# PRUNENET: CHANNEL PRUNING VIA GLOBAL IMPORTANCE

**Ashish Khetan & Zohar Karnin**
Amazon AWS, {khetan,zkarnin}@amazon.com

## ABSTRACT

Channel pruning is one of the predominant approaches for accelerating deep neural networks. Most existing pruning methods either train from scratch with a sparsity inducing term such as group lasso, or prune redundant channels in a pretrained network and then fine tune the network. Both strategies suffer from some limitations: the use of group lasso is computationally expensive, difficult to converge and often suffers from worse behavior due to the regularization bias. The methods that start with a pretrained network either prune channels uniformly across the layers or prune channels based on the basic statistics of the network parameters. These approaches either ignore the fact that some CNN layers are more redundant than others or fail to adequately identify the level of redundancy in different layers. In this work, we investigate a simple-yet-effective method for pruning channels based on a computationally light-weight yet effective data driven optimization step that discovers the necessary width per layer. Experiments conducted on ILSVRC-12 confirm effectiveness of our approach. With non-uniform pruning across the layers on ResNet-50, we are able to match the FLOP reduction of state-of-the-art channel pruning results while achieving a $0.98\%$ higher accuracy.

## 1 INTRODUCTION

The recent works on model compression can be divided into four main categories, namely, quantization Courbariaux et al. (2015); Rastegari et al. (2016); Hubara et al. (2017), low rank factorization Jaderberg et al. (2014); Lebedev et al. (2014); Tai et al. (2015), sparse connections Han et al. (2015a;b), and structured sparsity such as channel pruning Wen et al. (2016); Luo et al. (2017); Liu et al. (2017); Zhuang et al. (2018). Network quantization aims to reduce the model size and accelerate the inference by reducing precision of the network weights. Many of the modern computing devices support faster inference for the low precision networks. Low rank factorization approximates the convolutional kernels using tensor decomposition techniques. Sparse connections seeks to sparsify the network by pruning low importance weights in the model. We note that unstructured sparsity may lead to a reduction in the number of parameters, but typically will not accelerate the network due to its inherent need to access memory in a non-consecutive way. In contrast, channel pruned network has exactly the same architecture and back-end implementation but with fewer filters and channels. Hence it immediately yields smaller memory footprint and faster inference than the original model without requiring any additional hardware or software support. Further, channel pruning is complementary to network quantization and it is known that both can be applied together to achieve higher compression than any method individually Han et al. (2015a).

## 2 APPROACH

We begin by formally characterizing a CNN and its parameters. For readability we assume the structure of Convolution-BatchNorm-Activation and a ReLU activation. We note that our framework applies without this assumption, meaning for any order and for any activation type. Let $L$ denote the number of convolution operators in the network, and for $\ell \in [L] = \{1, \ldots, L\}$ let $W_\ell \in \mathbb{R}^{n_\ell \times m_\ell \times k_\ell \times k_\ell}$ denote the filter weights of the $\ell$-th convolution. Here, $n_\ell$ and $m_\ell$ represent the number of output and input channels respectively, and $k_\ell \times k_\ell$ is the size of each filter. A BatchNorm (BN) layer has the same input and output size. Denote by $z^{(in)} \in \mathbb{R}^{n \times r \times r}$ and $z^{(out)} \in \mathbb{R}^{n \times r \times r}$ its

input and output respectively. Here, the $z$ variables contain $n$ channels, each assumed to have $r \times r$ features[1]. Let $\mathcal{B}$ denote the current mini-batch, a standard BN layer performs the following affine transformation for each $i$-th feature map $z_i \in \mathbb{R}^{r \times r}$, for $i \in \{1, 2, \cdots, n\}$.

$$\hat{z}_i = \frac{z_i^{(in)} - \mu_{\mathcal{B}_i}}{\sqrt{\sigma_{\mathcal{B}_i}^2 + \epsilon}}; \quad z_i^{(out)} = \gamma_i \hat{z}_i + \beta_i \,, \tag{1}$$

where $\mu_{\mathcal{B}_i}$ is the scalar mean of the entire $i$-th feature map over the mini-batch $\mathcal{B}$, and $\sigma_{\mathcal{B}_i}^2$ is the scalar variance. Although individual numbers within the feature map $\hat{z}_i$ do not have a zero mean and unit variance, each feature map $\hat{z}_i$, when considered as a $r^2$-dimensional vector, has unit norm and sums to zero, in expectation. With this in mind, we say that $\gamma_i^2$ controls the variance of the $i$-th output channel $z_i^{(out)}$, and $\beta_i$ is its mean. When doing inference, the BN layer behaves slightly differently. Rather than taking the mini-batch statistics $(\mu_{\mathcal{B}_i}, \sigma_{\mathcal{B}_i}^2)$, it uses their global counterparts $(\mu, \sigma^2)$ obtained not from a single mini-batch but from the entire dataset (or sufficiently large sample). Since each convolution is followed by a BN layer, we let $\gamma_{\ell,i}$, $\beta_{\ell,i}$ represent the $i$-th scale and bias parameters of the $\ell$-th convolution layer. For the entire network, let $W \equiv \{W_\ell\}_{\{1,2,\cdots,L\}}$ denote the set of all convolution parameters, and let $\gamma = \{\gamma_{\ell,i}, \beta_{\ell,i}\}_{\ell,i}$ denote the parameters of the BN layers following the convolutions. Denote all other network parameters by $F$. This could include fully connected layers etc.

