Input-Dependably Feature-Map Pruning

Atalya Waissman(✉) and Aharon Bar-Hillel

Ben-Gurion University, Beer-Sheva, Israel
{ataliaw, barhille}@post.bgu.ac.il

Abstract. Deep neural networks are an accurate tool for solving, among other things, vision tasks. The computational cost of these networks is often high, preventing their adoption in many real time applications. Thus, there is a constant need for computational saving in this research domain. In this paper we suggest trading accuracy with computation using a gated version of Convolutional Neural Networks (CNN). The gated network selectively activates only a portion of its feature-maps, depending on the given example to be classified. The network's 'gates' imply which feature-maps are necessary for the task, and which are not. Specifically, full feature maps are considered for omission, to enable computational savings in a manner compliant with GPU hardware constraints. The network is trained using a combination of back-propagation for standard weights, minimizing an error-related loss, and reinforcement learning for the gates, minimizing a loss related to the number of feature maps used. We trained and evaluated a gated version of dense-net on the CIFAR-10 dataset [1]. Our results show that with slight impact on the network accuracy, a potential acceleration of up to $\times 3$ might be obtained.

Keywords: Neural networks · Pruning · Acceleration
Conditional computation · Feature-map

1 Introduction

The variability and richness of natural visual data make it almost impossible to build accurate recognition systems manually. Thus, it is machine learning algorithms which dominate these problems today. Deep Neural Networks (DNN) are hierarchical machine learning algorithm which currently provide the best results at the fields of computer vision, speech processing and Natural Language Processing (NLP). Focusing on vision, CNN allow obtaining good solutions for difficult tasks such as image classification, object detection/localization, captioning, segmentation and image generation. The research regarding CNNs is constantly evolving and the industrial integration of these nets increases significantly.

CNNs are a cascade of convolution, sub-sampling and activation layers which are applied on the input. The computational cost of these networks is high, often preventing their usage in real time applications. The improvement in computer hardware and specifically GPUs allow the usage of deeper networks providing more accuracy but raises the need for computational saving even more.

In this paper we present a network that uses only some of its feature maps, chosen in an input-depended manner, to classify images. Using the assumption that each

V. Kůrková et al. (Eds.): ICANN 2018, LNCS 11139, pp. 706–713, 2018.
https://doi.org/10.1007/978-3-030-01418-6_69

feature map of the network allocates and extracts a certain feature from its input [2, 3] we assume that given a specific example, only the computation of some feature maps is indeed improving the network accuracy. We hence build decision mechanisms, termed 'gates', which decide for each feature map in every layer whether the map should be computed or not. Since these gates make sharp decisions, we optimize their parameters using a reinforcement learning framework. We experiment with a dense-net base architecture, which is one of the more accurate contemporary alternatives, on the Cifar-10 dataset. Our results indicate that speedups of up to $3\times$ are obtainable with less than 2% error reduction.

2 Related Work

Many studies considered saving deep neural networks' computational cost. Diverse approaches are suggested, including low rank decomposition for convolutional and global layers using separable filters [4, 5], tensor decomposition [6, 7], weights and activations quantization [8, 9] and implementation using FFT [10]. In this research we decrease CNN's computational cost using conditional computation, and we hence focus on this literature.

Combining conditional computation with networks is often non-trivial since it makes the training process difficult. Despite this, the usage of conditional computation during train and test time has been studied using several approaches. A CNN that deals with dynamic time budget was suggested in [11], allowing output estimation without completing the entire forward propagation process, using additional loss layers in earlier stages of the network. Although an early classification is obtained, the accuracy decreases significantly when having low time budget.

The model described in [12] suggests selection of a sub-networks combination located between stacked LSTMs, in an input-depended fashion. This model aims to increase its number of parameters using these sub-networks, thus allow handling tasks with many parameters, such as language modeling. This model does not save computation on tasks such as image classification considered in the present paper.

Another suggested model is a recurrent neural network which selectively processes only some regions of the input, using reinforcement learning methods [13]. Also in [14] reinforcement learning technique is used to train a fully connected neural network to drop neurons in an input-depended manner. These models use sparse tensors which are less compliant with hardware constrains, causing computational saving to be less efficient.

