Deep neural network model compression

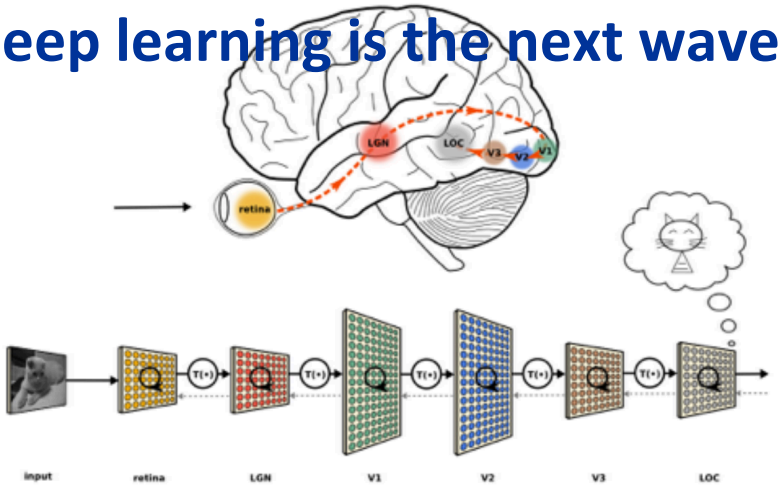
* Introduction and Background
* Deep Neural Network Model Compression
* Result shows on Compression AlexNet and VGGNet with Cifar10 and ImageNet dataset.
* My compression package, applying to RNN and SLTM.

• Master student in UIUC, Computer Science.
• Interest Area: deep learning model compression and Software and hardware acceleration, to make inference more efficient for deployment.

This Talk: Deep neural network model compression

* Introduction and Background
* **Deep Neural Network Model Compression**
* Result shows on Compression AlexNet and VGGNet with Cifar10 and ImageNet dataset.

Deep learning is the next wave of AI



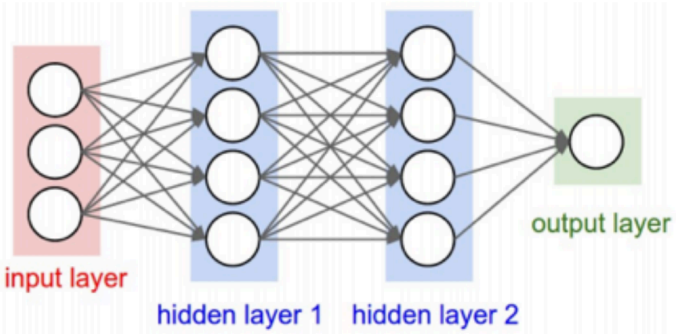
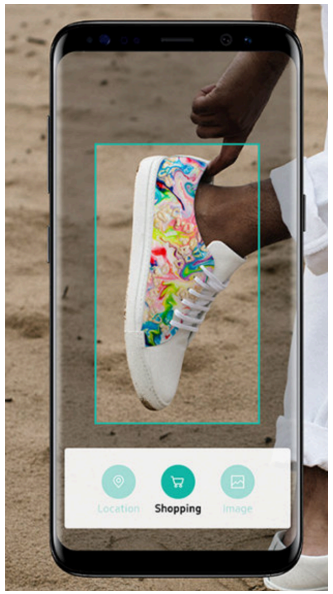
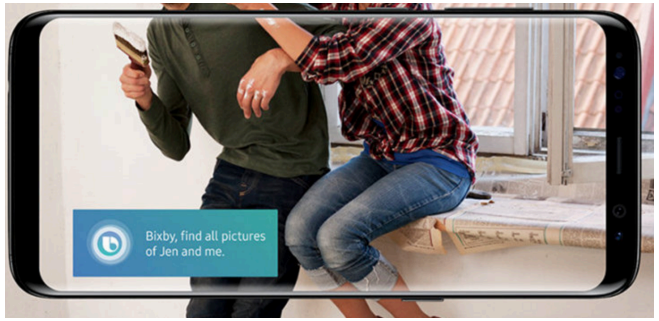