**Global Importance Score:** We are now ready to explain our approach and the motivation leading to it. We first observe that if we ignore the effect of the activations, the $i$-th input channel of the $\ell$-th convolution layer has a variance of $\gamma_{\ell-1,i}^2$. Therefore, the contribution of the $i$-th input to the variance of the $j$-th output in the $\ell$-th convolution layer is $\gamma_{\ell-1,i}^2 \|W_{\ell,i,j}\|_2^2$, where $W_{\ell,i,j}$ is the filter in the $\ell$-th convolution layer corresponding to the $i$-th input and the $j$-th output. Indeed if the filters and input feature maps were of dimension 1, then the convolution would be reduced to a linear function. In this simplified case where the convolution is reduced to a linear function, if we wanted to ignore one of the inputs and minimize the squared distance of the output change for the $j$-th output channel, the importance of the inputs would exactly be determined by $\gamma_{\ell-1,i}^2 \|W_{\ell,i,j}\|_2^2$. When considering all outputs simultaneously the score becomes $\gamma_{\ell-1,i}^2 \sum_{j=1}^{n_\ell} \|W_{\ell,i,j}\|_2^2$. We let this score be the global importance score of the $i$-th input channel to the $\ell$-th convolution layer. Now, since the ReLU activation is linear, and as detailed before, when applying sparsity regularization we do not modify the convolution weights $W$, rather we multiply the convolution weights with a scalar such that $\sum_{j=1}^{n_\ell} \|W_{\ell,i,j}\|_2^2 = 1$ and shift that scalar into $\gamma_{\ell-1,i}$ and $\beta_{\ell-1,i}$. For this reason, when we discuss the details of our implementation we will assume that the importance score is simply $\gamma_{\ell-1,i}$.

**Regularization and Importance Scores:** Given this Global Importance Score, a naive approach for pruning channels of a pretrained network would be to rank all the channels according to their score and prune the least important ones. However, a sophisticated approach would be train a model while regularizing the cost associated with each channel and then prune the low importance channels. Let's proceed to define our optimization objective. First, denote by $\mathcal{L}\Big(f\big(x; \{W, F, \gamma, \beta\}; \{\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}^2\}\big), \;\; y\Big)$, the loss of the network during a standard training. The most straightforward objective when considering channel pruning is

$$\mathcal{L}\Big(f\big(x; \{W, F, \gamma, \beta\}; \{\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}^2\}\big), \;\; y\Big) + \lambda \sum_{\ell > 1, i} \alpha_\ell \mathbf{1}[\gamma_{\ell-1,i} > 0]$$

Here, a channel with $\gamma_{\ell,i} = 0$ is considered pruned even if $\beta_{\ell,i} \neq 0$, since the bias term can be simulated by modifying the other network parameters. We let $\alpha_\ell$ denote the cost associated with each input channel of the $\ell$-th convolutional layer. The value of the $\alpha$ costs are determined by our objective. For example, for FLOPs minimization we compute the number of FLOPs needed for the full network, and for the network after pruning an output of the $(\ell - 1)$-th layer which is also an input to the $\ell$-th layer. This difference determines the scale of $\alpha_\ell$. The objective of channel pruning can be any of the following: minimize the model size (# params), reduce the # FLOPs required for inferencing an image, or minimize latency on a given hardware (GPU/CPU). We choose the costs $\alpha_\ell$

[1]This assumption is purely for readability. Our framework works for 1d or 3d convolution as well, with any feature map size.

according to the objective. Since $\ell_0$ norm is non-smooth and has zero gradient almost everywhere, we consider the standard continuous relaxation of it and use the $\ell_1$ norm [2] which is well known to induce sparsity. Therefore, we seek to minimize the following objective function.

$$\mathcal{L}\Big(f\big(x;\{W,F,\gamma,\beta\};\{\mu_{\mathcal{B}},\sigma^2_{\mathcal{B}}\}\big),\ \ y\Big)\ +\ \lambda\sum_{\ell,i}\alpha_\ell|\gamma_{\ell,i}|\,. \tag{2}$$

