International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-4, November 2019

Better Communication Tool for Specially Disabled

Gogineni Saikiran, Anjusha Pimpalshende, Porika Dhanrajnath

Abstract: ***Sign language is widely used when a dumb communicates. However, non-sign-language people find it difficult in interpreting them. So, we had come up with a system that enables speech impaired to speak with an artificial voice in public communities using Artificial intelligence techniques. we propose a hybrid-weighted metric known as weighted pruning in deep convolutional neural networks. In this work, we report experiments of weighted pruning. we show that using a weighted pruning strategy we can achieve significant speed up in Faster RCNN object detection model by discarding 50% of filters. In this paper we show evidences to our claim by reporting mean Average Precision of weighted pruned CNN is slightly higher than existing pruning techniques. The former part of the paper focus on moulding convolutional neural networks in terms of their speed and scalability for deploying them on mobiles, embedded and further small gadgets. The latter part of the paper describes novel approaches in letting dumb speak as fast as normal person in public, without time lapse using natural language algorithms and recommendations.***

Keywords,Faster RCNN, Hand gesture recognition, Recommendations, SQL, TD IDF vectorizer, Weighted pruning.

I. INTRODUCTION

The system initially aims at building a Faster RCNN hand gesture object detector that's capable of reducing computations but retaining its original accuracy. On top of this detector, better and faster communication tool for specially disabled (dumb) is built. Over the past few years, deep convolutional neural networks have been very successful in computer vision recognition and detection tasks. Indeed, with each passing year, the performance of these networks is increasing with increase in depth of layers. This increase in depth accounts for increase in no of parameters and computations. Suppose if we perform a convolution operation with k filters, the resulting matrix will have depth equal to k. Of-Course out of these k filters some might be either redundant or doesn't contribute to actual task end task. The basic intuition is to remove such filters which eventually result in reducing unnecessary computations and a shrink in output tensor.

Manuscript published on November 30, 2019.
* Correspondence Author
Gogineni Saikiran*, CMR College of Engineering & Technology, Hyderabad, India
Anjusha Pimpalshende, CMR College of Engineering & Technology, Hyderabad, India
Porika Dhanrajnath, CMR College of Engineering & Technology, Hyderabad, India

The challenge is to retain original accuracy. Generally, to compress these CNNS we choose to prune either weights or filters. However, we constrain our self to prune filters (in convolution layers) rather than weights (in last fully connected layers) because they account for most of the floating-point operations.

All the existing works on filter pruning follows a similar recipe of choosing a single function (f) and ranking the filters based on the score generated by this function. This function determines the importance of each filter for the end task. In discarding the 50% of filters out of n filters, we set m=(n/2). The top m ranked filters are retained. The resulting pruned network is fine tuned. The existing works consider one among these (1) mean activation (2) L1 norm (3) entropy (4) average percentage of zeros (5) sensitivity as the required function to rank filters. Unlike above strategies, our method takes combined weighted soft-max score of above two mentioned functions (L1 norm and entropy) while ranking filters. We termed this to be weighted pruning. The Faster RCNN developed on this approach has retained approximately its original accuracy by reducing 50% of filters.

Consider an input image M of shape $(n_i \times h_i \times w_i)$ for the i^{th} convolution layer. Here n_i refers to no of channels in input tensor. The output tensor of this convolutional operation results in the shape $(n_{i+1} \times h_{i+1} \times w_{i+1})$. Filter F of shape $(n_{i+1} \times n_i \times k \times k)$ is used. In this convolution layer there are n_{i+1}filters, each of shape $(n_i \times k \times k)$. In these n_{i+1}filters, only certain are important, the rest can be omitted. [1] used an approach of calculating average percentage of zeros in each activation channel. Filters responsible for higher percentage of zeros in their corresponding activation channels are discarded. But in this way, we might end up in retaining values near to zeros as there are not actually zeros. [2] used L1 norm as criteria to find the importance of each filter. L1 norm of a filter is sum of its absolute weights. Filters with small L1 norm tend to produce feature maps with weak activations as compared to other filters in that layer. L1 norm of a filter is small, then weights in the filter will be small and hence produce small activations. These small activations may not influence end output. So, corresponding filters can be pruned away. [3] used concept of entropy. Entropy is measure of randomness or uncertainty. larger value of entropy means system possess more information. If activation channel consists of less information, corresponding filter can be pruned. In workflow of entropy-based approach, the output tensor of shape $(n_{i+1} \times h_{i+1} \times w_{i+1})$ is flattened to a vector of shape $(n_{i+1} \times 1)$ after applying max-pooling. These n_{i+1}values represent scores of corresponding activation channels for given image.