In our model we trained a CNN with bypass connections in an input depended manner, such that only the necessary feature-maps are computed. Our results show that using this approach allows computational saving with significant test time speedup. The method is orthogonal to many methods suggested above [4–7, 10], and hence can be combined with them to obtain further acceleration.

3 Baseline Network – 'DenseNet'

In traditional networks the layers are sequentially connected one after the other. Different studies [15, 16] have shown that networks containing shorter connections (bypasses) enable deeper architectures which are more accurate. One such recent architecture is our network's baseline – DenseNet [17]. While traditional CNNs' layers are serially connected, in DenseNet the layers are sorted in large blocks, each containing multiple convolutional layers. Within each block, convolutional layers (followed by ReLU and Batch Normalization (BN)) forward their output maps to all subsequent convolution layers within the same block. Transition layers (convolution and average-pooling) separate between blocks and decrease the feature map size. Deeper layer in the block hence get as input maps from all their predecessors in the block, so their input size (number of input maps) increases. To reduce the amount of input maps, the output size of all layers is limited to k maps.

Using the DenseNet architecture, each omitted feature map implies significant computation saving, as the map is used as input for all following layers in the block, and not only the one next following layer. Our architecture takes advantage of this insight to reduce computation amount during test time.

4 Model

4.1 Motivation

As part of the efforts to reduce the deep networks' computational cost, our model prunes feature maps input-dependably. A computation of a feature-map can have significant or negligible influence on the output accuracy for a given example, depending on its content. For example, classifying dogs and cats is considerably different from distinguishing trains from tracks, thus these two tasks depend on different feature maps. Motivated by this observation, we created an architecture selecting which maps to compute and which to omit while classifying a given input.

4.2 Architecture

The model is based on the architecture described in [17], and to identify the essential feature-maps of the networks, for a given input to be classified, input depended 'gates' are added. Each gate is associated with a feature-map and indicates if the feature-map computation is necessary or not. These gates are binary valued: '1' implies to the necessity of using the feature-map, '0' implies it is unnecessary. As shown in Fig. 1, to produce these gates we connected to each layer a 'gates branch' whose outputs are k gate values (one for each map). To keep the net's computation's efficiency, the computational cost of the gates branches is low. Average pooling with fixed 4×4 output size is applied on the input layers (all the L preceding layers in the block), diminishing the input size of the following fully-connected (FC) layer. The FC layer outputs k neurons, each associated with one output map (of the $L + 1$ layer). Following

this, BN and sigmoid activation layers are applied, producing probability-like values for map computation. Following the sigmoid layer, a stochastic decision is made regarding map computation by a Bernoulli trail with the probability of '1' provided by the gate. At test time, only the maps whose gate output is '1' are computed.

At training, to apply the gate decision for information flows on the main branch, each output map of the convolutional layer is multiplied by its associated gate value. Therefore, a gate's decision of "not computing a map" causes multiplication of the corresponding output map by 0 and avoiding the unnecessary map, while a gate valued 1 keeps the information of the map unchanged.

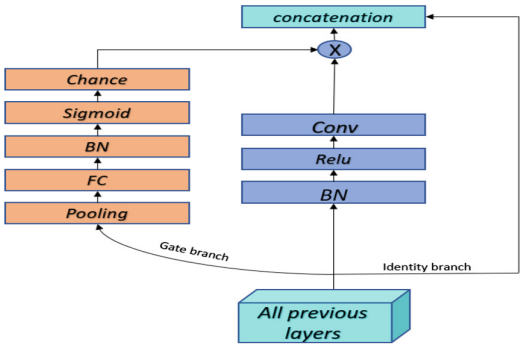

Fig. 1. The unit structure: The unit consists of 3 branches: the main branch, the bypass (identity) branch and the gate branch.

4.3 Optimization

The loss used for training is the standard softmax loss (the gates' actions and probabilities are implicit). During training the weights of the network's main branch (i.e. the DenseNet architecture) are adjusted using standard Stochastic Gradient Descent (SGD).