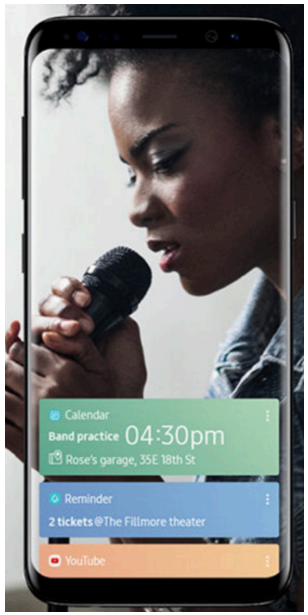
Image Recognition



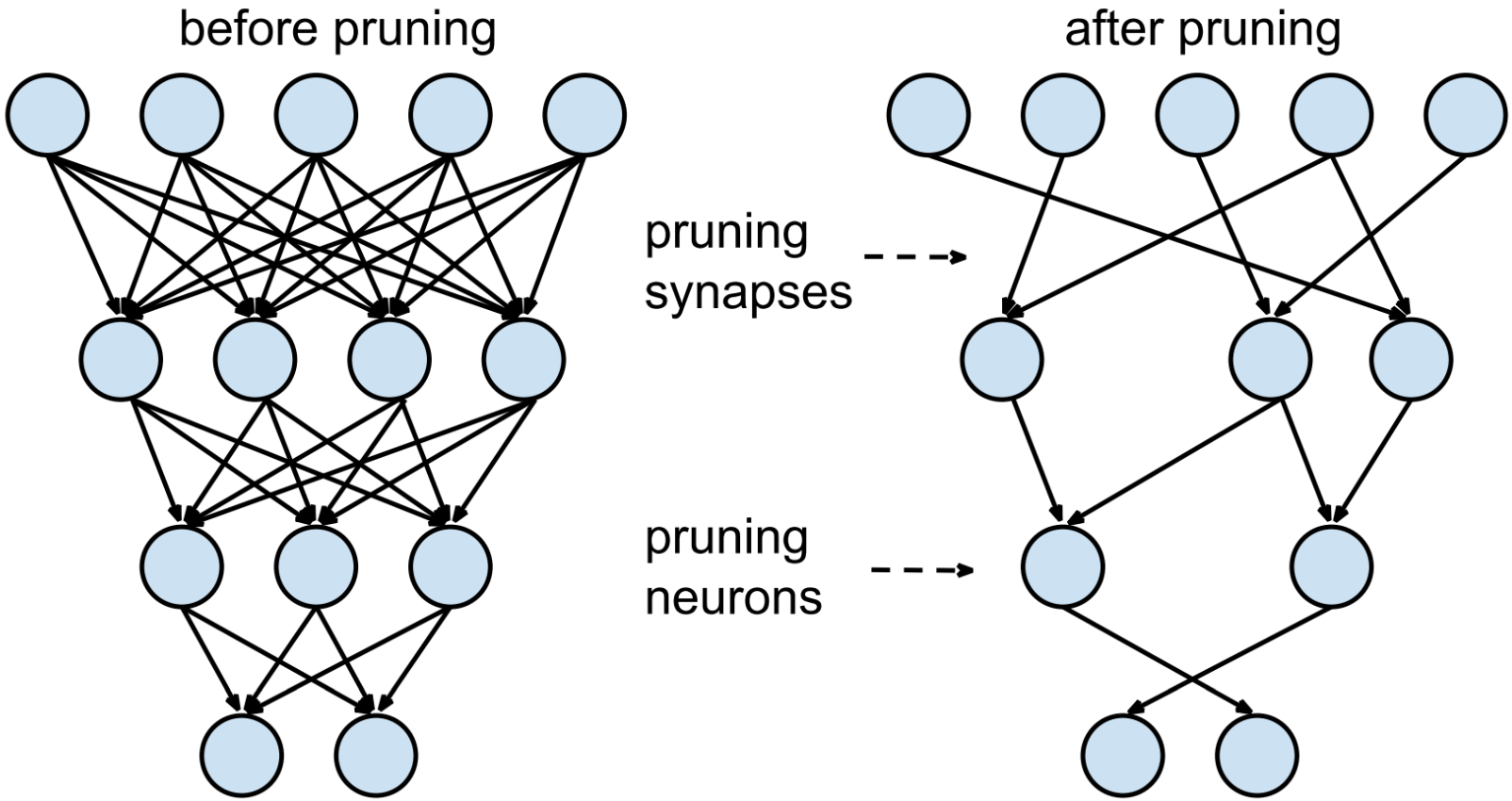
Speech Recognition



Natural Language Processing



Pruning


before pruning
after pruning
pruning
synapses
pruning
neurons

My proposed Method： Dynamic threshold selection achieved substantially better results.
That is done by iteratively finding for each layer its achievable sparsity instead of selecting a single threshold for the whole network and then applying dense sparse retraining while pruning.