**Optimize with BatchNorm off:** An immediate approach would be to optimize the regularized objective in equation 2 by training all the model parameters $W, F, \gamma, \beta$ with random initialization. Recall $(\mu_{\mathcal{B}}, \sigma^2_{\mathcal{B}})$ are the mini-batch mean and variance values, not the trainable model parameters. However, a closer investigation of BatchNorm layer, equation 1, reveals that in doing so the impact of regularizing $\gamma_{\ell,i}$ would be nullified by the normalization operation in the BatchNorm layer. Particularly, the regularization term $\sum_{\ell,i}\alpha_\ell|\gamma_{\ell,i}|$ can be pushed to any arbitrary positive value $\epsilon > 0$ without changing the network output and the loss function $\mathcal{L}(\cdot)$. It can be seen that for any given scalar $\tau > 0$ there exists $\widetilde{\gamma} = \tau\gamma$, $\widetilde{\beta} = \tau\beta$, $\widetilde{\mu}_{\mathcal{B}} = \tau\mu_{\mathcal{B}}$, and $\widetilde{\sigma}^2_{\mathcal{B}} = \tau^2\sigma^2_{\mathcal{B}}$, such that

$$\begin{aligned}&\mathcal{L}\Big(f\big(x;\{W,F,\gamma,\beta\};\{\mu_{\mathcal{B}},\sigma^2_{\mathcal{B}}\}\big),\ \ y\Big)\\ =\ \ &\mathcal{L}\Big(f\big(x;\{W,F,\widetilde{\gamma},\widetilde{\beta}\};\{\widetilde{\mu}_{\mathcal{B}},\widetilde{\sigma}^2_{\mathcal{B}}\}\big),\ \ y\Big)\,.\end{aligned}$$

Note that when the $\gamma$ and $\beta$ are reduced by a factor of $\tau$ then by definition, equation 1, the mean and variance of feature maps change to $\widetilde{\mu}_{\mathcal{B}}$ and $\widetilde{\sigma}^2_{\mathcal{B}}$. Therefore, we propose optimizing the regularized objective equation 2 by turning the BatchNorm layer off. We do so by fixing the BatchNorm mean and variance values $(\mu_{\mathcal{B}}, \sigma^2_{\mathcal{B}})$ to their global counterparts $(\mu, \sigma^2)$. Note that during inference also $(\mu_{\mathcal{B}}, \sigma^2_{\mathcal{B}})$ are set to $(\mu, \sigma^2)$. Further, the convolutional weight parameters $W$ also have the same impact on the regularization term; $\gamma$ and $\beta$ can be reduced by a factor of $\tau$ while keeping the network output same by increasing $W$ by a factor of $1/\tau$. Therefore, we start with a pretrained model and fix the convolution weight parameters $W$ to their value $W^*$ in the pre-trained network. We minimize the following objective to induce sparsity in $\gamma$.

$$\mathcal{L}\Big(f_{(W^*,\mu,\sigma^2)}\big(x;\{F,\gamma,\beta\};\big),\ \ y\Big)\ +\ \lambda\sum_{\ell,i}\alpha_\ell|\gamma_{\ell,i}|\,. \tag{3}$$

Note that the objective is only a function of BatchNorm scale parameters $\gamma, \beta$ and the fully connected layer parameters $F$. Since $F$ is typically a single fully connected layer, the overall number of parameters to be optimized is small and a single epoch suffices. Note that Liu et al. (2017) follows the approach of optimizing the objective in equation 2 by training all the model parameters $W, F, \gamma, \beta$ with random initialization. However, as explained above this approach nullifies the effect of regularizing $\gamma$, and requires a long time to converge due to the large number of parameters in $W$.

**Step-wise Pruning of Channels:** The step of optimizing equation 3 leaves us with a ranking of the channels based on their Importance Score $\gamma$. We prune sufficiently many channels to reduce the total cost $C = \sum_{\ell,i}\alpha_\ell$ to $\eta C$, for a given fraction $\eta$. By prune we mean we set their $\gamma$ value to zero if it wasn't already zero, and move on to the next phase. In the next phase we fix the bias suffered by the sparsity inducing regularization and optimize

$$\mathcal{L}\Big(f_{W^*}\big(x;\{F,\gamma,\beta\};\{\mu_{\mathcal{B}},\sigma^2_{\mathcal{B}}\}\big),\ \ y\Big), \tag{4}$$

the same objective as in equation 3 but without the regularization term. In this step, we use the mini-batch mean and variance values $(\mu_B, \sigma^2_B)$ for BatchNorm layers. Also, we compute the updated values of their global counterparts $(\mu, \sigma)$ which are used in the next iteration of the regularization, equation 3. Here, we abused notation and use $\gamma$ to denote the set of non-pruned weights. The pruned weights are fixed as zeros. As before, since the number of parameters here is small, a single epoch typically suffices to solve the optimization problem. Finally, we alternately repeat optimizing objectives equation 3 and equation 4 for $T$ times. In each step we reduce the cost by $(\eta/T)C$ for a desired fraction of cost reduction $\eta$. We optimize the hyper-parameters associated with these two steps in a way that minimizes the loss of Equation equation 4. Since the overall procedure is lightweight, exploring hyper-parameters is much cheaper compared to those relevant to a full training job. Once we obtained the final pruned model, we fine tune all of its weights, including $W$, by optimizing the original objective with the appropriate $\gamma$ values being fixed as zeros.

