YOLOv3 Model Compression Based on Clustering and Pruning

*Github Address:

Lanxuan Zhou
Shanghai Jiao Tong University
Student ID: 516030910491

Quan Luo
Shanghai Jiao Tong University
Student ID: 516030910506

***Abstract*—Convolutional neural networks have made great success however, it needs lots of computational resources and much of them are redundant. Thus DNN compression become more and more important. YOLO is a object detection deep neural network model which achieves great success last year. Though it's fast, implementing it on a embedded system is almost impossible because of limited computing resources.**

In this paper, we proposed a new pruning method to the certain model and made weight clustering on it in order to compress the model. Eventually we achieve mAP 95 on a small dataset with 2 million parameters pruned and clustering weights into 32 groups.

I. Introduction

YOLO is a recently proposed object detection framework which is fast and accurate at the same time. Not similar to the previous approaches like R-CNN which use region proposal methods to first generate potential bounding boxes in an image, YOLO reasons globally about the image when making predictions. Unlike those sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

Traditionally, we use weight as a standard to prune out the useless weights. However, we found this kind of pruning is just intuitive and not reasonable. We proposed a different way of pruning based on similarity according to the feature of YOLO. It has several benefits over traditional methods of object detection. First it acts more reasonable but not violently cut the weight that are small. Second it literally acts better than weight pruning in our experiment.

Weight clustering and quantization are other approaches to decrease the quantize of the model and both of them are important. The difference between them is that quantization will cluster the weights into predefined, fixed numbers while clustering just cluster the weights by K-means. Generally, clustering will result in high accuracy because of the flexibility and quantization is more hardware-friendly due to the predefined number. In our works, we implemented clustering and try to implement a quantization approach provided by Google.

II. Motivation

Humans glance at an image and instantly know what objects are in the image and where they are. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to do the same things, enabling assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Thus it's becoming increasingly significant.

YOLO rolls velocity and accuracy into one. Thus it's really satisfying real-time detection to some extent. Though it's very fast, it still cannot be applied to embedded system on real-time detection because of the great quantize of the model. However, many applications of object detection implemented on a small embedded system like self-driving car and so on. We have tested it on Jetson-TX2 and it didn't work on such a embedded system, only yolo-tiny can be implemented but it's not so accurate as YOLO. Thus, we proposed to compress the big model which is having about 6.2 million parameters by some state-of-the-art pruning and clustering method. Motivated by the weight pruning method, we try to find out a better standard for weights to be pruned since weight pruning is really unreasonable. For clustering, we notice that nobody has tried this kind of method on a larger network. In [4], they only tried ADMM Quantization and Pruning on MINIST datasets by a very small network consisting of two Convolutional layers and two FC layers.

III. Background & Related Work

A. YOLO Model

The YOLO model we are working on is the latest version of YOLOv3 mentioned in [1]. It improves the previous version using 3 x 3 and 1 x 1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers. The whole YOLO network structure is shown in Fig.1. It has a high accuracy on COCO dataset and others but we are working on some smaller datasets like 1-object kangaroo dataset and multi-object datasets.

Cutting the graph into cells, YOLO predicts multiple bounding boxes per grid cell. At training time it only wants on ebounding box prexictor to be responsible for each object. Thus it assgins one predictor to be "responsible" for predicting an object based on which prediction has the highest current

IOU with the ground truth. During training it uses multi-part loss function denoting penalizes classification error and bounding box coordinate error.

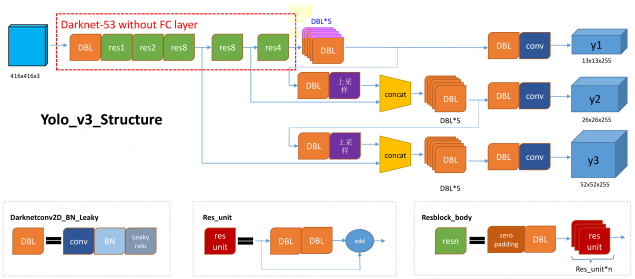

Fig. 1. YOLO Structure

B. Weight Pruning

Neural networks are typically over-parameterized, and there is significant redundancy for deep learning models. This results in a waste of both computation and memory. Nowadays, there are a lot of pruning methods leading by this weight pruning method presenting by Song Han in [2]. They prune on VGG-16 by sequencing the weight inside a certain layer. Prune the least weighted weights and get a good result. After that, many other kinds of pruning methods and better standard appears. In [3], Nvidia proposed the Oracle Pruning method which may be most satisfied the goal by

$$min_W \|C(D|W') - C(D|W)\| \quad s.t \|W'\|_0 \leq B$$

in order to find a minimized cost value $C(D|W)$ and the l_0 norm in $\|W'\|_0$ bounds the number of non-zero parameters B in W'. And in [4], [6] and [7], they transformed the problem into a math non-convex optimization problem and use ADMM approach to solve it. They consider weight pruning with clustering or quantization as the question to minimize the function and solve the problem via ADMM while it's difficult to solve directly. In [8], they tried to learn not only weights but also connections for efficient neural networks. Then they remove all connections whose weight is lower than a threshold. This pruning converts a dense, fully-connected layer to a sparse one.

