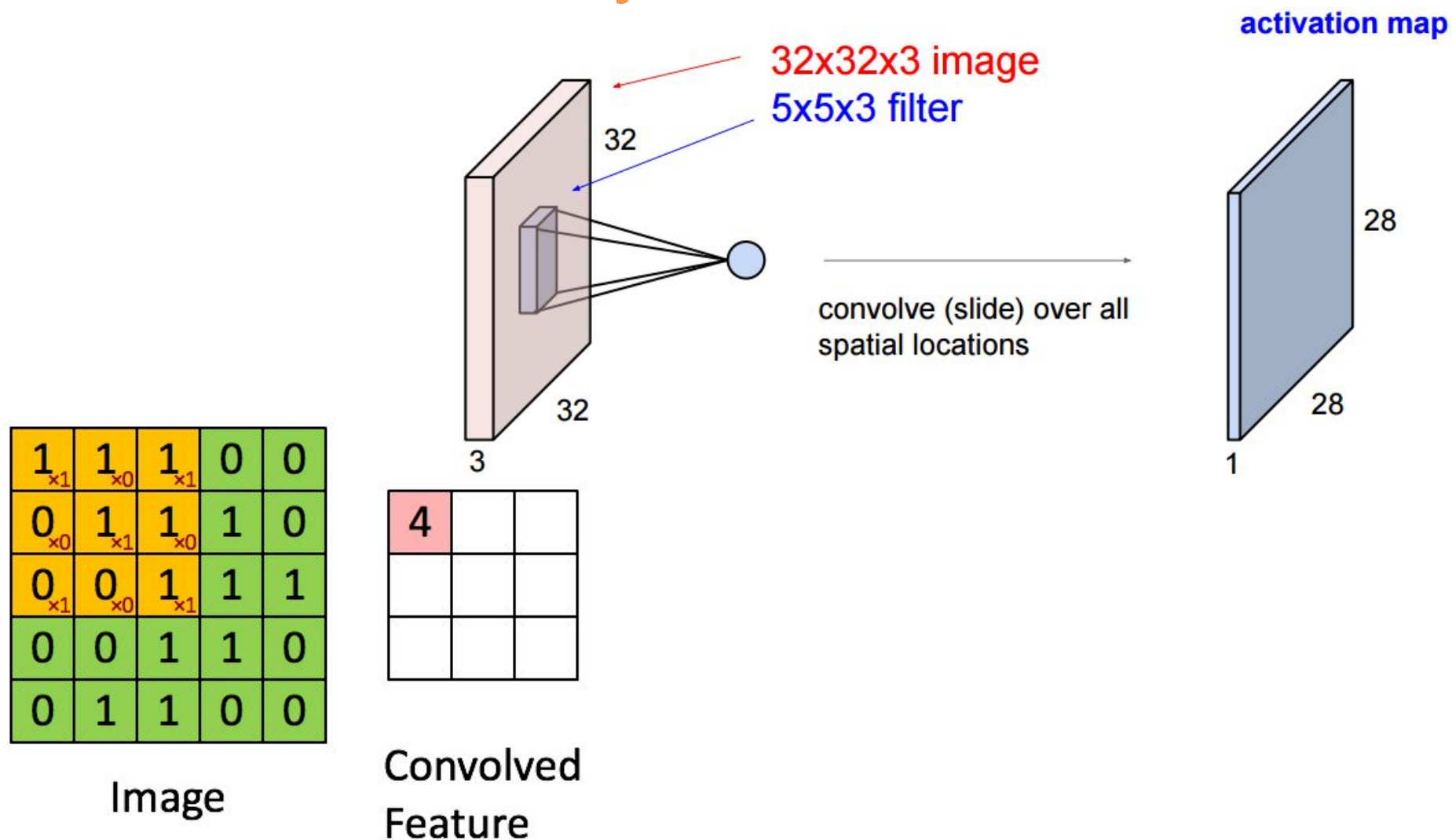


The full connection algorithm of ordinary Neural Network would lead to tremendous computation.

