上海科技大学

信息科学与技术学院

科学研究基本训练总结报告

学　号 **16067192**

姓　名 叶菁菁

学科专业 电子信息

入学年月 **2015**

题　目 **A New CNN Pruning Method and Problems**

成绩评定 **□√ A □ B □ C □ F**

导师签字

2018 年 10 月 30

A New CNN Pruning Method and Problems

Jingjing Ye 16067192 Advisor:Yajun Ha

【**Abstract**】 Pruning neural networks is an old idea going back to 1990 (with Yan Lecun's optimal brain damage work) and before. The idea is that among the many parameters in the network, some are redundant and don't contribute a lot to the output. There are some ways to imitation, we introduce "deep compression", a three stage pipeline: pruning, rained quantization and Huffman coding, pruning method based on k-means++ algorithm, weighting pruning and so on. We find Affine Arithmetic which can help us find the weight not so important. We apply it to the AlexNet and last get benchmark loss 0.92719088, benchmark accuracy 0.77584; coeff*weight loss 1.84362456, coeff*weight accuracy 0.61764, pruning rate 50%.

【**Key Words**】 CNN Pruning; AlexNet; Affine Arithmetic

1 INTRODUCTION

Inspired by Hubel and Wiesel's electrophysiological studies on cat visual cortex, a convolutional neural network (CNN) was proposed. Yann Lecun first used CNN for handwritten digit recognition and maintained its dominance in this issue. In recent years, convolutional neural networks continue to exert force in multiple directions, and have made breakthroughs in speech recognition, face recognition, general object recognition, motion analysis, natural language processing, and even brain wave analysis.

But first, it takes very long time to train a model and use a lot of sources. For example, the classic deep convolutional network VGG-16 has a model size of 528M, making it difficult for users to download such a large model to a mobile phone or other terminal device. At the same time, on a typical smartphone, the VGG-16 recognizes an image for up to 3000+ms, which is unacceptable for most users. Secondly, Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources.

Getting faster/smaller networks is important for running these deep learning networks on mobile devices.

Low bit CNN is waiting for us to search.

2 CNN

Use one picture (figure 1) to have a lot about CNN

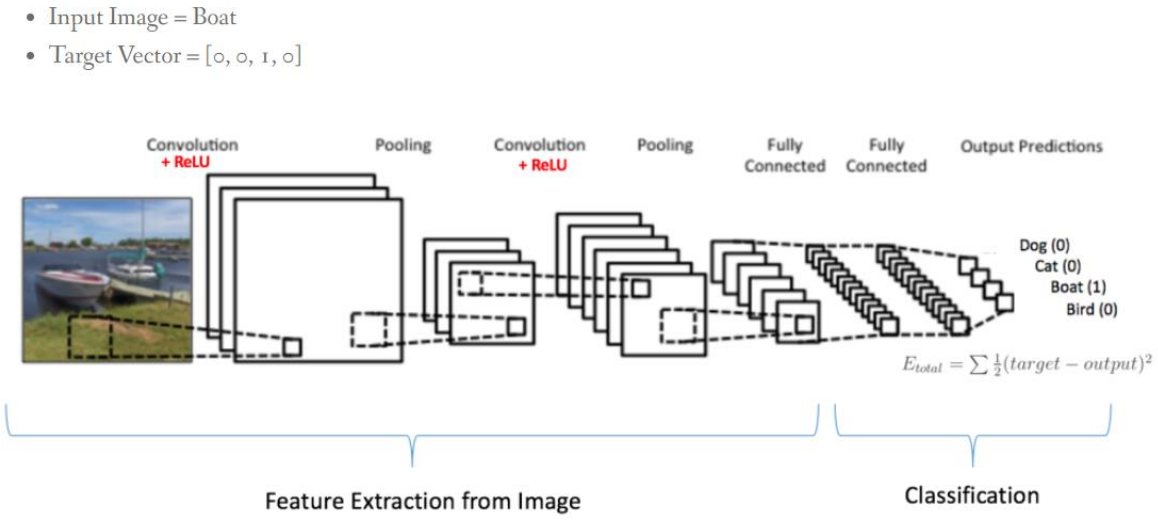

Figure 1: Training the ConvNet

The overall training process of the Convolution Network can be summarized as below:

Step1: We initialize all filters and parameters / weights with random values

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]

Since weights are randomly assigned for the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

Total Error = $\sum$ ½ (target probability – output probability) ^{2}

Step4: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.