Though these methods have their advantages, they also have some disadvantages. For weight pruning, we hardly know if a single weight really means nothing. When we do weight pruning, a single change of the percentile may cause great changes and some of the weights may be small but significant. It is not reasonable. For Oracle Pruning, we don't have such GPU resources like Nvidia does. ADMM is hard to implement on a larger network like YOLO, their tensorflow implementation is on a 2-convolutional + 2-FC small network training on MINIST dataset. Thus they may be not good for our project.

C. Weight Quantization

The quantization scheme we have tried is provided in [2]. The basic requirement is that it permits efficient implementation of all arithmetic using only inter arithmetic operations on quantized values. The general scheme is an *affine mapping* of integer q to real number r of the form

$$r = S(q - Z)$$

for some constants S (for "scale") and Z (for "zero-point"). By using this equation, The matrix multiplication can be transformed to

$$q_3^{(i,k)} = Z_3 + \frac{S_1 S_2}{S_3} \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2)$$

which is easier and less time-consuming for hardware to calculate. Quantization involves modify the structure of neural networks

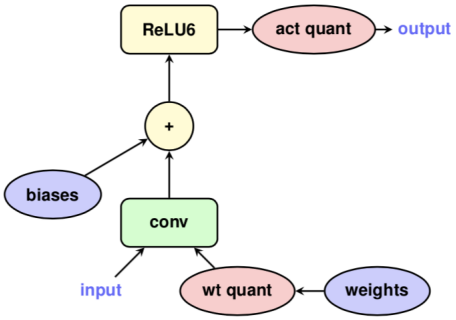

Fig. 2. The structure of quantized network

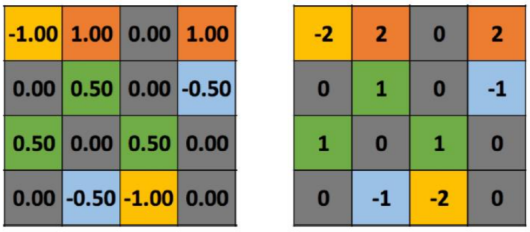

Fig. 3. Two representation of quantization. The left figure are weights in quantization level, and the right figure are weights stored in hardware

This quantization scheme improves the tradeoff between accuracy and on-device latency. discussion

D. Weight Clustering

The clustering scheme is mentioned in [5], which is also called *weight sharing*.In this scheme, clustering is implemented by k-means to identify the shared weights for each layer of a trained network. It partitions n weights $W = \{w_1, w_2, ..., w_n\}$ into k groups $C = \{c_1, c_2, ..., c_k\}$ so as to minimize the within-cluster sum of squares (WCSS). This scheme obtain a good performance in YOLOv3 model. Also, both of clustering and quantization can tanke the advantage of cache because of the time localization generated by using limited weights values.

Fig. 4. Two representation of clustering. The left figure are weights after clustering, and the right figure are centroid values

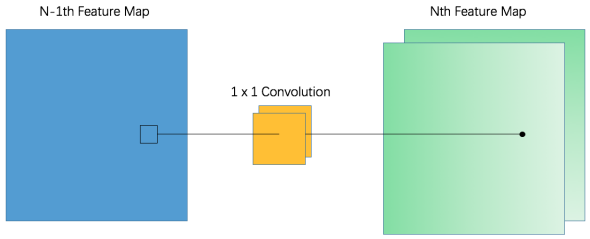

Fig. 5. The two 1 x 1 filters are similar, thus the output feature maps are also almost the same and therefore useless. Hence, we prune out one of those 1 x 1 filters to get a simplified model of the YOLO neural network

E. Similarity-based Pruning Theory

When getting the object, what we first want to do is to look at the distinct feature of YOLO and propose a method maybe only satisfying this certain structure. Finally we found that YOLO is only having 1 x 1 and 3 x 3 kernels. For those 1 x 1 kernels, we can view it as a vector instead of a tensor. Considering a single layer shown in Fig.5. If two 1 x 1 kernels are really "similar", then the same input feature map can get a very "similar" feature map out. These two "similar" feature maps are really useless since the features they extract are almost the same. Then we can conclude we can prune one of these two "similar" 1 x 1 filters since the little difference of the two feature map.