Deep Compression- Pruning

- AlexNet12: 7.14×, 240MB => 27MB
- With no loss of accuracy on ImageNet12

Pruning + weights sharing +

My proposed Method: Dynamic threshold selection achieved substantially better results.

That is done by iteratively finding for each layer its achievable sparsity instead of selecting a single threshold for the whole network and then applying dense sparse retraining while pruning.

Algorithm 1: Determine achievable sparsity in a layer

Input: error_tolerance, $Starting_Sparsity_L$ for each layer defaults to 100, pre-trained network

Output: Achieved sparsity in each layer

```
foreach trainable layer L from n to 1 do
    Sparsity_L ← Starting_Sparsity_L ;
    Prune the weights in layer L up to defined sparsity Sparsity_L ;
    t ← 0 ;
    current_error_t ← computed error with the modified network ;
    while current_error_t > error_tolerance do
        Train the model ;
        Prune again up to same level, Sparsity_L ;
        t ← t + 1 ;
        current_error_t ← current error of the network ;
        if current_error_t < current_error_{t-1} then
            Decrease the sparsity, Sparsity_L, by 1% ;
        end
    end
    Achieved_Sparsity_L ← Sparsity_L ;
end
return Achieved_Sparsity
```

Deep Compression- Pruning

Algorithm 2: Dynamic threshold calculation and pruning of the network

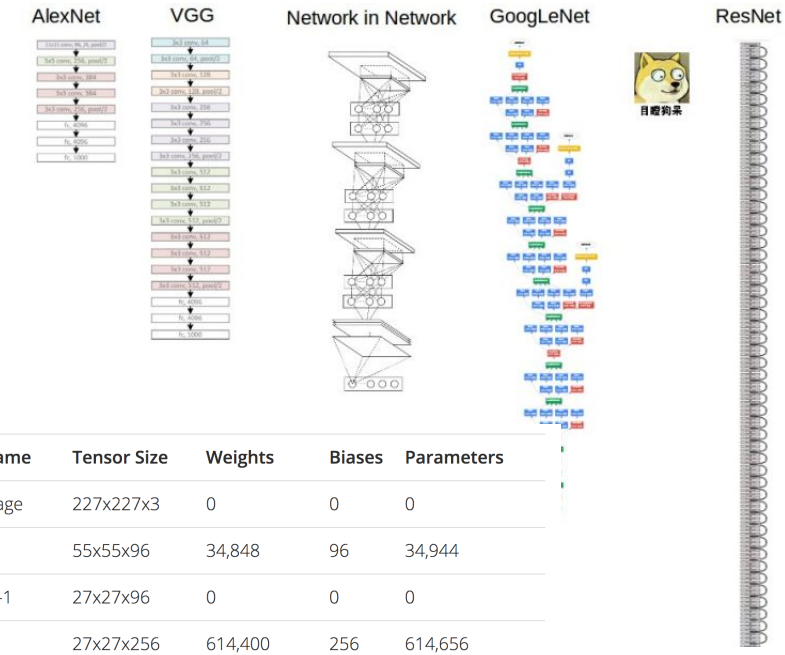
Input: error_tolerance, $Achieved_Sparsity$ for each layer, $Extra_Sparsity_L$ for each layer, pre-trained network

Output: Sparse network

```
Sparsity_L ← Achieved_Sparsity_L + Extra_Sparsity_L ;
Sparse phase(for each layer simultaneously):
Get weights W_L for each layer separately ;
Calculate threshold, λ_L needed to obtain Sparsity_L ;
Prune i_th weight, W_L^ith ← 0 if W_L^ith < λ_L ;
Dense phase ;
t ← 0 ;
current_error_t ← Evaluate the network ;
do
    do
        Retrain the network ;
        Again prune the network using, λ_L ;
        current_error_t ← Evaluate the network ;
        t ← t + 1 ;
    while current_error_{t-1} > current_error_t;
    Decrease the Sparsity_L in each layer by 1% ;
    Recalculate λ_L ;
while current_error > error_tolerance;
return pruned network
```

Results- Data Visualization by TensorBoard- prune away the covolution layers by 7.14 times on AlexNet, Overall for 35-50X on AlexNet