The weights are adjusted in proportion to their contribution to the total error.

When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].

This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.

Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2-4 with all images in the training set.

In step 1 and step 2, we have too much parameters / weights, so we use algorithm to cut the weights.

3 PRUNING

3.1 DSD ALGORITHM

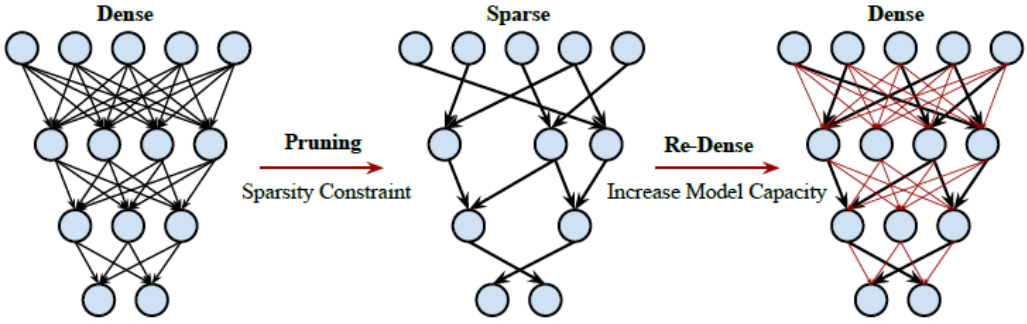

Figure 2: Dense-Sparse-Dense Training Flow. The sparse training regularizes the model, and the final dense training restores the pruned weights (red), increasing the model capacity without overfitting.

Algorithm 1: Workflow of DSD training

Initialization: $W^{(0)}$ *with* $W^{(0)} \sim N(0, \Sigma)$
Output: $W^{(t)}$.

Initial Dense Phase

while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $t = t + 1$;
end

Sparse Phase

// initialize the mask by sorting and keeping the Top-k weights.
$S = sort(|W^{(t-1)}|)$; $\lambda = S_{k_i}$; $Mask = \mathbb{1}(|W^{(t-1)}| > \lambda)$;
while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $W^{(t)} = W^{(t)} \cdot Mask$;
  $t = t + 1$;
end

Final Dense Phase

while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $t = t + 1$;
end
goto *Sparse Phase* for iterative DSD;

The algorithm of DSD, an algorithm to prune and re-dense the network.

Put our eyes on the third part(pruning)

// initialize the mask by sorting and keeping the Top-k weights.
$S = sort(|W^{(t-1)}|)$; $\lambda = S_{k_i}$; $Mask = \mathbb{1}(|W^{(t-1)}| > \lambda)$;
while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $W^{(t)} = W^{(t)} \cdot Mask$;
  $t = t + 1$;
end

In this part we see, the algorithm keeps the Top-k weights and the author use this way to find top key.

For each layer W with N parameters:

1.Sorted the parameters using their absolute value.

2.Picked the $k - th$ largest one $\lambda = S_k$ as the threshold.
$k = N * (1 - sparsity)$

3.Generated a binary mask to remove all the weights smaller than λ.

It is a simple heuristic to quantify the importance of the weights using their absolute value.

To find a better way to identify how important the weight is. We introduce the Affine Arithmetic.

3.2 AFFINE ARITHMETIC

An ideal quantity x (given or computed) is represented by an affine form $\hat{x}$, which is a first-degree polynomial:

$\hat{x} = x_0 + c_1\varepsilon_1 + c_2\varepsilon_2 + \cdots + c_n\varepsilon_n$

x_0: central value of the affine form $\hat{x}$.

c_i: coefficients, finite floating-point numbers.

ε_i: symbolic real variables, $\varepsilon_i \in [-1, 1]$.

Each noise symbol ε_i stands for an independent component of the total uncertainty of the ideal quantity x.The corresponding coefficient c_i gives the magnitude of that component. The sources of this uncertainty may be "external" (already present in the input data) or "internal".