Since stochastic decisions are made in the gate branch, the introduced discontinuity and non-differentiability prevent optimization of the entire network using SGD. For optimization of the gate branches we use a reinforcement learning derivation as in [18] and minimize the expected loss while taking expectations also with respect to the stochastic decision made. For a single batch with B examples, the loss we minimize to encourage map pruning is

$$L_{Gate} = \frac{\lambda \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=1}^{B} \left(p_{ki}^{l} - t\right)^{Q} - |t|^{Q}}{K \cdot L \cdot B} \tag{1}$$

Where p_{ki}^{l} is the map computation probability of map k of layer l in example i. K is the number of maps in a layer and L is the number of gated layers of the network. λ, t and Q are scalar parameters. Raising the probabilities p_{ki}^{l} to the power $Q > 1$

encourages diversity of p_{ki}^{l} values. The parameter t is used to avoid the derivatives from obtaining very high values approaching $+\infty$, which may happen otherwise for $Q < 1$. This loss is added to the standard Softmax loss to provide the total loss minimized. λ is used to weight the pruning-related loss function and create a balance between accuracy and computational saving.

5 Experiments and Results

5.1 Dataset – CIFAR-10

We evaluate the network on the CIFAR-10 dataset [1], which consists of 60,000 images of 10 categorical classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. All the images are RGB sized 32 × 32. The images are divided to a train set and a test set: 50,000 and 10,000 images respectively. Followed by [17], we used 5,000 images from the training set as a validation set.

5.2 Training

We trained the network using a batch size of 64 examples for 300 epochs on a single GPU. We set the learning rate to 0.1 over the first 150 epochs, then diminish it to 0.01 for the next 75 epochs, and again diminish it to 0.001 for the last 75 epochs. We set the number of output maps of each unit to be $k = 12$ to prevent the network from growing too wide. The layers are sorted in 3 dense blocks with feature-map sized 32 × 32, 16 × 16 and 8 × 8, each block contains 12 BN-Relu-Conv units. Followed by [17], between the blocks we set a sequence of BN-Relu-Conv-Average-pooling with 2 × 2 pool size. The weight decay is set to 0.0001, the momentum is set to 0.9 and Q is set to 6. We initialize all the main branch weights to the final weights of a trained same-sized DenseNet. We removed the drop-out layers from the network, since the map-pruning produces significant training noise, and the additional drop-out noise disturbs the network optimization and convergence.

Gradual Learning
In some experiments we trained the network gradually, with each dense block trained separately at a time for 300 epochs, using the same parameters stated above. First, for 300 epochs, only the gate branches' parameters of the bottom dense block were adjusted. Then, for another 300 epochs, the gate branches' parameters of the second block as well as the first block were adjusted. Finally, the third block's gate branches were added to the process, and the entire network was trained simultaneously for 300 epochs.

5.3 Results

We evaluated the networks on the 10,000 remaining test set images. At test time, a pruning threshold was set to 0.5 and if the gate probability is above this threshold, the map is computed. To evaluate the computational saving potential of our gated version,

note that the computation complexity of a convolutional layer with input size $W \times H \times d_{in}$, output size $W \times H \times K$ and filter size $F \times F$ is

$$O\left(W \times H \times d_{in} \times K \times F^2\right) \tag{2}$$

In our framework, the average probability of a map to be computed is $P = E_{l,i,k}\left[P_{ki}^{l}\right]$. If both the d_{in} input maps and the K output maps are not pruned with probability P, the complexity of convolutional layer computing is

$$O\left(W \times H \times \mathrm{P}d_{in} \times \mathrm{P}K \times F^2\right) \tag{3}$$

And the speedup resulting from dividing (2) by (3) is $1/P^2$, i.e. quadratic in P. Assuming that the pruning probability is approximately invariant across layers, this provides a good estimation of the potential acceleration. We hence compute the expected acceleration as $1/P^2$ where P is estimated by averaging $P_{i,k}^{l}$ over all the maps and all test examples.

Our main results are shown in Table 1. For $\lambda = 0$, i.e. when the map pruning loss is not active, the network prefers to keep almost all its gates at '1', thus using almost all the maps. When λ is raised to 5 our input dependent version can provide significant potential accelerations of up to $\times 3$, with small accuracy drops of up to 1.7%.

We compare the results of our input-dependent version with DenseNet networks containing less maps. In each row, we compare to a DenseNet with the number of maps reduced to get a computational cost comparable to the gated network. This is done by choosing $k_{DenseNet} = k \cdot P$ with P is the gate not-pruning probability. It can be seen that the input dependent versions provide an advantage over the simpler alternatives.