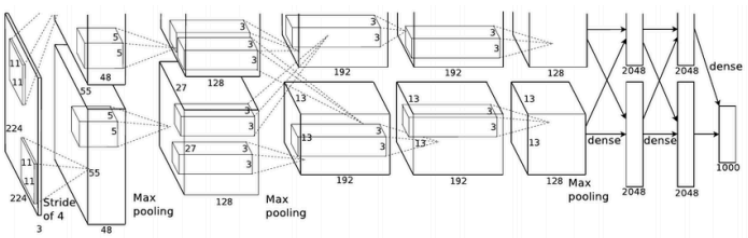
- ImageNet:
- The ImageNet project is a large visual database designed for use in visual object recognition software research. Over 14 million[1][2] URLs of images have been hand-annotated by ImageNet to indicate what objects are pictured; in at least one million of the images, bounding boxes are also provided.[3] ImageNet contains over 20 thousand categories;[2] a typical category, such as "balloon" or "strawberry", contains several hundred images.[4] The database of annotations of third-party image URL's is freely available directly from ImageNet; however, the actual images are not owned by ImageNet.[5]



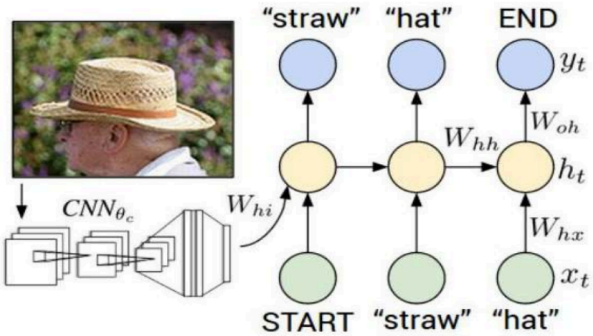
- One of my Network AlexNet matched ImagNet contains 60 millions parameters with 238MB

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

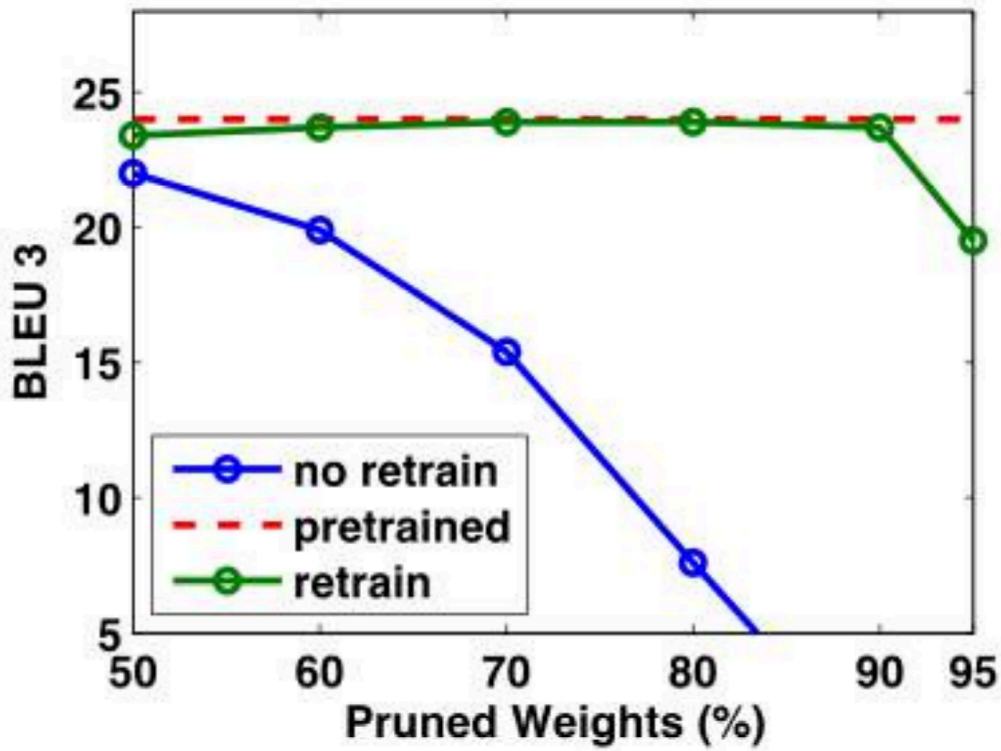
AlexNet (Krizhevsky, Sutskever, Hinton 2012)