Then the question becomes how to measure the similarity between two 1 x 1 vectors. As we mentioned above, they can be seen as a vector. For vectors, we can use Euclidean distance & Cosine similarity to denote how they distinguish with each other. We add a weight to Euclidean distance and cosine similarity to get a balanced result. Actually for 3 x 3 vectors, matrix norm can do the same job but in vectors this is much simpler than in tensor. Then we can get the following formula to compute the similarity between two 1 x 1 filters:

$$w_1 ED(k_1, k_2) + w_2 CS(k_1, k_2)$$

where w_1, w_2 denotes the weight distributed to the two standard function, ED denotes euclidean distance, CS denotes cosine similarity and k_1, k_2 denotes the two 1 x 1 filters. Getting the formula, we can measure the similarity and prune the most similar filters in order to wash out those "same" filters.

IV. Problem Formulation

Nowadays, computer vision has become one of the most significant part of deep learning and object detection is the most important part in computer vision. Traditional approaches of object detection such as sliding windows are powerful but not powerful enough due to the inference time which cannot be ignored. Compared with these traditional ways, YOLO is an ideal choice. It is extremely fast, it can reasons globally about the image, and it can also learn generalizable presentation of objects. But the problem is that YOLO (especially YOLOv3) cannot be put into embedded system.

The motivation that we want to put int into the embedded system is that the object detection play an important role in embedded system. For example, in a driving auxiliary system, object detection can detect the obstacles lying in the way and help the drivers to avoid it. It will certainly decrease the ratio of accidents and improve the safety during driving experience. That's why we think putting YOLO into embedded devices is significant.

In this project, what we want to do is to put the YOLOv3 model into an embedded system, which means if we put an image into the model, we should get the result of object detection by the embedded device within a significantly small time interval. The current problem is that YOLOv3 cannot be put into the embedded system due to the oversize of YOLOv3. By reading related papers, we got some elementary ideas about solutions — weights pruning and weights quantization.

The first is weights pruning and targeted dropout. Both of them can reduce the size of network by pruning some insignificant channels. The difference is that targeted dropout implements pruning during the training process while weights pruning prunes channels after training. Weights pruning has already been used widely to decrease the size of models in deep learning field. This scheme seems to achieve distinct performance because it surely minish the size of models, but pruning is not so robust, which means the result of pruning can be substantially influenced when weights change slightly. Targeted dropout can avoid this problem. It is robust enough to the subtle changes, but it is hard to implement because it must prunes during the training process. In this project, we implemented weights pruning.

The second is quantization and clustering. Both of these ways take the advantages of hardware directly by decreasing the number and the complexity of multiplications so that when the devices are running inference networks, the inference time can be reduced obviously. Quantization can group weights from float points to fixed points, which makes the hardware to implement multiplications faster than before. Clustering groups the weights into lots of groups (the number of groups is predefined) so that the overhead of memory usage is certainly dimished, which also decrease the size of models. Generally, quantization can achieve a better performance in network

acceleration and compression, but clustering yields higher accuracy when because the groups are not predefined. In this project, we implemented weights clustering and tried to implement quantization offered by Google.

V. Proposed Method

A. Similarity-based Pruning

Detailed formulation of similarity based pruning and the algorithm is as follow. We train YOLO network on a relatively small datasets and then get an model. We reimplement the convolutional layers with a mask to make the pruning efficient since it can prune out the gradients. Then we do the following similarity-based pruning onto every layer. For those 3 x 3 layers, we still use the weight-based pruning to get the network become smaller but not only pruned on those 1 x 1 filters. Then the whole structure is for 3 x 3 filters, we prune some of its weight based on the number. For 1 x 1 filters, we prune some of the filters based on their similarity. After that we finetune the network by retraining some episodes.