Input image $200 \times 200 \times 3$

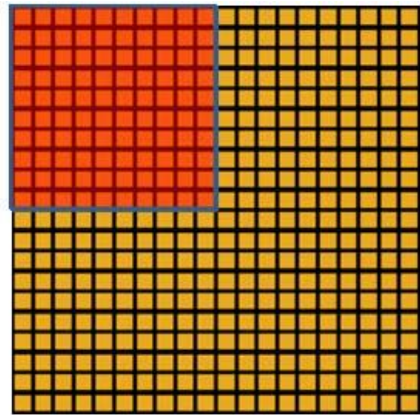


1. Convolutional layer

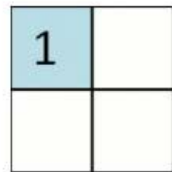


2. Pooling layer

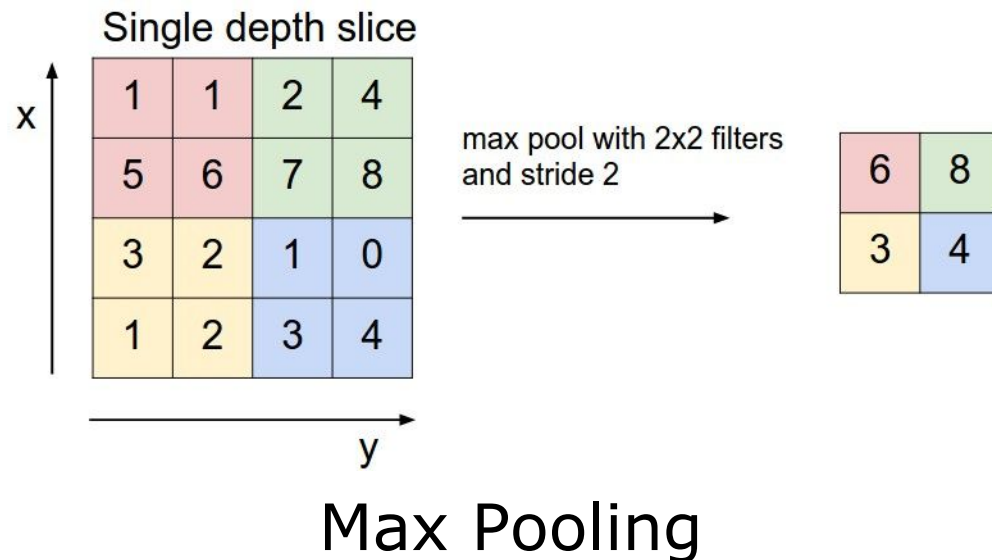
Down sample to make the representations smaller and more manageable