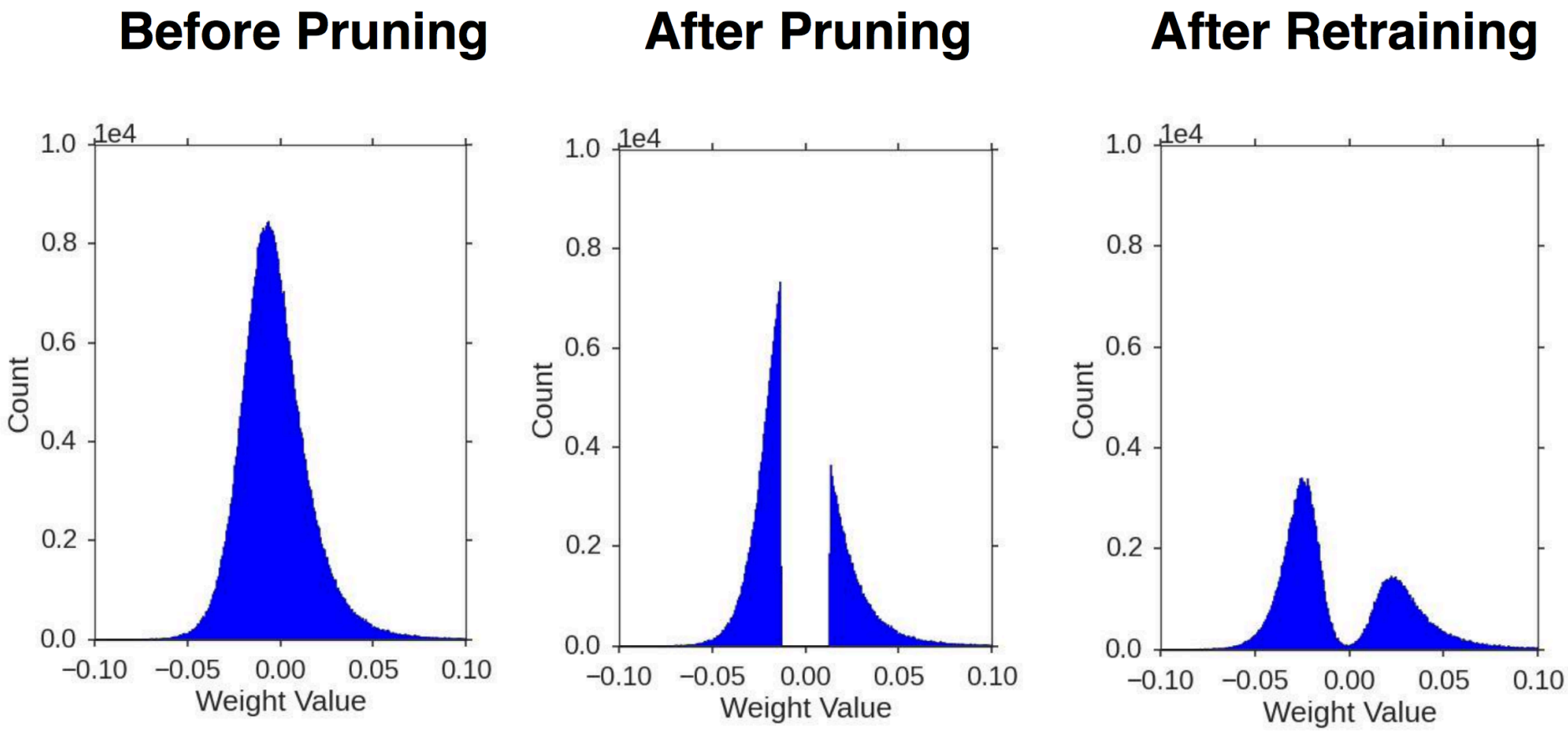
Pruning RNN and LSTM



*Karpathy et al "Deep Visual-Semantic Alignments for Generating Image Descriptions"



- This work contributes to based on the previous research on compression of deep networks performing classification. It is follow-up by Song Han Assistant Professor at MIT. Previously CVA group, Stanford University.
- Moreover I designed and implemented a two stage deep compression pipeline of pruning to improve their work including getting achieved sparsity and continually pruning by adding extra sparsity. Using different sparsity levels, I calculate different thresholds in each layer to perform pruning.


Before Pruning
After Pruning
After Retraining
1e4
1.0
0.8
0.6
0.4
0.2
0.0
Count
−0.10
−0.05
0.00
0.05
0.10
Weight Value