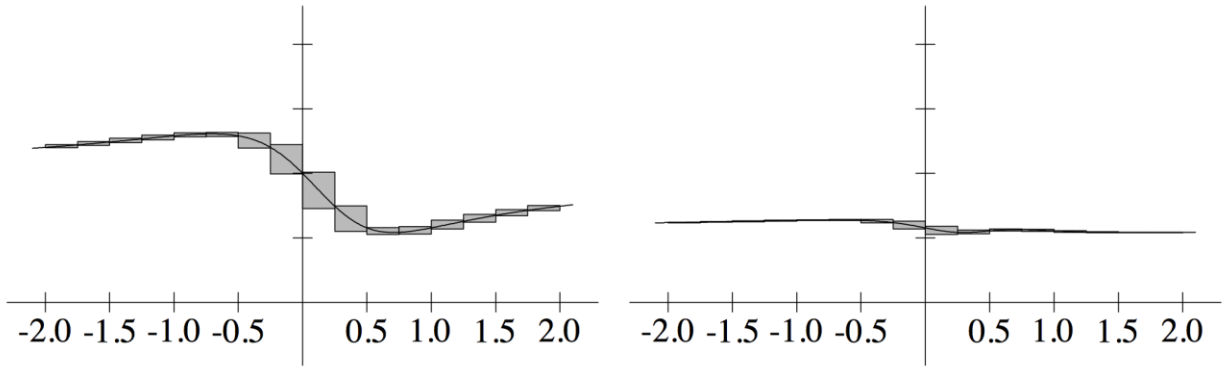

Figure 3: Avoiding error explosion in iterated functions with affine arithmetic. The plots show the result of evaluating y = g(x) = p x2 − x + 1/2/ p x2 + 1/2 (left) and its iterate y = h(x) = g(g(x)) (right), over 16 equal intervals ¯x of with 0.25 in [−2 +2]. Each sub-interval was converted to an affine form, the function was evaluated with AA, and the result ˆy was converted to an ordinaty interval ¯y for plotting.

4 EXPERIMENT: ALEXNET ON IMAGENET

4.1 ALEXNET CONV-5

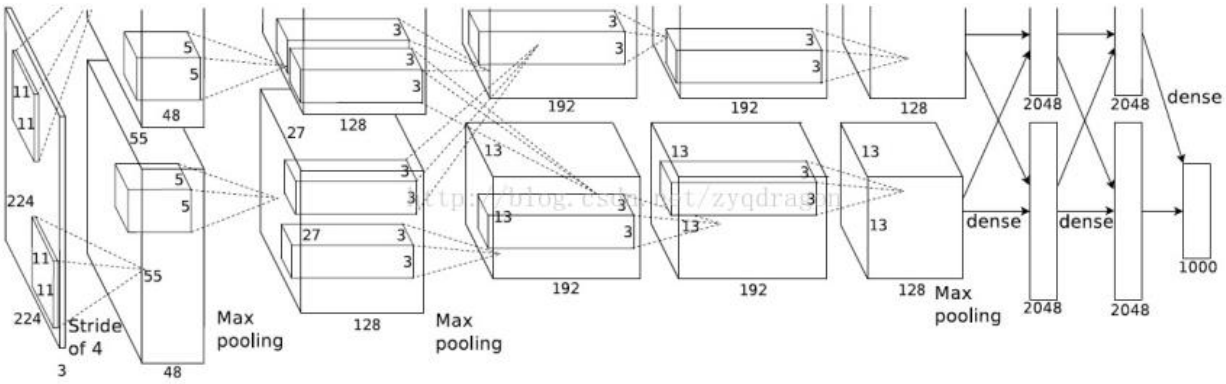

Figure 4: the picture in the classic thesis published by Alex in 2012

Basic structure of Alex net:
a. There are 8 layers, of which the first 5 layers are convolutional, the latter 3 layers are full-connected, and the last full-connected layer output is softmax with 1000 outputs. The final optimization goal is to maximize the average multinomial logistic regression.

b. Immediately after the first layer conv1 and conv2 is the Response-nomalization layer, which is the norm1, norm2 layer.

c. The operation that follows each conv layer and the full-connected layer is a ReLU operation.

d. Max pooling operation is followed by the first norm1, norm2, and the fifth conv layer, which is conv5

e. Dropout operation is in the last two full-connected layers.