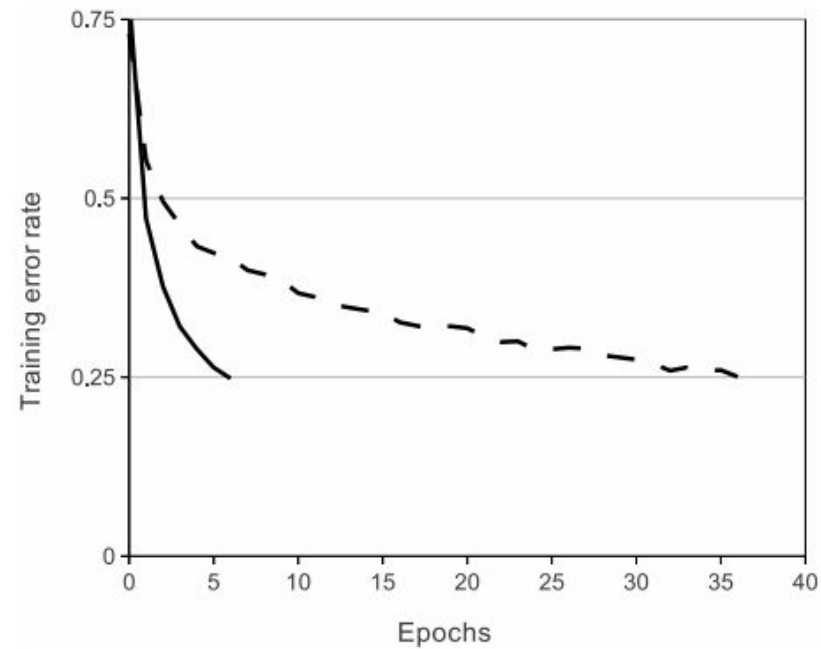
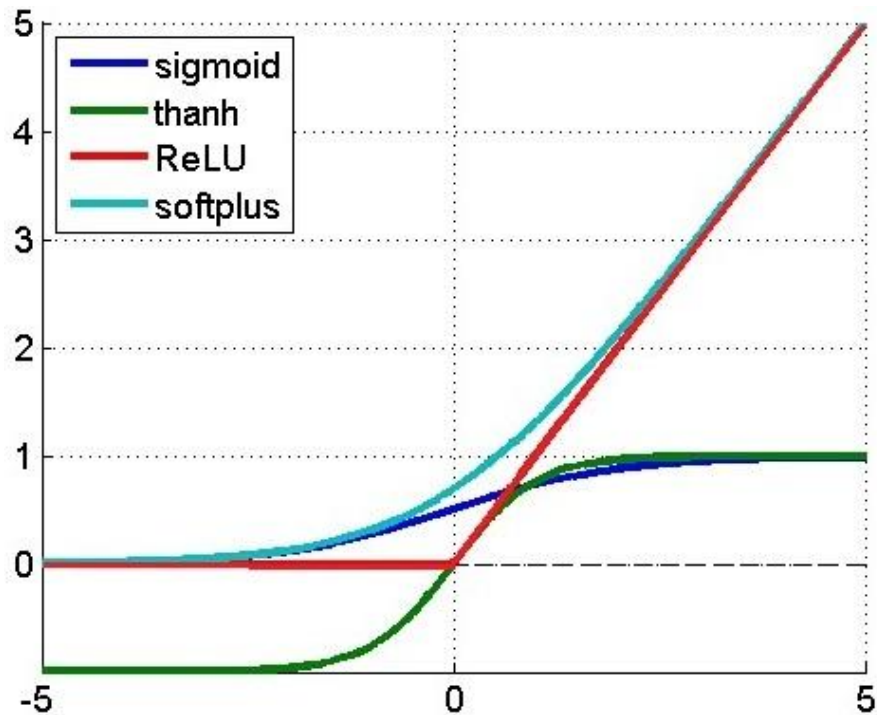
Convolved
feature