Details

pretrain.py | determine_sparsity.py | pruning.py | alexnet.py

```python
import ...

train_cifar10 = cifar10(data_dir='cifar-10-batches-py/', batch_size=100, subset='train', use_distorti
test_cifar10 = cifar10(data_dir='cifar-10-batches-py/', batch_size=100, subset='test', use_distortion=
error_tolerance = 0.30
decrease_rate = 0.01
epoch = 2
display_step = 100
sparsity = {
    'w_conv1': 1.0,
    'w_conv2': 1.0,
    'w_conv3': 1.0,
    'w_conv4': 1.0,
    'w_conv5': 1.0,
    'w_fc1': 1.0,
    'w_fc2': 1.0,
    'w_fc3': 1.0
    }

def get_threshold(sess, weight,
    if sparsity > 1. or sparsit
        raise ValueError('Spars
    if sparsity == 1.:
        return np.inf
    if sparsity == 0.:
        return -np.inf
    weight_arr = sess.run(weigh
    flat = np.reshape(weight_ar
    flat_list = sorted(map(abs,
    return flat_list[int(len(fl


def prune(sess, weight, thresho
    weight_arr = sess.run(weigh
```

```python
def prune(sess, weight, threshold):#session run--> weights的 real value
    weight_arr = sess.run(weight)
    under_threshold = abs(weight_arr) < threshold
    weight_arr[under_threshold] = 0
    return weight_arr

with tf.Graph().as_default() as determine_sparsity:

    x = tf.placeholder(tf.float32, [None, 32, 32, 3], name="x")
    y_ = tf.placeholder(tf.float32, [None, 10], name="y_")
    keep_prob = tf.placeholder(tf.float32, name="keep_prob")
```

Algorithm 1: Determine achievable sparsity in a layer

Input: error_tolerance, $Starting_Sparsity_L$ for each layer defaults to 100, pre-trained network

Output: Achieved sparsity in each layer

foreach *trainable layer* L *from* n *to* 1 **do**
  $Sparsity_L \leftarrow Starting_Sparsity_L$;
  Prune the weights in layer L up to defined sparsity $Sparsity_L$;
  $t \leftarrow 0$;
  $current_error_t \leftarrow$ computed error with the modified network ;
  while $current_error_t > error_tolerance$ **do**
    Train the model ;
    Prune again up to same level, $Sparsity_L$;
    $t \leftarrow t + 1$;
    $current_error_t \leftarrow$ current error of the network ;
    if $current_error_t < current_error_{t-1}$ **then**
      Decrease the sparsity, $Sparsity_L$, by 1% ;
    end
  end
  $Achieved_Sparsity_L \leftarrow Sparsity_L$;
end
return $Achieved_Sparsity$

pretrain.py | determine_sparsity.py

```python
        return 1. - test_acc

    sess.run(tf.global_variables

    saver.restore(sess, './model

    determine_sparsity.finalize(

    for layer_name in list(spars
        print('Start pruning Lay
        f.write('Start pruning L
        threshold = get_thresho
        sparse_weight = prune(se
        weights[layer_name].load

        t = 0
        error_before_retrain = g
        print('Iter %d, error ra
        f.write('Iter %d, error rate %g\n' % (t, error_before_retrain))

        while error_before_retrain > error_tolerance:
            for _ in range(1, train_cifar10.num_iter_per_epoch * epoch + 1):
                batch = train_cifar10.next_batch()
                if _ % display_step == 0:
                    train_acc = sess.run(accuracy, feed_dict={x:batch[0], y_:batch[1], keep_prob: 1.0})
                    print("Re-train step %d, error rate %g" % (__, 1. - train_acc))
                    f.write("Re-train step %d, error rate %g\n" % (__, 1. - train_acc))
                sess.run(train_op, feed_dict={x:batch[0], y_:batch[1], keep_prob:0.5})
    #
            threshold = get_threshold(sess, weights[layer_name], sparsity[layer_name])
            sparse_weight = prune(sess, weights[layer_name], threshold)
            weights[layer_name].load(sparse_weight)

            t += 1
```

```python
    print('Done')
    f.write('Done\n')

    for layer_name in sparsity.keys():
        print('Achieved sparsity in Layer %s is %g' % (layer_name, sparsity[layer_name]))
        f.write('Achieved sparsity in Layer %s is %g\n' % (layer_name, sparsity[layer_name]))
    f.close()
```

datagenerator.py | determine_sparsity.py | pruning.py

```python
import os
import pickle
import numpy as np
import tensorflow as tf

from alexnet import AlexNet
from datagenerator import ImageDataGenerator
from datetime import datetime

"""
Configuration Part.
"""

# Path to the textfiles for the trainin
train_file = './train.txt'
valid_file = './valid.txt'

# Learning params
learning_rate = 0.001
batch_size = 128

# Network params
num_epochs = 2
train_steps = 2000
dropout_rate = 0.5
num_classes = 1000
error_tolerance = 0.44
decrease_rate = 0.01
extra_sparsiy = 0.05

weights = dict()
threshold = dict()

with open('achieved_sparsity.pickle', '
```

datagenerator.py | determine_sparsity.py | pruning.py

```python
with open('achieved_sparsity.pickle', 'rb') as f:
    sparsity = pickle.load(f)

for layer_name in sparsity.keys():
    threshold[layer_name] = 0
    sparsity[layer_name] += extra_sparsiy
    if sparsity[layer_name] > 1.:
        sparsity[layer_name] = 1.

# How often we want to write the tf.summary data to disk
display_step = 200

# Path for tf.summary.FileWriter and to store model checkpoints
```