Use C++ to build an Alex-Net

```
layer ANet[Net_depth] = {
// ************************iw**ih***id***ow**oh**od**ww*wh*wd****s*pad
      { "data",      227,227, 3,   227,227,3,   0,0,  0,   0,0 }, // 0
      { "conv1",     227,227, 3,   55, 55, 96,  11,11,3,   4,0 }, // 1
      { "conv2",     27, 27,  96,  27, 27, 256, 5,5,  48,  1,2},  // 2
      { "conv3",     13, 13,  256, 13, 13, 384, 3,3,  256, 1,1 }, // 3
      { "conv4",     13, 13,  384, 13, 13,  384, 3,3,  192, 1,1 }, // 4
      { "conv5",     13, 13,  384, 13, 13, 256, 3,3,  192, 1,1 }, // 5
      { "pool1",     55, 55,  96,  27, 27, 96,  3,3,  0,   2,0 }, // 6
      { "pool2",     27, 27,  256, 13, 13, 256, 3,3,  0,   2,0 }, // 7
      { "pool3",     13, 13,  256, 6,  6,  256, 3,3,  0,   2,0 }, // 8
      { "norm1",      55, 55,  96,  55, 55, 96,  0,0,  0,   0,0 }, // 9
      { "norm2",      27, 27,  256, 27, 27, 256, 0,0,  0,   0,0 }, // 10
      { "fc6",       6,  6,   256,  1, 1,  4096,6,6,  256,   0,0 }, // 11
      { "fc7",       1,  1,   4096, 1, 1,  4096,1,1,  4096,   0,0 }, // 12
      { "fc8",       1,  1,   4096, 1, 1,  1000,1,1,  4096,   0,0 }, // 13
```

```
    { "softmax1",    1,  1,   1000, 1, 1,  1000,0,0,  0,     0,0 } // 14
};
```

4.2 USE THE LIB-LIBAFFA TO GET THE AA-OUTPUT, AA-WEIGHT

Lib-libaffa is a library built by Olivier Gay on Github.

We add a_i:*ε_i to w_i as s noise, and at the output we will get a coefficients c_i caused by the noise and know how important this weight to the net by c_i

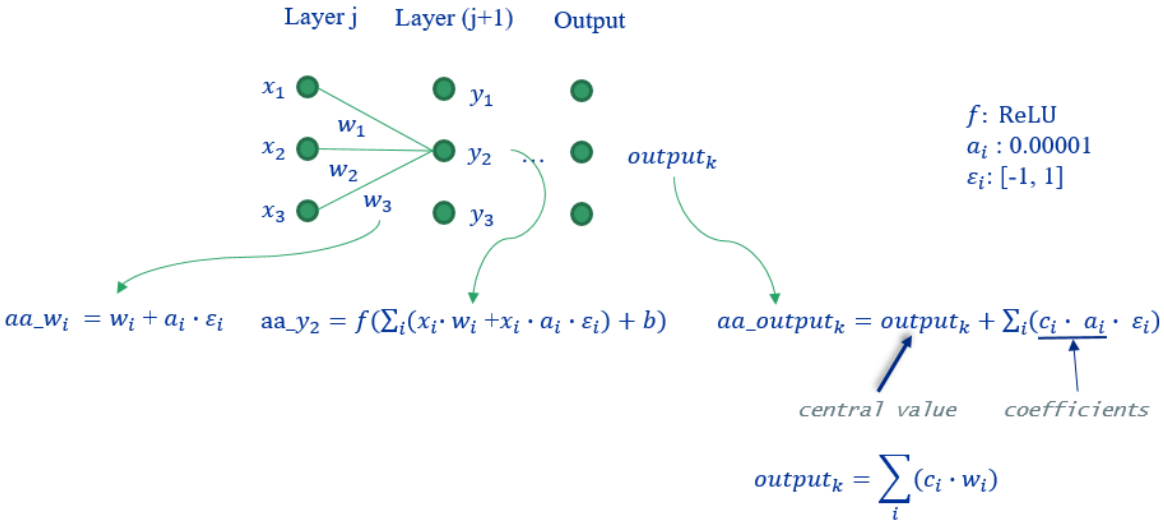

Figure5: Affine Arithmetic Lib-libaffa

For each layer W with N parameters:

1. Sorted the $c_i \cdot w_i$ using their absolute value.(which is w_i absolute value before)

2. Picked the $k - th$ largest one $\lambda = S_k$ as the threshold.
   $k = N * (1 - sparsity)$

3. Generated a binary mask that record location information where $|c_i \cdot w_i|$ is smaller than λ.

4. Remove all the weights based on the binary mask generated in step 3.

(according to the DSD algorithm at 3.1)