To calculate entropy more output values are needed, which are collected using a batch or subset of size m from training data. As a result, matrix E of shape $(m \times n_{i+1})$ is obtained for all m images. For each channel j, we focus on distribution $E_{:,j}$. Each column of matrix represents single activation channel for given m images. If entropy of any column of this matrix is low, then filter corresponding to such activation channel can be pruned.

Because that filter produces activation channel whose information is similar to all images. To compute entropy value of any column or channel of matrix E we split channel into s different bins and calculate entropy of each bin. Finally, the entropy is calculated as follows.

$$H_j = - \sum p_i \log (p_i)$$

Where, p_i is the probability of bin i, H_j is the entropy of channel j. A smaller value of H_j means channel j is less important. Channel j can be removed by pruning filter corresponding to channel j. If the feature map corresponding to filter produces same output for all the input then, this feature map and corresponding filter will not contribute to end output. We need to prune away such filters.

II. METHODOLOGY

To efficiently prove our point, it's crucial to look at some general facts other than the final performance of pruned model. To do so we draw an analogy with the products available in the market and observe that process of retaining few important filters is akin to purchasing few important products in the market. While we purchase a product, the importance of product is determined by several factors such as cost, likeliness, reliability and so on rather than by a single factor. Similarly, several characteristics are to be given attention while deciding importance of a filter rather than score generated by single function. Here, after severe experimentation we propose

2.1 Variant Score function of a filter

The existing works such as L1 norm consider the sum of weights of a filter in determining its importance. But during our experimentation we found odd filters whose sum of weights is large but actually don't contribute to end task. This might be because weights in filter are larger but similar or not properly learnt. These odd filters despite of their large L1 norm score produce similar activations to all images. Entropy of these filters is low. But when we opt to choose L1 norm as criteria we end up retaining these filters instead of discarding them. So, some other criteria should be helping hand to this L1 score to determine actual importance of a filter. Generally, small filters produce small activations resulting in small entropy. If the curve between L1 norm and entropy is strictly increasing or linear then we would have chosen one of these as criteria to rank filters. But experimentally we found at certain instances the curve tends to be nonlinear. As mentioned above some large filters resulted in low entropy as they produced similar activations for many images without differentiating them. The vice versa is also possible, a high entropy accompanied by weak weights in certain filters. But this case is rare to observe. However, we propose a medium level ground to balance this relation by retaining filters that tend to be good at both L1 norm and entropy. This strategy of pruning is performed layer by layer starting from last layer. Here L is the vector containing L1 norm of all filters in the layer. E is the vector containing entropy scores of all filters in the layer. Here we propose soft-max applied to each filters entropy score to relatively compare how important this filters entropy score is against other filters entropy scores. Soft-max of a value always lie in [0,1]. In below equations j takes values from 1 to n.

Score of a filter f_i = Soft-max $(L_i) \times$ soft-max (E_i)

Soft-max $(L_i) = (e^{\text{ith filter L1 norm}}) / (\sum e^{\text{jth filter L1 norm}})$

Soft-max $(E_i) = (e^{\text{ith filter entropy score}}) / (\sum e^{\text{jth filter entropy score}})$

Though L1 norm of a filter is high but entropy is relatively small then resultant score of a filter is negligible (for example $o.8 \times 0.2 = 0.16$) and filter is discarded. If both L1 norm and entropy of a filter are moderate the score of a filter is accepted (for example $o.6 \times 0.4 = 0.24$). If both are high, filter is perfect $(0.8 \times 0.8 = 0.64)$. This new strategy of pruning has produced mean Average precision that is higher than pre-existing strategies when 50% of filters are pruned away. Mean Average precision is slightly less but comparable when 25% of filters are pruned.

III. EXPERIMENTATION

In this section, we will show evidences for all our claims. We will compare the performance of different pruning techniques listed so far and will plug-in better communication algorithms on top of faster-RCNN.

3.1 Pruning Faster RCNN

Vgg-16 is the base component of faster RCNN model and other object detection specifics are built on top of it. We have chosen the pascal-VOC 2007 dataset [10] as [4] had already experimented with it using random pruning. So, we will consider experimenting with same dataset and demonstrate how this new strategy of weighted pruning slightly increased accuracy compared to other pruning methods. We first plugin standard unpruned vgg-16 into faster-RCNN and then train it. This model gives us mean Average Precision of 0.66. Later after plugging-in weighted pruned vgg-16, the mean Average Precision observed is 0.612. we can take a trained faster RCNN model and prune (weighted pruning) it directly. In this case, the no of frames per second are increased from 7.5 to 13. There is no significant drop in accuracy with this strategy. The accuracy obtained is slightly greater than rest of existing individual pruning techniques.