Algorithm 2: Dynamic threshold calculation and pruning of the network

Input: error_tolerance, $Achieved_Sparsity$ for each layer, $Extra_Sparsity_L$ for each layer, pre-trained network

Output: Sparse network

$Sparsity_L \leftarrow Achieved_Sparsity_L + Extra_Sparsity_L$;
Sparse phase(for each layer simultaneously):
Get weights W_L for each layer separately ;
Calculate threshold, λ_L needed to obtain $Sparsity_L$;
Prune i_{th} weight, $W_L^{ith} \leftarrow 0$ if $W_L^{ith} < \lambda_L$;
Dense phase ;
$t \leftarrow 0$;
$current_error_t \leftarrow$ Evaluate the network ;
do
  do
    Retrain the network ;
    Again prune the network using, λ_L ;
    $current_error_t \leftarrow$ Evaluate the network ;
    $t \leftarrow t + 1$;
  while $current_error_{t-1} > current_error_t$;
  Decrease the $Sparsity_L$ in each layer by 1% ;
  Recalculate λ_L ;
while $current_error > error_tolerance$;
return *pruned network*

```python
    valid_iterator = tf.data.Iterator.

    valid_next_batch = valid_iterator.


# Ops for initializing the two differe
training_init_op = train_iterator.make
validation_init_op = valid_iterator.ma

# TF placeholder for graph input and o
x = tf.placeholder(tf.float32, [batch_
y = tf.placeholder(tf.float32, [batch_
keep_prob = tf.placeholder(tf.float32)

# Initialize model
model = AlexNet(x, keep_prob, num_clas

# Load the pretrained weights into the non-trainable layer
load_op = model.load_initial_weights_ops()

# Link variable to model output
logits = model.fc8

# Op for calculating the loss
with tf.name_scope("cross_ent"):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))

var_list = [v for v in tf.trainable_variables()]
for var in var_list:
    if 'weights' in var.name.split('/')[1]:
        weights[var.name.split('/')[0]] = var

# Train op
```

```python
# Evaluation op: Accuracy of the model
with tf.name_scope("accuracy"):
    correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

with tf.name_scope("error_rate"):
    error_rate = 1. - accuracy

train_merged_summary = tf.summary.merge([tf.summary.scalar('accuracy', accuracy), tf.summary.scalar('error_rate', er

val_acc, val_acc_op = tf.metrics.accuracy(labels=tf.argmax(y, 1), predictions=tf.argmax(score, 1))
val_error = 1.0 - val_acc
running_vars = tf.get_collection(tf.GraphKeys.LOCAL_VARIABLES)
running_vars_initializer = tf.variables_initializer(var_list=running_vars)

valid_merged_summary = tf.summary.merge([tf.summary.scalar('accuracy', val_acc), tf.summary.scalar('error_rate', val
```

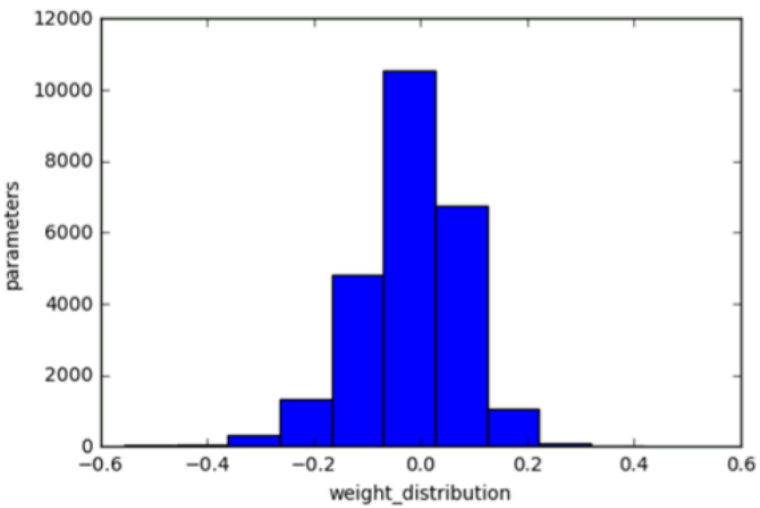
datagenerator.py | determine_sparsity.py | pruning.py