Algorithm 1 Similarity-based Pruning Algorithm

for layer in model **do**
  if $layer.shape = (1, 1)$ **then**
    for all distinct filter f1, f2 in layer **do**
      $similarityArray \Leftarrow w_1 CS(f_1, f_2) + w_2 ED(f_1, f_2)$
    end for
    $threshold \leftarrow w_3 \times max(similarityArray)$
    for all distinct filter f1, f2 in layer **do**
      for all $similarity(f_1, f_2) > threshold$ **do**
        // Gets Pruned
        $f_1 \leftarrow 0$
      end for
    end for
  else
    Do Weight Pruning
  end if
end for

The algorithm is simple but there are some crucial and knott hyperparameters like w_1, w_2 and w_3. We use the loss fall to choose a better hyperparameter. That is, inside a group of hyperparameters, we do the same pruning process to the model. We see if the loss ascend. If the loss ascends quickly then we desert this group of the hyperparameters and then we eventually find a proper set of parameters and get a magnificent result shown in our experiment section.

During our experiment, we found that the pruning process is sensitive. Some of the YOLO layers are really not approachable such as the last predicting YOLO layer. In addition some of the convolutional layers are also not "prunable" since it stands a essential function of YOLO object detection. Thus we choose only to prune the 1 x 1 filters only in the first 80 convolutional layers and set a overflow threshold. If all the similarity is low, then we don't prune any of the filters inside the layer. But for 3 x 3 weight pruning, we prune for all layers with the first 30 percent of weights since it's pruning only some of the weights but not the whole filter. Thus its influence is not as strong as our 1 x 1 kernel pruning and can be implemented on all the convolutional layers.

B. Weight Clustering

We implemented the clustering by training YOLOv3 on a very small dataset and get the weights after training. Then we implemented k-means for weights of each layer of the model so as to minimize the WCSS:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2$$

This is easy to implement by adding k-means in each layer,

Algorithm 2 Clustering

Define the number of groups $n = 2^k$
Training Model $model$
for $layer$ in $model$ **do**
  $weights \leftarrow$ weights of $layer$
  Implement k-means on $weights$ with n centroid values and save it to $new\_weights$
end for
Load $new\_weights$ to inference model
Infer by the new inference model

and the number of values of weights in each layer can be decreased to the number of groups, which can use cache better.

C. Weight Quantization

To implement quantization, we first need to quantize weights of each layer of a trained model by using *affine mapping* of integers q to real number r,

$$r = S(q - Z)$$

After quantization, we can get the parameters S and Z of each layer and save the quantized weights into another weights file. Then we can use these parameters and quantized weights to accelerate multiplications and compress the model. When facing a matrix multiplication, we have already know that,

$$r_\alpha^{(i,j)} = S_\alpha(q_\alpha^{(i,j)} - Z_\alpha)$$

From the definition of matrix multiplication, we know that,

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1) S_2(q_2^{(j,k)} - Z_2)$$

By rewriting this equation, we have,

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2)$$

where $M := \frac{S_1 S_2}{S_3}$. In this equation, M is the only non-integer parameters. But this can also be written as,

$$M = 2^{-n} M_0$$

where M_0 is can be seen as a fixed-point decimal and n is a non-integer. As a result, multiplication by M_0 can be implemented as a fixed-point multiplication and multiplication by 2^{-n} can be implemented with an efficient bit-shift. A common approach to get the corresponding parameters of each layer is by using the equations shown as follows,

$$clamp(r; a, b) := min(max(x, a), b)$$

$$s(a, b, n) := \frac{b-a}{n-1}$$

$$q(r; a, b, n) := \left\lfloor \frac{clamp(r; a, b) - a}{s(a, b, n)} \right\rceil s(a, b, n) + a$$

Then we can use the following algorithms to get the corresponding parameters and implement the quantizaiton,

Algorithm 3 Get quantization parameters

$bit\_width \leftarrow$ bit width of hardware
$max \leftarrow$ maximum value of weights
$min \leftarrow$ minimum value of weights
$S \leftarrow \frac{max-min}{2^{(bit\_width)}-1}$
$Z \leftarrow round(-\frac{min}{S})$

Algorithm 4 Quantization

Training Model $model$
for $layer$ in $model$ **do**
    $weights \leftarrow$ weights of $layer$
    Get quantization parameters of $weights$ and save the quantized weights to $new\_weights$
end for
Load $new\_weights$ to inference model
Modify the multiplications within the model
Add dequantization layer for the inference model
Infer by the new inference model

But during the project process, we failed to generate the dequantization layer in time so we only implement weights clustering.

VI. Experiment

The first experiment we do is on a one object simple dataset called kangaroo. It's used to detect kangaroos inside a image. We've done six different models of the dataset and get all of their mAP together. We can see in the table followed, before prunning the whole mAP is up to 98.8. When we don't use clustering and implement pruning only, weight pruning does worse than our proposed Similarity based pruning. Our pruning method has almost no decrease with 98.24 but weight pruning mAP drop to 94.48. Though weight pruning seems to prune more, we have tried another bigger threshold Similarity-based pruning, it gets mAP with 95.89 and prune more weights than weight pruning.