	Level of pruning	
Heuristics	25%	50%
Weighted pruning	0.646	**0.612**
Random	**0.647**	0.600
Mean Activation	**0.647**	0.601
Entropy	0.637	0.584
L1 – norm	0.628	0.608
Sensitivity	0.636	0.592

Table 1: Results of Object Detection obtained by Plugging in different pruned vgg-16 models in to Faster RCNN

Faster RCNN	Baseline	Pruning (50%)
mAP	0.66	0.652
fps	7.5	13

Table 2: Results of Object Detection when directly pruning(weighted) a fully trained Faster-RCNN

3.2 Natural Language Processing on top of Faster-RCNN

Indeed, our aim is not just to restrict to weighted pruning. Designing a system that maps sequential hand gestures to words or sentences in real time scenarios and finally spelling them out is highly challenging in consideration with speed and accuracy. The American sign language dataset consist of 29 gestures. Our weight pruned Faster-RCNN is capable of detecting hand in an image and classifying it to one of these 29 gestures. 26 gestures represent to 26 alphabets A-Z. one gesture represent NULL GESTURE to usually confirm end of either word or sentence. The other 2 gestures are meant for two recommendations that we discuss later. Weight pruned and fine-tuned Faster-RCNN achieved a descent accuracy on this custom sign language dataset. On test set, 90% of gestures are classified correctly. However, there are several challenges while pronouncing words or statements quickly. Let's limit our discussion to words and further move on to sentences.

Challenge 1: If user(dumb) wants to spell a word 'ring', he needs to make sequence of gestures representing corresponding letters in word 'ring'. Finally, user makes null gesture to confirm end of word and spells out using a speaker API. We used pyttsx3 to pronounce words. If Faster-RCNN incorrectly classifies one of the gestures, then it may be pronounced as 'sing' rather than 'ring'. This rate of misclassification is to be arrested to the possible extent. So, we decided to use a time buffer of 3 seconds to recognise a single gesture. Pruned Faster-RCNN detects 13 frames per second, faster than unpruned Faster-RCNN. This results in detection of 39 frames per 3 seconds and makes our task easy. In these 39 frames our model is working to classify same single gesture. It results in 39 predictions for a single gesture in 3 seconds. Picking out the largely occurring prediction has arrested misclassification of a gesture. After completion of 3 seconds, notification occurs on web cam asking user to enter the next gesture.

Challenge 2: Another issue to be dealt is quickness in spelling a word. While pronouncing a word 'television' it takes ten time-steps, each time step responsible for identifying a single character in the word 'television'. This result in huge time delay. Recommendations come into rescue. After making sequence of gestures for sub-string 'te', two words ('television' and 'teeth') pop up on screen as recommendations asking to choose between the two. If user continues and make another gesture 'a' in addition to 'te' without choosing any of two recommendations, then recommendations r1 and r2 dynamically changes to ('tear' and 'teacher'). This style of recommending continues till a null gesture or a recommended word is chosen. When a null gesture or recommended word is chosen then pyttsx3 speaker spells the word and clears the screen. Now instead of t time steps we produce a word in t/4 or t/8 time-steps, reducing delay in time.

suggesting two random words that start with given sub string is not accepted. Generally, if we find patterns in our daily life, we end up using hardly same 5000 words daily. In similar manner recommendations should work with knowledge of frequently used previous words. The system uses a SQL database consisting of top 40000 frequent words from NLTK brown corpus. Initially frequencies of all words are set to zero. When user choose null gesture or r1 or r2, the frequency of corresponding word gets incremented. System always recommend words that possess high frequency in database and that start with given substring. After immediate end of present word, we are no more left with sub string to make recommendation until the first gesture of next word is made. In this gap we can recommend a sentence possessing the context of previous word (through TD IDF VECTORISER). We have chosen a dataset consisting of most widely used 2000 daily life sentences. Instead of passing last word as query to TD IDF document parser (usually turns to be string searching algorithm) we passed last two or three words (tri gram) to retain the actual context. For example, if last two words spelled are ('which', 'dress') the sentence recommendations might be ('which dress is nice among these', 'which dress costs least'). The algorithms capability steadily increases as user frequently use it.