[2] We are aware of papers that use various non-convex regularization. We postpone exploring the benefit of those to future works, so as to minimize the number of unknowns

| | # Params | # FLOPs | Top-1 | Top-5 |
|---|---|---|---|---|
| DCP | 1.51× | 1.56× | **−0.39** | **−0.14** |
| Uniform | 1.57× | 1.55× | +0.57 | +0.18 |
| GCP-f | 1.12× | 1.55× | +0.38 | +0.17 |
| ThiNet | 2.06× | 2.25× | +1.87 | +1.12 |
| DCP | 2.06× | 2.25× | +1.06 | +0.61 |
| CP | _ | 2 × | _ | +1.40 |
| Uniform | 2.26× | 2.24× | +1.37 | +0.74 |
| GCP-f | 1.32× | 2.24× | **+1.02** | **+0.56** |
| DCP | 2.94× | 3.47× | +3.26 | +1.80 |
| Uniform | 3.51× | 3.46× | +3.10 | +2.14 |
| GCP-f | 2.18× | 3.47× | **+2.28** | **+1.35** |
| Uniform | 5× | 5× | +5.08 | +2.8 |
| GCP-f | 3.09× | 5× | **+3.86** | **+2.06** |
| Uniform | 10× | 10× | +10.05 | +5.89 |
| GCP-f | 5.75× | 10× | **+8.16** | **+4.89** |

| | # Params | # FLOPs | Top-1 | Top-5 |
|---|---|---|---|---|
| DCP | 1.51× | 1.56× | **−0.39** | **−0.14** |
| GCP-p | 1.51× | 1.13× | −0.05 | −0.19 |
| ThiNet | 2.06× | 2.25× | +1.87 | +1.12 |
| DCP | 2.06× | 2.25× | +1.06 | +0.61 |
| GCP-p | 2.08× | 1.32× | **+0.55** | **+0.20** |
| DCP | 2.94× | 3.47× | +3.26 | +1.80 |
| GCP-p | 2.95× | 1.53× | **+1.52** | **+0.65** |

Table 1: Comparisons on ILSVRC-12 for ResNet-50. The top-1 and top-5 error % of the pre-trained model are 22.81 and 6.47 respectively. "_" denotes that the results are not reported. # Params and # FLOPs represent multiplicative reduction from the original ResNet-50.

## 3 EXPERIMENTS

In this section, we empirically evaluate the performance of our Global Channel Pruning (GCP) algorithm. The GCP operates on a pretrained model and requires two inputs: the minimization objective, for e.g. minimize # FLOPs, and the target fraction, $\eta$. The target fraction $\eta$ is the ratio of the desired objective value in the pruned network and the objective value in the original network. Note that we chose the regularization scale $\alpha_\ell$ for each channel $\ell \in \{1, 2, \cdots, L\}$ according to the objective we seek to minimize. We refer GCP by GCP-p when $\alpha_\ell$ are chosen to minimize the # params and by GCP-f when $\alpha_\ell$ are chosen to minimize # FLOPs. We refer GCP by GCP-$\ell$ when $\alpha_\ell$ are chosen to minimize the latency. Our method can also be used to maximize throughput as it is the inverse of latency.

We evaluate GCP on state-of-the-art ResNet models for ILSVRC-12 (ImageNet) and Caltech-256 datasets. We compare performance of GCP against several well known channel pruning methods, including Discrimination Aware Channel Pruning (DCP) Zhuang et al. (2018), ThiNetLuo et al. (2017), and Channel Pruning (CP) He et al. (2017). To the best of our knowledge, these are the state-of-the-art works that report channel pruning results on ResNet models for ImageNet dataset. These works set a target fraction $\eta$ for pruning channels and uniformly prune the $\eta$ fraction of channels from each layer. However, since the true objective of channel pruning is to reduce the # params and # FLOPs, they report these numbers as well. To fairly compare with these works we run GCP-f with the target fraction $\eta$ equal to the FLOPs reduction reported in these works, and run GCP-p with $\eta$ set to the parameter reduction reported in these works. Besides, to investigate the effect of the non-uniform pruning achieved by GCP, we study the uniform pruning approach wherein we prune each layer uniformly. We prune $\eta$ fraction of channels from each layer in the pretrained model and retrain the pruned model.

**ResNet-50:** Table 1 (left) gives the results of channel pruning of ResNet-50 for various FLOPs reduction targets. For example, the numbers in the eleventh row correspond to the results of GCP-f for # FLOPs minimization with target fraction $\eta = 1/3.47$. It returns a pruned model that has $1/2.18$ fewer parameters, $2.28\%$ higher top-1 and $1.35\%$ higher top-5 error rate than the original pretrained model. Compared to this, DCP achieves the same # FLOPs reduction at $3.26\%$ increase in top-1 error rate. Further, the last four rows show the gain of GCP-f over uniform pruning in the extreme setting of $5\times$ and $10\times$ FLOPs reduction. Table 1 (right) gives similar results for the case of GCP-p applied on ResNet-50 for minimizing the model parameters. In the high pruning regime, GCP-p significantly outperforms the DCP Zhuang et al. (2018). We provide experimental results on pruning ResNet-18 and ResNet-101 on ImageNet, and ResNet-50 on Caltech-256 in the Appendix.