4.3 THE EXPERIMENT PROGRESS AND RESULT

4.3.1 AA Coefficients c_i

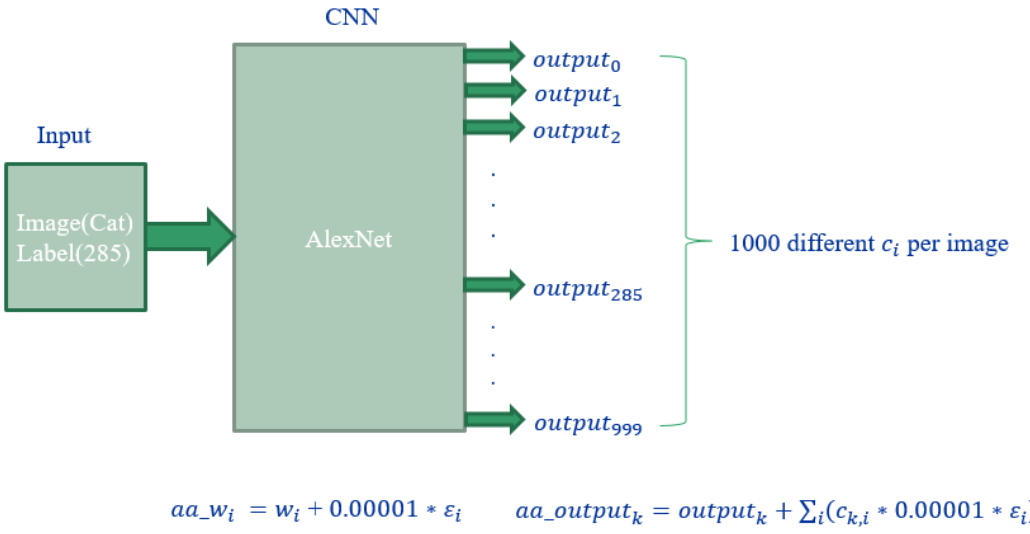

Figure6 1000 outputs of CNN

From figure 6 we can see there are 1000 output, it means they are 1000 Classifications.

Output0=$c_0 \cdot w_0 + c_1 \cdot w_1 + c_2 \cdot w_{2+\ldots} + c_{0,0} * 0.00001 * \varepsilon_0 + c_{0,1} * 0.00001 * \varepsilon_1 + \cdots$

$$= \sum_i (c_i \cdot w_i) + \sum_i (c_{0,i} * 0.00001 * \varepsilon_i)$$

Output1=$c_0 \cdot w_0 + c_1 \cdot w_1 + c_2 \cdot w_{2+\ldots} + c_{1,0} * 0.00001 * \varepsilon_0 + c_{1,1} * 0.00001 * \varepsilon_1 + \cdots$

$$= \sum_i (c_i \cdot w_i) + \sum_i (c_{1,i} * 0.00001 * \varepsilon_i)$$

.

.

.

Output285=$c_0 \cdot w_0 + c_1 \cdot w_1 + c_2 \cdot w_{2+\ldots} + c_{285,0} * 0.00001 * \varepsilon_0 + c_{285,1} * 0.00001 * \varepsilon_1 + \cdots$

$$= \sum_i (c_i \cdot w_i) + \sum_i (c_{285,i} * 0.00001 * \varepsilon_i)$$

$$\boldsymbol{aa\_output_k = output_k + \sum_i (c_{k,i} * 0.00001 * \varepsilon_i)}$$

Every output has a c_0 , a c_1 , a c_2 … ,but every output's c_0 is different. There are 1000 c_0,1000 c_1, 1000 c_2… We run eight pictures, so there are 8000c_0, 8000 c_1…