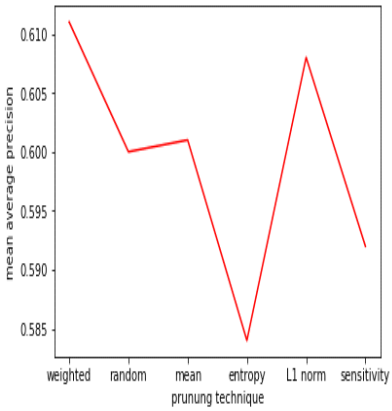

FIGURE 1: comparing mean average precision score of directly pruned Faster RCNN on pascal-VOC 2007 dataset [10] over different pruning techniques.

Alphabet ‘y’

we can observe from fig1 that weighted pruning attained a mean average precision of 0.610 which tends to be greater than remaining pruning techniques. Generally, this helps Faster RCNN to increase no of frames per second and decrease computation cost by 50%.

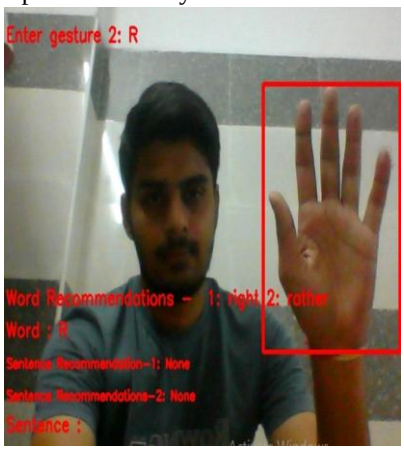

Figure 3: model recognises gesture1 as alphabet ‘r’ and recommends words such as ‘right’ and ‘rather’. User can choose any of the recommendations as per his need with ease of 1 time-step. If user wants to spell different words such as ‘route’. Then user can show second gesture to system.

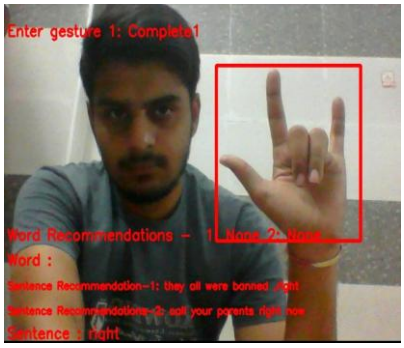

Figure 4: If user just spelled word ‘right’ using pyttsx3 (text to speech converter). The system recommends sentences (such as ‘call your parents right now’) that user would like to spell before entering first gesture of second word. But we used bigram (last 2 words for sentence recommendations).

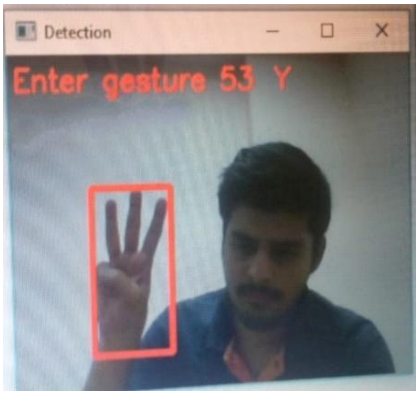

Figure 2: weight pruned Faster-RCNN detecting the hand and classifying it as Alphabet ‘y’

IV. CONCLUSION

with this application specially disabled (dumb) can speak sentences and words in public communities with ease and quickness. The model built on weight pruned faster RCNN and simple NLP algorithms can be easily deployed on smaller gadgets. It is highly scalable and flexible. Being run on computer vision techniques, this also benefits users from wearing overhead sensors. Unlike always showing gestures to mobile and speak, a similar hardware like watch, that can be attached to hand can be built.

REFERENCES

1. H. Hu, R. Peng, Y. Tai, and C. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. CoRR, abs/1607.03250, 2016.
2. H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, 2016.
3. J.-H. Luo and J. Wu. An entropy-based pruning method for cnn compression. arXiv preprint arXiv:1706.05791, 2017.
4. Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, Balaraman Ravindran. Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. arXiv:1801.10447v1 [cs.CV] 31 Jan 2018
5. R. Girshick. Fast R-CNN. In Proceedings of the International Conference on Computer Vision (ICCV), 2015.
6. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.
7. Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. arXiv preprint arXiv:1511.06744, 2015.
8. Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In CVPR, 2015b.
9. Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In ISCA, 2016a.
10. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascalnetwork.org/challenges/VOC/voc2007/workshop/index.html