## REFERENCES

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2270–2278, 2016.

Miguel A Carreira-Perpinán and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8532–8541, 2018.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pp. 177–185, 1989.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *NeurIPS*, pp. 2164–2174, 2018.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2017.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2755–2763. IEEE, 2017.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 455–462. IEEE, 2017.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 883–894, 2018.

# A APPENDIX

Due to space limitation in the main paper, we provide a summary of related works and conclusion of our work, followed by additional experimental results in this Appendix.

## A.1 RELATED WORKS

There have been a significant amount of work on compressing and accelerating deep CNNs. Most of these works fall primarily into one of the four categories: quantization, low rank factorization, sparse connections, and structured pruning.

Quantization approaches do not reduce the number of parameters but rather their precision, meaning the number of bits representing each parameters. This is done by quantizing the parameters to binary (Rastegari et al., 2016; Hubara et al., 2017), ternary (Zhu et al., 2016), or 4 or 8 bits per parameter (Han et al., 2015a; Gong et al., 2014). Low rank factorization techinques exploit various low rank structures in the convolution parameters, such as a basis for the filters (Jaderberg et al., 2014), or a tensor low rank decomposition (Lebedev et al., 2014; Tai et al., 2015). We note that the practical compression and the acceleration achieved with these low-rank approaches is curretnly limited as the standard deep learning libraries do not support convolutional operation by low rank weight tensors.

Different methods have been explored to prune the model parameters. The early work by (Hanson & Pratt, 1989) performed magnitude based pruning and (Hassibi & Stork, 1993) suggested Hessian matrix based approach to prune the network weights. More recently, (Han et al., 2015b) introduced an iterative method to prune weights in deep architectures, together with an external compression by quantization and Huffman encoding Han et al. (2015a). The network is pruned by removing low weight connections, followed by fine-tuning to recover its accuracy. Training with sparsity constraints has also been studied by (Srinivas et al., 2017; Wen et al., 2016) to achieve higher compression. (Hu et al., 2016) extends parameter pruning and seeks to identify the optimal number of parameters that should be pruned in each layer to minimize the loss in accuracy for a given budget on the pruning. They achieve this by using a measure of average percentage of zeros (APoZ) for each layer.

Existing works in channel pruning can be divided into two categories: training from scratch and pruning redundant channels in a pretrained network. The first type Wen et al. (2016); Alvarez & Salzmann (2016); Liu et al. (2017) trains the network with regularization terms aimed to induce sparsity along the channels. This approach has several shortcomings. First, by requiring full training it does not apply to situations where we already have a trained model and we aim to compress it in a lighter procedure involving either less computation or less data. Second, the sparsity inducing regularization can often insert bias that degrades performance. This issue can be somewhat mitigated by considering non-convex regularization or some iterative pruning approach (see e.g. Carreira-Perpinán & Idelbayev (2018)); this fix however comes at the expense of a complicated system that requires a longer training time due to new hyper parameters to be optimized[3], or the need for multiple training rounds when pruning repeatedly without sparsity inducing regularization. Another issue comes up in networks that use Batch Norm Liu et al. (2017). In all recent architectures a batch-norm layer is present after every convolution. The normalization operation done by this layer can completely modify the effect of a regularization term, as the magnitude of the parameters does not change the output of the model. For example, Hoffer et al. (2018) show that the $\ell_2$ regularization term's effect can be mimicked with a learning rate schedule, meaning that the 'weight-decay' parameter helps more as a learning rate scheduler as opposed to a regularization parameter. This result along with additional papers cited within suggest that a naive combination of sparsity inducing regularization combined with batch-norm may not have the desired effect.

The second type of channel pruning works that start with a pretrained model can be further sub-categorised into two. First sub-category of works prune channels uniformly in each layer using reconstruction based approaches Luo et al. (2017); He et al. (2017). The high level mode of operation is to obtain a sample of inputs and outputs of a convolutional layer, then learn a layer with lesser input channels that produce approximately the same output. The measure of approximation is typically in $\ell_2$ norm but can be more sophisticated, see e.g. Zhuang et al. (2018). The main drawback of these methods comes from the fact they handle convolutional layers one at a time. As such, they cannot

[3] non-convex regularization typically has more than one parameter associated with it, informally determining how far away it is from L1 regularization

detect correlations occurring across layers, and more important, it is hard to adapt these methods to prune one layer more aggressively than another. It follows that these works are inherently limited as they prune the same fraction of channels across all the layers. Our methods can in fact be seen as complimentary to these results, as we provide a light-weight scheme that among others, discovers the extent by which the layers should be pruned. The second sub-category of works prune channels in a pretrained model based on the network parameters and their basic statistics. (Li et al., 2016) prunes channels based on the $\ell_1$ norm of the filters, and (Hu et al., 2016) defines redundancy of neurons based on their activations and prunes the more redundant ones. In contrast to these methods, our method is optimization driven. Instead of identifying redundancy of channels based on parameter values of the pretrained model we optimize over parameters which correlate with the redundancy of the channels. Therefore our method encompasses these existing methods and improves upon them.