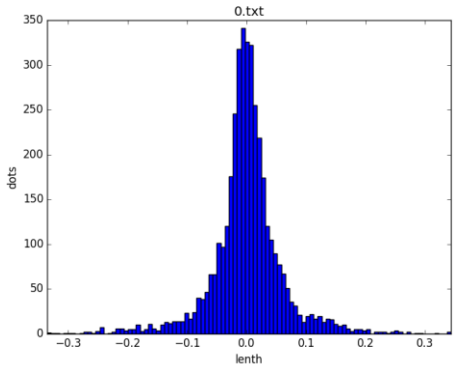

Figure7 distribution of c_0

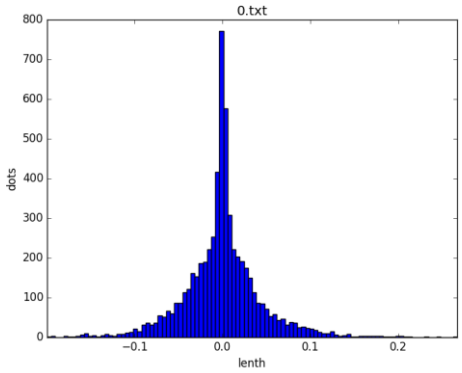

Figure8 distribution of c_1

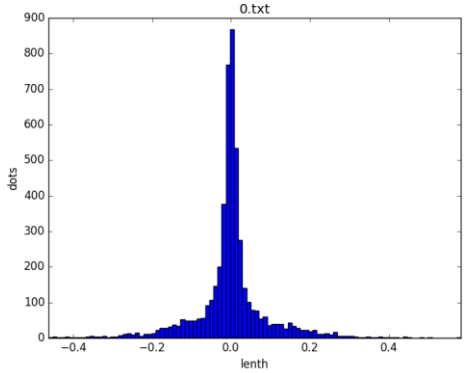

Figure9 distribution of c_2

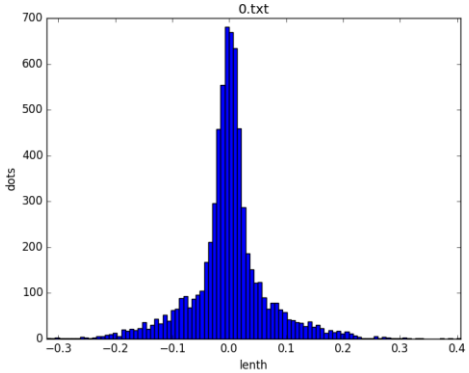

Figure10 distribution of c_3

From the pictures we can see all coefficients satisfy Normal distribution. Most of them are around zero.

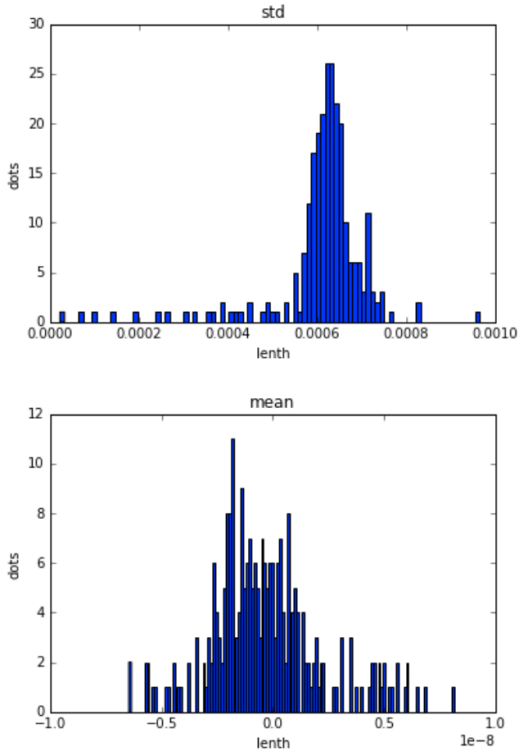

Figure 11 Standard Deviation and mean values of c_0

4.3.2 different method to use the AA Coefficients c_i

Method 1 and Method 2:

