Structured DropConnect for Convolutional Neural Networks

ECE1512

Sajad Norouzi

Supervisor: Konstantinos N Plataniotis

2 April 2019

University of Toronto

Contents

1 Introduction

Deep neural networks (DNNs) with a huge number of parameters trained with a massive amount of regularization show pretty good results on different tasks. However, due to a huge number of parameters over-fitting is still an important issue. Dropout [9] is one of the well-known stochastic regularization techniques let giant DNNs converge to a stable point without overfitting. However experimental results show that the improvements due to adding dropout to convolutional neural networks (CNNs) are not considerable. The main hypothesis for this phenomena is the spatial correlation in the activation maps of CNNs. Hence if we drop a neuron from CNNs activation maps, we are not necessarily blocking a part of the information to flow into the subsequent layers of the network. Based on this assumption, some variants of dropout has been proposed that we will review in next sections.

One of the early extensions of the original dropout is DropConnect [12]. Except for one recent work [8], we are not aware of any work on applying DropConnect to CNNs. By applying DropConnect to CNNs, we are basically removing the connection between a particular part of a kernel and the inputs, so definitely we are changing the flow of information. Besides, instead of dropping different connections in convolutional kernels randomly we propose to divide connections into different meaningful groups and drop a group of connections instead of dropping a single connection.

2 Background

2.1 Dropout

One of the observed problems with trained neural networks is co-adaption. Co-adaption is happening whenever some neurons depend strongly on the activation of other neurons. In other words, they have not learned any independent feature. By changing the information flow randomly during the training, dropout can avoid the co-adaption problem. Moreover, the final network is more robust against perturbations and noise.

In [12], it's been proposed to randomly drop the neurons of a fully connected layer during training based on a Bernoulli distribution with mean equal to *keep_prob* which is a hyper-parameter we need to tune. At the inference time, we compute the expected value of each neuron activation. Because the distribution we used for training is Bernoulli, then the expected output is the original activation multiplied by keep_prob. The mathematical equation for a dropout layer is:

$$A = a(Wx) \times M$$

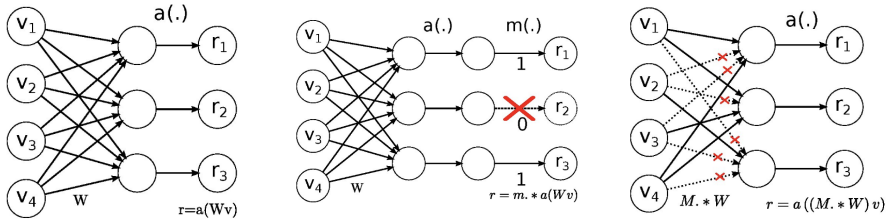
One intuition behind dropout is that at the test time network behaves like ensemble of networks with different configurations.

2.2 Drop-Connect

With the same motivation of Dropout[12], DropConnect has been proposed to add more noise to the network. The primary difference is that instead of randomly dropping the output of the neurons we randomly drop the connection between neurons. The mathematical equation for this method is:

$$A = a(Wx \times M)$$

The training procedure for DropConnect is exactly the same as Dropout. However, we know that $E[a(Wx \times M)]$ is not necessarily equal to $a(E[Wx \times M])$. Therefore, we cannot use the same procedure we used for the inference of Dropout. In the original paper, they proposed to use moment matching between the Bernoulli distribution and a Gaussian distribution. At the inference time, they take samples from the Gaussian distribution and feed them to the activation function. Finally, they average the values to obtain expected value. In figure 1 you can find the visualization of Dropout and DropConnect on fully-connected networks.



(a) (b) (c)

Figure 1: Comparison of Dropout and DropConnect on a normal fully connected network. (a) Normal fully connected (b) Dropout (c) DropConnect

One important property of these methods which requires more attention is that for each training case in a mini-batch, they used a different mask to drop the neurons or connections. This way most of the weights will be updated after each mini-batch update.

3 Related Works

There have been some attempts to change the original dropout to match with the characteristics of convolutional neural networks. The main difference between fully-connected and convolutional neural networks is the fact that there is a spatial correlation in CNNs activation maps and these methods tried to improve Dropout based on this property.

3.1 Spatial Dropout

In [11], they have proposed to drop each channel of activation maps randomly based on a Bernoulli distribution. Dropping a particular channel of an activation map corresponds to turning off a particular filter in a convolutional layer. Spatial dropout definitely changes the flow of the information but the amount of noise they add to the network is limited. Experimental results show that the gain out of adding spatial dropout for the task of classification which is the focus of this paper is marginal. We can say that at the inference time the network behaves similar to an ensemble of different CNNs with various number of filters.