Our paper presents a channel pruning technique that enjoys the global view and precise pruning of the full training techniques, while requiring a short training time for the pruning procedure, similarly to the techniques that prune based on basic statistics of the weights. In contrast to the papers described above, the importance of channels is based on the motivating example of sparse linear regression, takes into account the affects of BatchNorm, and most importantly, is based on a short (handful of epochs) training procedure rather than the model weights or a few basic statistics of the data. Our claim that this technique achieves a more accurate importance ranking for the channels is backed up by our experiments achieving state-of-the-art results on ResNet architectures on the ImageNet dataset.

### A.2 Summary of Results

In this work, we aim to investigate simple yet effective methods to perform channel pruning in the entire network across the layers, allowing the possibility of pruning more channels in one layer compared to others. This freedom gives our results a flavor of architecture search, as the resulting model's architecture is based on the dataset rather than an educated guess. We compare the pruned ResNet models with the next available standard smaller model. Figure 1 shows a scatter plot of various models. The x-axis represents throughput and the y-axis corresponds to accuracy. The grey colored dots represent our PruneNets, the networks obtained by pruning channels of ResNet-50. The base models and their training script have been taken from MXNET GluonCV GitHub repository. It can be seen that the PruneNets (pruned ResNet-50) significantly outperform ResNet-34, ResNet-18 and MobileNet-1.0. Pruned ResNet-101 (light pink color) outperforms ResNet-50 (black color). The light blue color dot on the extreme right represents pruned ResNet-34. The pruned ResNet models are available in GluonCV repository and their performance can be seen on accuracy-vs-throughput plot on its webpage `https://gluon-cv.mxnet.io/model_zoo/classification.html`.

In Figure 1, we have pruned channels to maximize throughput on a batch-size of 64 for NVIDIA V100 GPU. However, in general the objective of pruning channels can vary. We may want to minimize #FLOPs or memory footprint or maximize throughput for a different batch-size on a different CPU/GPU machine. Our channel pruning approach takes the particular objective into consideration and accordingly gives different pruning pattern across the layers.

Figure 2 and Figure 3 show the channel pruning patterns achieved by our approach when the objective is to minimize #FLOPs ($3.47\times$ reduction) and when the objective is to minimize the number of parameters ($2.95\times$ reduction) respectively. When the objective is to minimize the #FLOPs, Figure 2, the layers closer to the input are pruned aggressively whereas when the objective is to minimize the #Params, Figure 3, the layers closer to the output are pruned heavily. This very different pruning pattern depending upon the pruning objective can be understood by the fact that the layers closer to the input are more compute intensive (size of feature map is larger in the layers closer to the input) whereas the layers closer to the output are more parameter intensive (number of channels is larger in the layers closer to the output). The accuracy of these pruned networks are given in the eleventh row of Table 1 in Section 3.

We note that most of the existing channel pruning works uniformly prune the channels across all the layers, or minimize the number of channels in the network ignoring the real objective (reducing FLOPs/Params) of the channel pruning.

In Section 2 we explain our channel pruning approach in detail. In Section 3 we provide experiments on several ResNet architectures, on the ILSVRC-12 (ImageNet) dataset. Since the existing channel

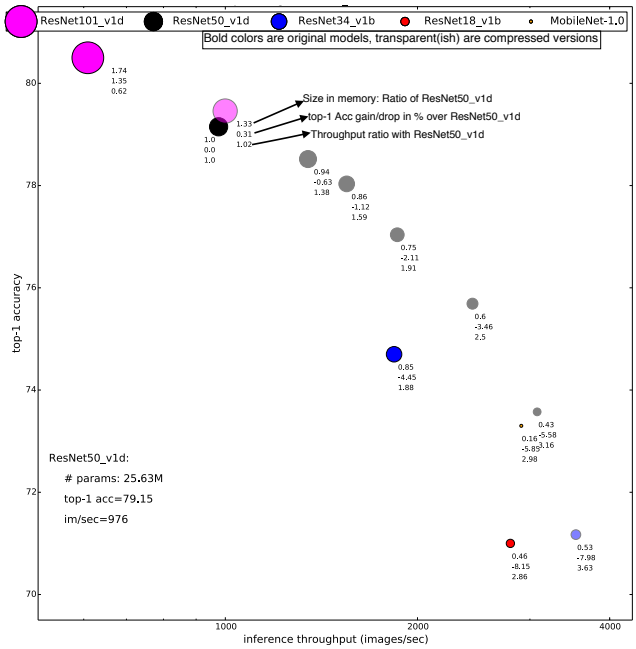


Figure 1: The black circle corresponds to ResNet-50 model. It has 25.63 million parameters, 79.15% top-1 accuracy on ImageNet dataset, and it gives a throughout of 1060 images/sec for batch size of 64 on a NVIDIA V100 GPU. The other colored circles represent the other state-of-the-art models. The grey circles represent our PruneNets. The three numbers associated with each of these circles show the following: the first number gives size of the model (multiplicative w.r.t. ResNet-50), the second number gives loss in accuracy w.r.t. ResNet-50, and the third number gives multiplicative gain in throughput w.r.t. ResNet-50.