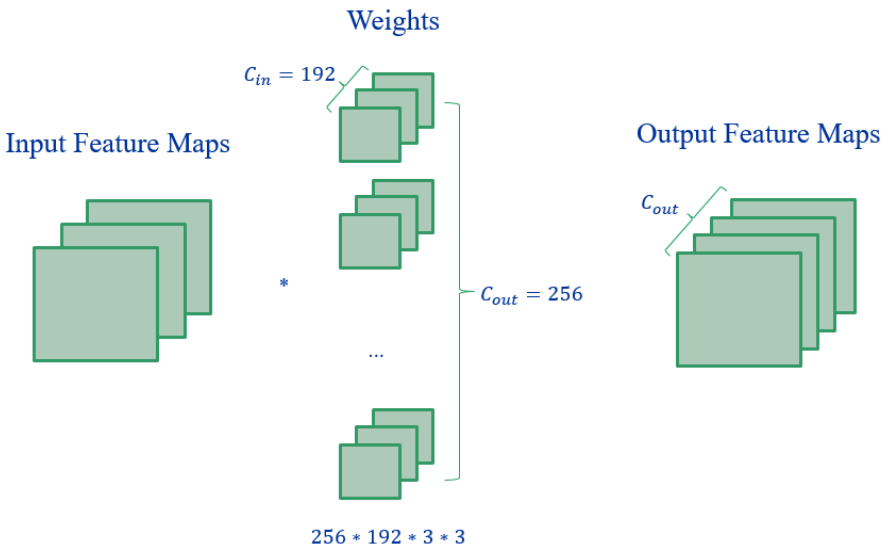

Figure12

Pruning Method-1

As we mentioned in 4.3.1 eghit pictures have 8000 $\boldsymbol{c_i}$ **,** the first way is to calculate the mean of every picture's absolute value of 1000 coefficients, then add 8 pictures and get the mean.

- AA Output:

$$aa\_output_k = output_k + \sum_i (c_{k,i} * 0.00001 * \varepsilon_i)$$

- Choose c_i of one image :

  $c_{img,i} = \frac{1}{1000}\sum_{k=0}^{999} |c_{k,i}|$

- We have aa-coeff of 8 images:

$$c_i = \frac{1}{8}\left(|c_{img0,i}| + |c_{img1,i}| + \cdots + |c_{img7,i}|\right)$$

The result is shown below:

Benchmark: Loss [0.92719088] Accuracy [0.77584]
Coeff*Weights: Loss [1.84362456] Accuracy [0.61764] Pruning Rate: 50%
Song: Loss [0.96775612] Accuracy [0.76844] Pruning Rate: 50%

Worse than the way that just leave the weight with big absolute value.

Pruning Method-2:

In the second Method, when we input one picture we find the output channel which has the biggest coefficient and just use this number instead of all coefficients of this pictures.

- AA Output:

$$aa\_output_k = output_k + \sum_i (c_{k,i} * 0.00001 * \varepsilon_i)$$

- Choose c_i of one image :

  $c_{img,i} = c_{k,i}$, here $k = label$

- We have aa-coeff of 8 images:

$$c_i = \frac{1}{8}\left(|c_{img0,i}| + |c_{img1,i}| + \cdots + |c_{img7,i}|\right)$$

The result is shown below:

```
Benchmark:
Loss:  0.931621611982584 Accuracy: 0.7750599976181984

Ci*Wi: Pruning Rate 50.0%
Loss:  2.046257845520973 Accuracy: 0.5617200018465519

Song: Pruning Rate 50.0%
Loss:  0.9632426638603211 Accuracy: 0.7698199979662895
```

Worse than method 1 and the way just leave the weight with big absolute value.

Other attempt:

We set the same coefficients for one whole filter which I circled out.