Table 1. Accuracy and potential speedup of gated networks using different train methods and parameters. Together with the initial convolutional layer and the transition layers, the networks depth is L = 40.

Training technique	λ	Bias	Error rate	Average P	Potential speedup	$k_{DensNet}$	Baseline error rate
Standard	0	4	7.2%	99.95%	$\times 1$	12	7.2%
Gradual	5	1	8%	83.82%	$\times 1.42$	10	8.18%
Standard	5	1	8.9%	57.73%	$\times 3$	7	9.34%

Another advantage of a gated network version is that one can further control the speed-accuracy trade-off by tuning the pruning threshold as test time. Hence the same network can provide a certain range of speed-accuracy working points, chosen at test time according to the application needs. This trade-off obtained by the network of row three from Table 1 is shown in Fig. 2.

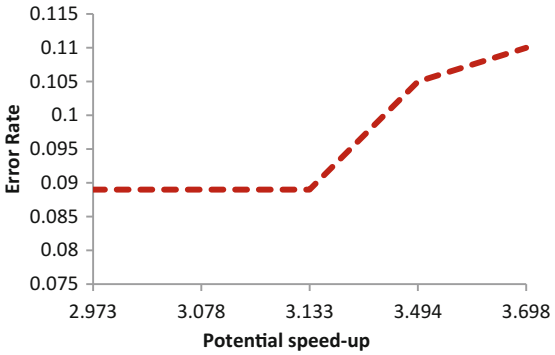

Fig. 2. The potential speed-up and the corresponding error rate using different test threshold values. The results are using a network that was trained with $\lambda = 5$ and bias = 1.

6 Conclusions and Further Work

We have presented an architecture with input dependent gates, enabling partial computation of feature maps in an input dependent manner. We showed that such a gated version provides convenient accuracy to speed trade off, which is slightly preferable to the trade-off obtained with plain DenseNet versions. Beyond that, the gated version allows additional accuracy-speed trade-off at run time, hence enabling further flexibility when computational constraints are present.

We currently work on optimizing the model during training using other techniques and extending the testing to more datasets.

References

1. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report, Computer Science Department, University of Toronto, pp. 1–60 (2009)
2. Erhan, D., Bengio, Y., Courville, A., Vincent, P.: Visualizing higher-layer features of a deep network. Bernoulli **1341**, 1–13 (2009)
3. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization (2015)
4. Rigamonti, R., Sironi, A., Lepetit, V., Fua, P.: Learning separable filters. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2754–2761 (2013)
5. Mamalet, F., Garcia, C.: Simplifying ConvNets for fast learning. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012. LNCS, vol. 7553, pp. 58–65. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33266-1_8
6. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv Preprint. arXiv 1405.3866, p. 7 (2014)
7. Jin, J., Dundar, A., Culurciello, E.: Flattened convolutional neural network for feedforward acceleration. ICLR Work. **2014**, 1–11 (2015)

8. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 1–17. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
9. Vanhoucke, V., Senior, A., Mao, M.: Improving the speed of neural networks on CPUs. In: Proceedings of Deep Learning and Unsupervised Feature Learning NIPS, pp. 1–8 (2011)
10. Mathieu, M., Henaff, M., LeCun, Y.: Fast training of convolutional networks through FFTs. In: International Conference on Learning Representations, pp. 1–9 (2014)
11. Amthor, M., Rodner, E., Denzler, J.: Impatient DNNs - deep neural networks with dynamic time budgets, no. 2 (2016)
12. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Dean, J.: Outrageously large neural networks : the sparsely-gated mixture-of-experts layer, pp. 1–15 (2017)
13. Mnih, V., Heess, N., Graves, A., Kavukcuoglu, K.: Recurrent models of visual attention. Adv. Neural. Inf. Process. Syst. **27**, 1–9 (2014)
14. Bengio, E., Bacon, P.-L., Pineau, J., Precup, D.: Conditional computation in neural networks for faster models, pp. 1–9 (2015)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Arxiv.Org, vol. 7, no. 3, pp. 171–180 (2015)
16. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks importance of identity skip connections usage of activation function analysis of pre-activation structure, no. 1, pp. 1–15 (2016)
17. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks (2016)
18. Willia, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3), 229–256 (1992)