Pooled
feature

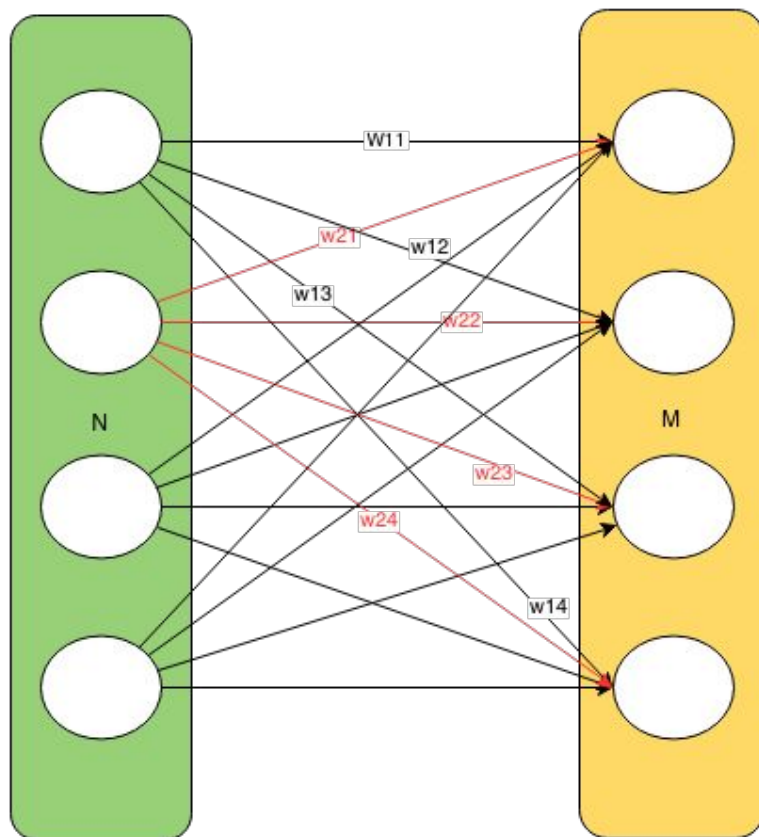


3. ReLU



- non-saturating nonlinearity
- Faster learning
- preventing overfitting

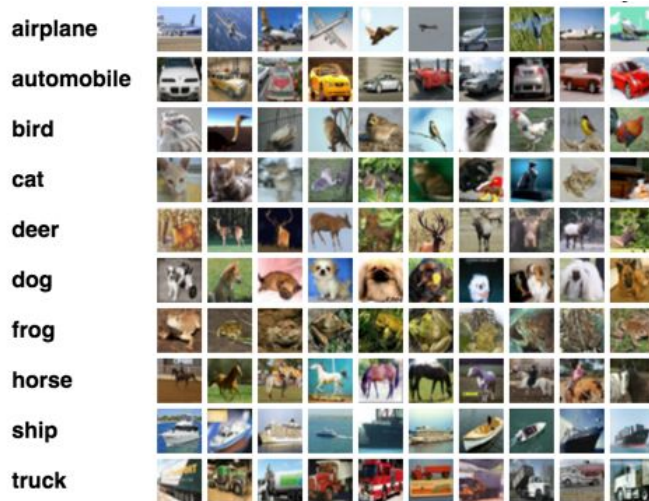
4. Fully-Connected layer



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

Implementation

Input:



Training images: 50000
(10000 for validation)
Test images: 10000
Class: 10
batch size: 128

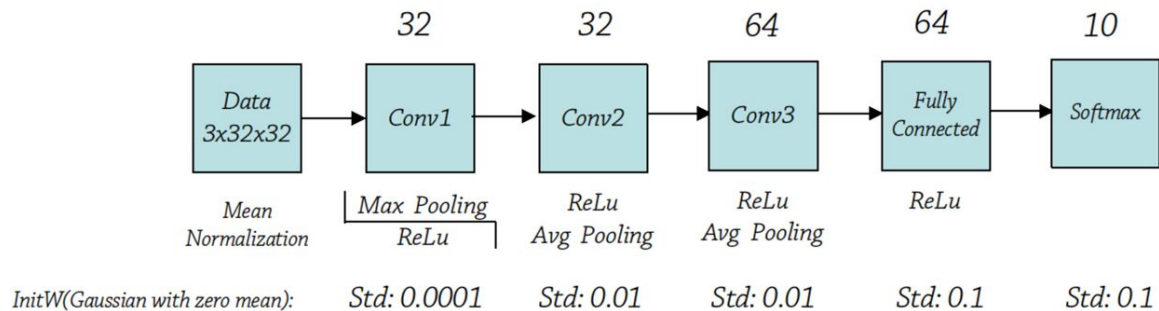
Code:

```
# 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.""")
```

Implementation

Training:

Cifar-10 Fast Model(5 epochs with 25% Validation Error Rate)



Code: Generate several Convolutional layers

```
# conv1
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)
    _activation_summary(conv1)

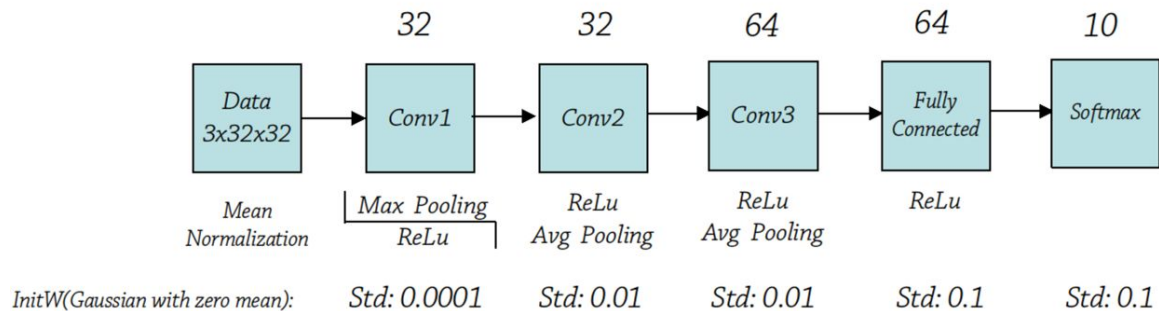
# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool1')

# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm1')
```