Then we do clustering on the same dataset. After clustering we can see the mAP is almost the same. But the model has been accelerated. After clustering, the values of weights are limited into $2^5 = 32$ values so that the network can use cache in a better way. Without pruning, the network can be down slightly faster than before with the mAP even increasing.

Pruning Method(Before Clustering)	mAP	Prune Ratio
No Pruning	98.91	0 / 6147W
Weight Pruning	94.48	1844W / 6147W
Similarity Pruning	98.24	1673W / 6147W
Similarity Pruning(Bigger Threshold)	95.89	1957W / 6147W

Whether Clustering	mAP	Inference time
No Clustering	98.79	19.99s
Clustering	99.04	18.93s

For weight pruning its mAP has falled swiftly to 93.15. But for our similarity pruning, the mAP dropped slightly to 98.51. Even we use bigger threshold, it also has a mAP of 95.88. After that we applied our algorithm onto a multi-object dataset and got the following result. Weight clustering is still worse than our similarity-based pruning. However we haven't got enough time to do weight clustering on the multi-object dataset. Thus there are only pruning results here.

Pruning Method(After Clustering)	mAP	Prune Ratio
No Pruning	98.80	0 / 6147W
Weight Pruning	93.15	1844W / 6147W
Similarity Pruning	98.24	1673W / 6147W
Similarity Pruning(Bigger Threshold)	95.88	1957W / 6147W

Pruning Method(Multi-Object)	mAP	Prune Ratio
No Pruning	82.6	0 / 6147W
Weight Pruning	75	1844W / 6147W
Similarity Pruning	77.8	1957W / 6147W

VII. Discussion & Conclusion

Due to time limit and resource limit, we haven't finished something yet. When we are training the YOLO on such a small dataset, it comes out OOM error on the server. We can only use callbacks to save all the trained weights and restart training by ourselves. However, we have tried finetune on our pruning. But we are surprised to find that sometimes the finetuned result mAP is lower than the original one. That inferred that these pruned weights maybe have something to do with the final result. What's more, due to the limited resources, we haven't tried on bigger dataset. At first we tried to train on COCO. However, after one day the epoch 1 only gave us an OOM error.

Due to the failure of designing dequantization layer, we haven't finished the quantization yet. And also, with the limited period, we implement the clustering by raw k-means with 5 bits rather than preprocessing the weights by CSR matrix or CSC matrix, which is implemented in [5].

In short, we have finished YOLO model compression by using weight clustering and proposed similarity-based pruning methods. The result is satisfying on our proposed method and clustering, mAP is almost the same as the original YOLO but a lot of redundant weights are pruned and clustered into 32 groups on the kangaroo and VOC multi-object dataset. After

clustering the inference time is faster than before, and after pruning the network become sparser being pruned about 2 million parameters.

ACKNOWLEDGMENT

Thanks for Professor Li Jiang's devotion to us. Also thanks for TA Chaoqun Chu and Zishan Jiang for answering our many boring questions and help us finish the whole project.

REFERENCES

[1] Joseph Redmon and Ali Farhadi, "YOLOv3: An Incremental Improvement".

[2] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", in arXiv 15 Dec 2017

[3] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila and Jan Kautz, "Pruning Convolutional Neural Networks For Resouce Efficient Inference", in ICLR 8 Jun 2017

[4] Shaokai Ye, Tianyun Zhang, Kaiqi Zhang, Jiayu Li, Jiaming Xie, Yun Liang, Sijia Liu, Xue Lin and Yanzhi Wang, "A Unified Framework of DNN Weight Pruning and Weight Clustering/Quantization Using ADMM", in CoRR 5 Nov 2018

[5] Song Han, Huizi Mao, William J.Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", in ICLR 15 Feb 2016

[6] Tianyun Zhang, Kaiqi Zhang, Shaokai Ye, Ning Liu, Jian Tang, Wujie Wen, Xue Lin, Makan Fardad and Yanzhi Wang, "ADAM-ADMM: A Unified Systematic Framework of Structured Weight Pruning for DNNs", in CoRR 21 Nov 2018

[7] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad and Yanzhi Wang, "A Systematic DNN Weight Pruning Framework using Alternating Directino Method of Multipliers", in CoRR 22 Aug 2018

[8] Song Han, Jeff Pool, John Tran and William J. Dally, "Learning both Weights and Connections for Efficient Neural Networks", in CoRR 30 Oct 2015