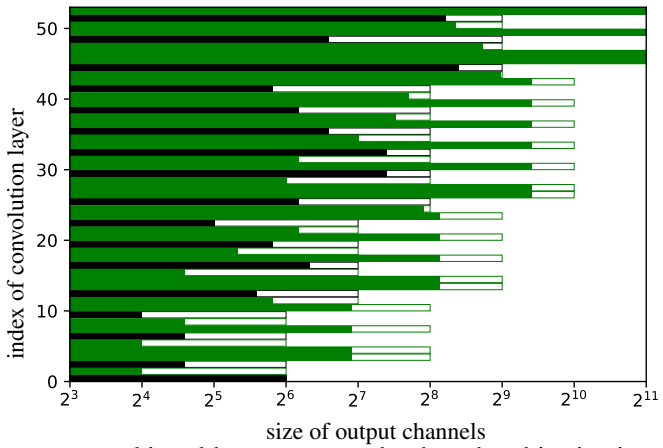


Figure 2: Pruning pattern achieved by our approach when the objective is to minimize #FLOPs. Shallow layers are pruned more aggresively. The black bars represent $3 \times 3$ convolution layers and the green bars represent $1 \times 1$ convolution layers. The white colored area in each bar represents pruning.

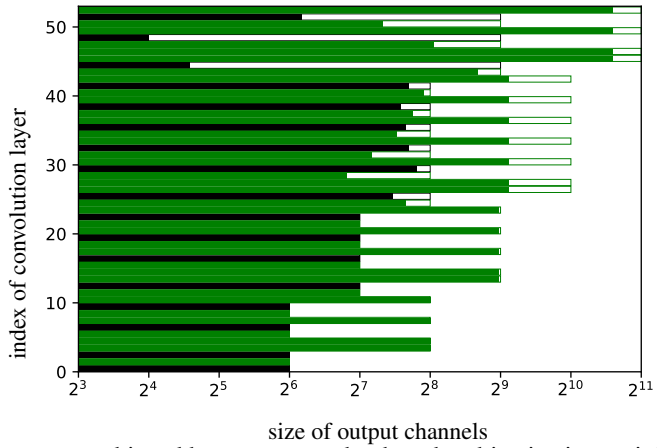


Figure 3: Pruning pattern achieved by our approach when the objective is to minimize #Params. Deep layers are pruned more aggresively. The black bars represent $3 \times 3$ convolution layers and the green bars represent $1 \times 1$ convolution layers. The white colored area in each bar represents pruning.

pruning works only report reduction in memory footprint and FLOPs count, for comparison purpose we restrict to these two objectives. The resulting PruneNets achieve state-of-the-art performance when compared to other pruned models reported in the literature. In order to test the resulting model we assess not only its performance on a test set, but its applicability in other tasks. We use a common transfer learning task and measure how a pruned ResNet-50 model performs when fine tuned on the Caltech-256 dataset. The performance of the pruned model is comparable and in some settings even better than that of the original ResNet-50 model.

### A.3 Discussion

We note that our choice of DCP Zhuang et al. (2018) as the main baseline is due to the fact that they achieve the state-of-the-art results of compressing ResNet50 on ImageNet via channel pruning, and not due to our method being an alternative to theirs. We in fact view our method as complementary the theirs in that it provides a clean way to learn the required width of each layer. It is in fact reasonable to assume that the specialized manner in which they optimize the weights of their network is advantageous compared to ours and that accounts for their superiority in the regime of mild pruning. However, once we require a more aggressive pruning, our method is superior due to increased gain from non-uniform pruning. An interesting follow up would be a method combining the techniques, using our methods for attaining the architecture, combined with alternative methods for tuning the weights of the new network.

### A.4 Comparisons on ILSVRC-12

ILSVRC-12 (ImageNet) contains 1.28 million training images and 50 thousand testing images for 1000 classes. This is one of the most widely used datasets for evaluating performance of CNNs. Further, networks trained on ImageNet are commonly used for transfer learning on smaller datasets such as Caltech-256. We show effectiveness of GCP on three different ResNet architectures, namely ResNet-18, ResNet-50 and ResNet-101. On all the three networks, in the high pruning regime, GCP outperforms the baselines models with a significant margin. In the low pruning regime, its performance is comparable to the baseline models. In all the settings, it performs better than the naive uniform pruning.