Implementation

Training:

Cifar-10 Fast Model(5 epochs with 25% Validation Error Rate)



Code:

The convolution result go through SoftMax first, and then go through fully connected neuron network

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

Calculate the loss and gradient and modify the weight and bias.

```
# Compute the moving average of all individual losses and the total loss.
loss_averages = tf.train.ExponentialMovingAverage(0.9, name='avg')
# Compute gradients.
with tf.control_dependencies([loss_averages_op]):
    opt = tf.train.GradientDescentOptimizer(lr)
    grads = opt.compute_gradients(total_loss)
# Apply gradients.
apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)
```


Implementation

Evaluation:

The evaluation runs after every `max_step`'s steps.

```
tf.app.flags.DEFINE_integer('max_steps', 1000000,  
                            """"Number of batches to run.""")
```

Track the neuron network information from the check point.

```
saver.restore(sess, ckpt.model_checkpoint_path)  
# Assuming model_checkpoint_path looks something like:  
# /my-favorite-path/cifar10_train/model.ckpt-0,  
# extract global_step from it.  
global_step = ckpt.model_checkpoint_path.split('/')[-1].split('-')[-1]
```

Run the evaluation

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

Implementation

Tests on CPUs:

We choose google cloud platform to do the experiment on multiple CPUs.(Yes, It is cheaper!)

Create time
Oct 3, 2016, 10:58:20 PM

Tags

http-server x https-server x

Machine type

16 vCPUs

60 GB memory

Customize

CPU platform

Unknown CPU Platform

Zone

us-east1-d

External IP

Ephemeral

Internal IP

10.142.0.2

IP forwarding

off

Boot disk and local disks

Name	Size (GB)	Type	Mode
elastictest1	10	SSD persistent disk	Boot, read/write

☒ Delete boot disk when instance is deleted

Additional disks (Optional)

We changed the CPU number to test difference of the training speed.

Implementation

Tests on GPUs:

GPU is mostly design for calculation of graphics, it can be paralleled and calculates faster than CPUs.

The screenshot displays the AWS Management Console interface for an EC2 instance. At the top, there are buttons for 'Launch Instance', 'Connect', and 'Actions'. Below these is a search bar and a table of instances. The table has columns for Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, and Public DNS. Two instances are listed, both with a state of 'stopped'. The second instance, 'i-08e7d2897bab9b90b', is selected. Below the table, the instance details are shown for 'i-08e7d2897bab9b90b' with a private IP of '172.31.53.245'. The 'Description' tab is active, showing a list of instance attributes. The instance is a 'g2.2xlarge' type in the 'us-east-1b' availability zone, running the 'ubuntu14.04-cuda7.5-tensorflow0.9' AMI. It has a public IP of '172.31.53.245' and is associated with the 'launch-wizard-1' security group.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
	i-0417b7388a2a0c19d	g2.8xlarge	us-east-1b	stopped		None	
	i-08e7d2897bab9b90b	g2.2xlarge	us-east-1b	stopped		None	

Instance: **i-08e7d2897bab9b90b** Private IP: 172.31.53.245

Description | Status Checks | Monitoring | Tags

Instance ID	i-08e7d2897bab9b90b	Public DNS	-
Instance state	stopped	Public IP	-
Instance type	g2.2xlarge	Elastic IPs	-
Private DNS	ip-172-31-53-245.ec2.internal	Availability zone	us-east-1b
Private IPs	172.31.53.245	Security groups	launch-wizard-1. view rules
Secondary private IPs	-	Scheduled events	-
VPC ID	vpc-c65765a1	AMI ID	ubuntu14.04-cuda7.5-tensorflow0.9 (ami-81960a96)
Subnet ID	subnet-934988be	Platform	-
Network interfaces	eth0	IAM role	-
Source/dest. check	True	Key pair name	dotaliu
EBS-optimized	False	Owner	772185490472
		Launch time	October 7, 2016 at 12:49:53 AM UTC-4 (62 hours)