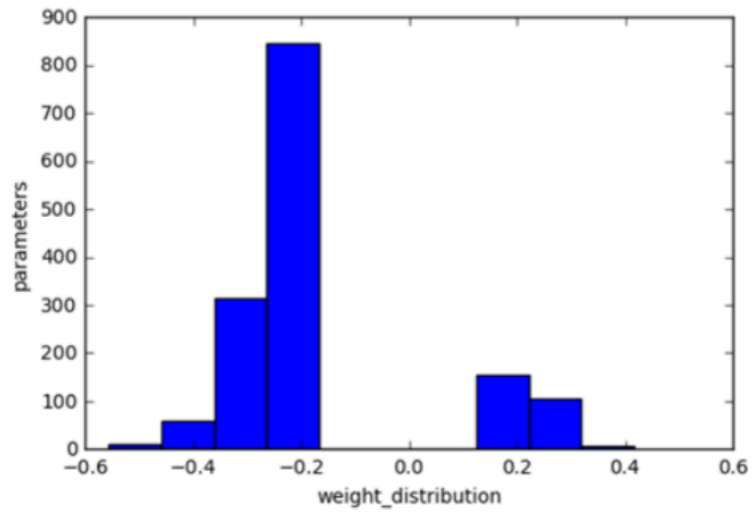
```python
train_batches_per_epoch = tr_data.data_size // batch_size
val_batches_per_epoch = val_data.data_size // batch_size

# Start Tensorflow session
with tf.Session() as sess:

    # Initialize all variables
    sess.run(tf.global_variables_initializer())
    train_writer.add_graph(sess.graph)
    sess.run(load_op)
    sess.run(training_init_op)

    f = open('./log/log_pruning.txt', 'w')

    def log_print(s):
        print(s)
        f.write(s + '\n')

    def get_threshold(sess, weight, sparsity):
        if sparsity > 1. or sparsity < 0.:
            raise ValueError('Sparsity out of range')
        if sparsity == 1.:
            return np.inf
        if sparsity == 0.:
            return -np.inf
        weight_arr = sess.run(weight)
        flat = np.reshape(weight_arr, [-1])
        flat_list = sorted(map(abs,flat))
        return flat_list[int(len(flat_list) * sparsity)]

    def prune(sess, weight, threshold):
        weight_arr = sess.run(weight)
        under_threshold = abs(weight_arr) < threshold
        weight_arr[under_threshold] = 0
```

datagenerator.py | determine_sparsity.py | pruning.py

```python
    prune_network()

    current_error = get_error()

    while True:
        while True:
            for _ in range(1, train_steps + 1):

                try:
                    img_batch, label_batch = sess.run(train_next_batch)
                    sess.run(train_op, feed_dict={x: img_batch, y: label_batch, keep_prob: dropout_rate})
                except:
                    sess.run(training_init_op)
                    img_batch, label_batch = sess.run(train_next_batch)
                    sess.run(train_op, feed_dict={x: img_batch, y: label_batch, keep_prob: dropout_rate})

                if _ % display_step == 0:
                    summary, train_acc = sess.run([train_merged_summary, accuracy], feed_dict={x: img_batch, y :label_
                    train_writer.add_summary(summary)
                    log_print("Re-train step %d, error rate %g" % (_, 1. - train_acc))

            log_print('Pruning network.')
            prune_network()

            previous_error = current_error
            current_error = get_error()
            t += 1

            if current_error >= previous_error:
                log_print('After pruning, previous error %g is no more than current error %g. Evaluating pruned networ
                break
            else:
                log_print('After pruning, previous error %g is more than current error %g. Retraining network.' % (prev
                log_print('-------------------- Iter %d --------------------' % t)
```

Weight distribution of second convolutional layer of LeNet before pruning (in the left) and after pruning (in the right)





Comparison of our pruning results on LeNet-5 with that of Han et al. (2015)

Model	Layer	Percentage of remaining parameters [ours]	Number of remaining parameters [ours]	Percentage of remaining parameters (Han et al., 2015)	Number of remaining parameters (Han et al., 2015)
LeNet-5	F2	5%	250	19 %	950
	F1	6%	24000	8 %	32000
	C2	15%	3750	12 %	3000
	C1	50%	90	66 %	330
Total		≈ 6.5%	28090	≈ 8 %	36280
Accuracy		99.27%		99.26 %	
Storage Requirement		6 KB		44 KB	