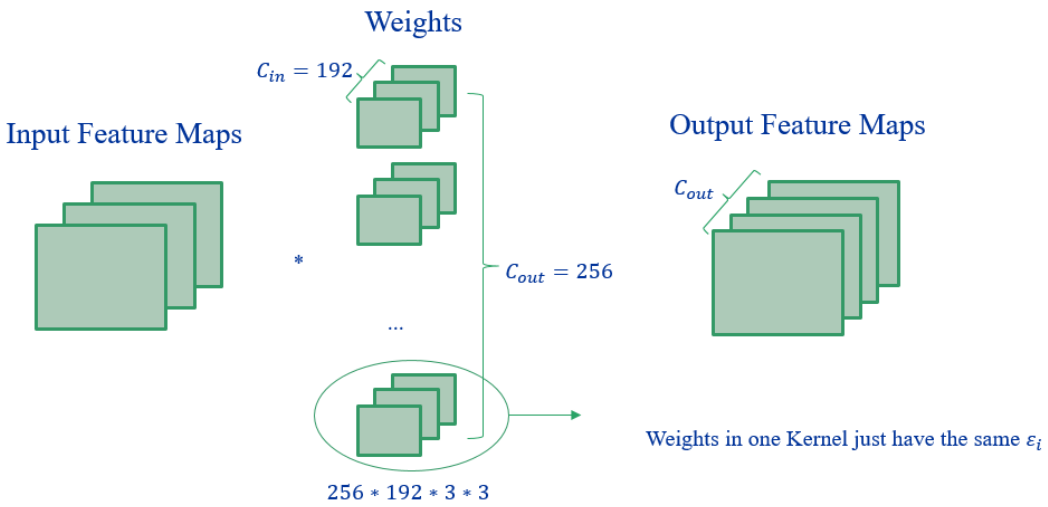

Figure 13

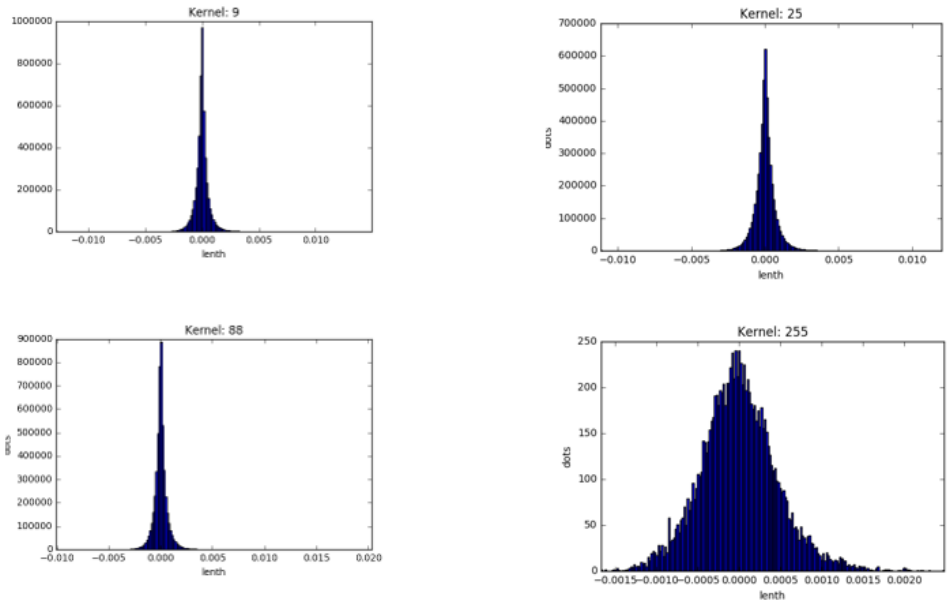

Figure 14 Distribution of AA Coefficients c_0 c_1 c_2 c_3 use method3

From the result it is similar to method1 and method2 and worse than method1.

5 CONCLUSION

We actually make the model smaller and quicker by pruning and the loss and accuracy are in a good number, but after use the Arithmetic, the result is worse than the easy way which leave the big $|w_i|$. There are a team is doing research that compared many ways to pruning the weight, they find some algorithm indeed can get a god result but the easy way to rank the absolute value is easier, quicker and has the same good result. In my opinion, the CNN is a way full of Randomness. At the first step, we initialize all filters and parameters / weights with random values, so maybe the easy way is better or how we pruning is not so important or even maybe the weight is not so important. Which really play an important role is the structure, the way all nodes are connected but not the weight on that connecting line.

6 FUTURE WORK

We can compare more algorithm of rank the importance of parameters to see is that true that the easier algorithm has a better effect or just we haven't found the right way. Next, we can throw the highlight on the structure of the network. For example, try to use simulated annealing to arrange the connect between nodes.

7 ACKNOWLEDGEMENTS:

First of all, I would like to extend my sincere gratitude to my supervisor, Yajun Ha, for he gave me such an interesting theme to do during my summer vacation which make me learn a lot on deep learning and Neural Network.

The I must say thanks to Yingjie Zhang my senior apprentice who do lots of work of this research and teach me much knowledge on coding and reading papers and even help me install the Caffe.

Reference

[1] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz Nvidia

pruning convolutional neural networks for resource efficient inference

[2] Cong Leng , Zesheng Dou, Hao Li ,Shenghuo Zhu, Rong Jin Alibaba Group, " Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM"

[3] An Intuitive Explanation of Convolutional Neural Networks from the data science blog

[4] Han et al. DSD: Dense-sparse-dense training for deep neural networks. In *ICLR, 2017*

[5] J. Stolfi et. al An Introduction to Affine Arithmetic