Experiment setup

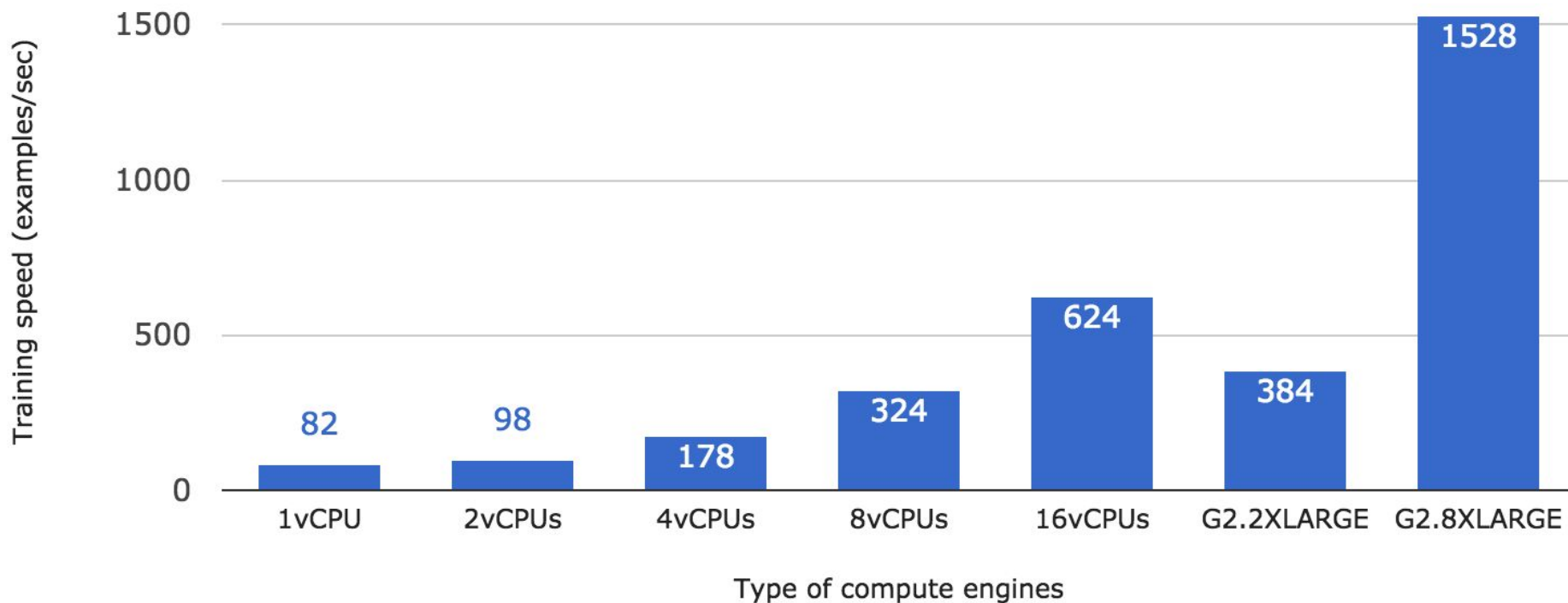
Hardware: AWS GPU instances / gcloud virtual CPUs /
Nvidia Tesla K80 GPUs in UF HiperGator 2.0

Software: Ubuntu 14.04 LTS / CUDA 7.5 / cuDNN v5 /
TensorFlow 0.11

Metrics: speed / error rate / parameter count / epoch

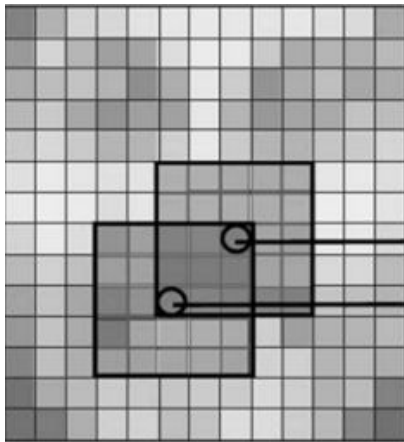
Experiment 1

Speed of different platforms: CPU vs GPU



Experiment 2

Model error rate of overlapping versus nonoverlapping stride of pooling layer scheme.



```
# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                        padding='SAME', name='pool1')
```

Change the strides of the pooling function and test on both overlapping and nonoverlapping pooling scheme.

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%



Team Coordination

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



Future Experiments

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.

Thank you!

T11 PangCloud

Image classification based on CNN