### A.5 ResNet-18

Table 2 gives the results of channel pruning of ResNet-18 for various FLOPs reduction targets. For example, the numbers in the sixth row correspond to the results of GCP-f for # FLOPs minimization with target fraction $\eta = 1/1.87$. It returns a pruned model that has $1/1.19$ fewer parameters, $1.70\%$ higher top-1 and $1.06\%$ higher top-5 error rate than the original pretrained model. Compared to this, DCP achieves the same # FLOPs reduction at $2.29\%$ increase in top-1 error rate. Note that since

| ResNet-18 | # Params. ↓ | # FLOPs ↓ | Top-1 | Top-5 |
|---|---|---|---|---|
| DCP | 1.39× | 1.37× | **+0.43** | **+0.12** |
| Uniform | 1.42× | 1.35× | +1.31 | +0.67 |
| GCP-f | 1.03× | 1.36× | +0.45 | +0.32 |
| DCP | 1.89× | 1.85× | +2.29 | +1.38 |
| Uniform | 1.92× | 1.85× | +3.01 | +1.90 |
| GCP-f | 1.19× | 1.87× | **+1.70** | **+1.06** |
| DCP | 2.92× | 2.79× | +5.52 | +3.30 |
| Uniform | 2.76× | 2.79× | +6.75 | +4.2 |
| GCP-f | 1.53× | 2.81× | **+4.34** | **+2.70** |

Table 2: Comparisons on ILSVRC-12 for ResNet-18. The top-1 and top-5 error % of the pre-trained model are 29.21 and 10.13 respectively. The proposed method GCP-f incurs 1.18% lower error rate than the DCP method for the same 2.81× reduction in # FLOPs.

| ResNet-18 | # Params. ↓ | # FLOPs ↓ | Top-1 | Top-5 |
|---|---|---|---|---|
| DCP | 1.39× | 1.37× | **+0.43** | **+0.12** |
| GCP-p | 1.41× | 1.09× | +0.71 | +0.31 |
| DCP | 1.89× | 1.85× | +2.29 | +1.38 |
| GCP-p | 1.89× | 1.21× | **+1.84** | **+1.18** |
| DCP | 2.92× | 2.79× | +5.52 | +3.30 |
| GCP-p | 2.97× | 1.38× | **+5.21** | **+2.84** |

Table 3: Comparisons on ILSVRC-12 for ResNet-18. The top-1 and top-5 error % of the pre-trained model are 29.21 and 10.13 respectively. In the high pruning regime, the proposed method GCP-p outperforms the DCP.

GCP-f prunes channels non-uniformly, its parameter reduction is not same as its FLOPs reduction unlike the case of naive Uniform pruning and the DCP approach which prunes channels uniformly.

Table 3 gives similar results for the case of GCP-p applied on ResNet-18 for minimizing the model parameters. Again, in the high pruning regime, GCP-p significantly outperforms the DCP Zhuang et al. (2018).

### A.6 RESNET-101

Table 4 gives the results of channel pruning of ResNet-101 network for various FLOPs reduction targets. For both the FLOPs reduction target values of $\eta = 1/2$ and $\eta = 1/3.35$, GCP-f outperforms the naive uniform pruning approach. Figure 1 in Section A.2 shows the results of GCP-$\ell$ applied on ResNet-101 for maximizing the throughput for a batch-size of 64 on a NVIDIA V100 GPU, and results in a network both faster and more accurate than ResNet-50.

### A.7 COMPARISONS ON CALTECH-256

In the following we show that the pruned ResNet models when used for transfer learning on smaller dataset such as Caltech-256 performs comparable to the original model. Caltech-256 contains 30 thousand images for 256 classes. We resize the top fully connected layer of the pruned ResNet network to match the output size of 256 classes and randomly initialize its weights. For training and testing we follow the standard protocol. We sample 60 images from each class as the training

| ResNet-101 | # Params. ↓ | # FLOPs ↓ | Top-1 | Top-5 |
|---|---|---|---|---|
| Uniform | 2.03× | 2× | +0.95 | +0.34 |
| GCP-f | 1.56× | 2× | **+0.68** | **+0.18** |
| Uniform | 3.59× | 3.35× | +2.35 | +0.75 |
| GCP-f | 2.12× | 3.35× | **+2.11** | **+0.54** |

Table 4: Comparisons on ILSVRC-12 for ResNet-101. The top-1 and top-5 error % of the pre-trained model are 19.57 and 4.94 respectively.

| ResNet-50 | # Params. ↓ | # FLOPs ↓ | Top-1 | Top-5 |
|---|---|---|---|---|
| GCP-f | $1.12\times$ | $1.55\times$ | $-0.78$ | $-0.31$ |
| GCP-f | $1.32\times$ | $2.24\times$ | $+0.16$ | $+0.03$ |
| GCP-f | $2.18\times$ | $3.47\times$ | $+0.98$ | $+0.23$ |
| GCP-f | $3.09\times$ | $5\times$ | $+2.18$ | $+0.67$ |
| GCP-f | $5.75\times$ | $10\times$ | $+7.53$ | $+3.87$ |

Table 5: Comparisons on Caltech-256 for ResNet-50. The top-1 and top-5 error % of the pre-trained model are 18.96 and 9.73 respectively.

set, and the rest for the test set. We fine-tune the network for 60 epoch with a learning rate of 0.01 with cosine learning scheduler. Table 5 compares the results obtained from various pruned versions of ResNet-50 with those obtained via the unpruned ResNet-50. For the lightest pruned version of ResNet-50 ($1.55\times$ FLOPs reduction), the performance actually improved. It remains comparable even with $2.24\times$ FLOPs reduction.