3.2 Cutout

In [1], while they have been trying to make CNNs robust against occlusion for the task of classification, they have found an interesting augmentation approach. They have put a block in different positions of an input image and feed that to the network. Hence a particular spatial part of the network becomes disable. Basically, with this method, we have changed the information we feed to the network and consequently, each training case uses a different part of the network to make a prediction. In the original paper, it has been shown that this method improves the final accuracy of the CNNs on Cifar10 and Cifar100.

3.3 DropBlock

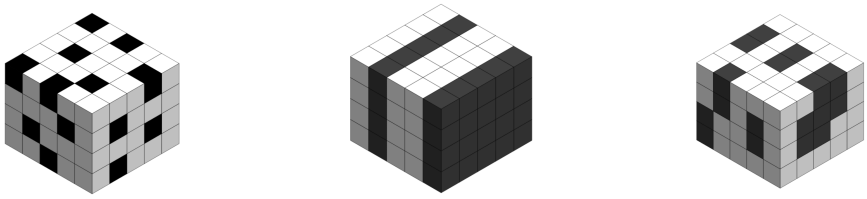
Following the intuition behind Cutout, in [2] it's been proposed to drop a continuous region of activation maps. They have also shown that applying Cutout on the task of Imagenet classification does not give that much of improvement. based on their benchmark on the Imagenet, Dropblock outperforms all other variants of dropout proposed previously. It's worth to mention that the masks applied to the activation maps are different for various channels.

From dropping filters point of view, we can say that DropBlock is similar to disabling convolutional kernels for a defined period of time during the operation of convolving them with the input. DropBlock also introduced a new hyper-parameter that is the block-size which is normally set between 3-7. DropBlock with block size equal to 1 is exactly the same as original Dropout.

In figure 2 you can find geometric visualization of the activation maps after applying Dropout, SpatialDropout, and DropBlock. Cutout visualization is exactly the same as DropBlock but it is only applied to the first layer.

3.4 DropFilter

In a Parallel work, [8] proposed to apply DropConnect to the CNNs kernels. [8] is the first work on applying DropConnect to CNNs. They have also proposed DropFilterPlus which is a DropConnect applied to filters of CNNs with different masks during different steps of convolution operation. Unfortunately, there is no experimental result for their proposed method on DNNs. The only available



(a) (b) (c)

Figure 2: visualization of CNNs activation maps after applying (a) Dropout (b) SpatialDropout (c) DropBlock

result of the method is applying DropFilterPlus on a toy network on MNIST dataset. There is also no available implementation for this technique, so we could not reproduce their results. Generally, this works look like incomplete research.

3.5 Stochastic Max Pooling

One of the first stochastic regularization methods proposed to address the problem of combining Dropout and CNNs is Stochastic Max-Pooling [13]. The intuition behind this method is that by using stochastic Max-pooling we can imitate the behavior of elastic deformation that has been used as a data augmentation technique. Moreover, by using stochastic max-pooling the error signal goes through all parts of the network, so we can use the power of all the network parameters. While this regularization technique is more effective than an original dropout, but in new CNN architecture, there is no Max-pooling layer in between convolutional layers [5, 4]. Most of the recently proposed networks used a bigger stride to make the activation maps smaller. Hence we are not able to use Stochastic MaxPooling for newly proposed networks.

3.6 Scheduled Dropout

All the aforementioned dropout variants have a hyperparameter for the probability of keeping a particular unit in the activation maps (keep_prob). [7] proposed to use adaptive probability during the training. The intuition behind this proposal is the fact that at the beginning of the training when the network weights are initialized randomly, there is no annoying co-adaption between the neurons of the network. As we process in the training and weights took more meaningful values, we need to use more dropout to avoid co-adaption. [7] propose to use exponential decay to set the probability of keeping a particular unit. Clearly, the starting point of keep_prob should be 1 and we can set the threshold to stop

the decay after that. The threshold value is mostly between 0.5 and 0.9. In [2] they have also used this scheduling and scheduled their DropBlock with a linear decay.

4 Structured Drop-Connect

In this work, we propose to rather than dropping activation maps units randomly either in continuous manner like DropBlock or in discrete format like Dropout, drop the filters connections to the input. The first point of this work is that it definitely change the flow of information since we decide to discard some of the units from the images or activation maps of the previous layer. However, the number of connections in CNNs is much more than fully-connected networks, so dropping connections randomly can make the training of the networks overwhelming. Using the intuition that one of the decision that neural network engineers should decide on is kernel size in each layer and knowing the fact that combination of different kernel size can be beneficial [10], we proposed to divide the connection into groups corresponding to different kernel sizes. In next two sections we explain in detail training and inference sections of the algorithm.

4.1 Training

During the training, we randomly select a kernel size which is less than or equal to the original kernel size that has been set by the architecture designer. This kernel size can be different for various kernels in a particular layer. If we wanted to use the same kernel size over different kernels then at the bottleneck layers with 1x1 convolution the flow of the information would be blocked. After selecting a kernel size for each kernel of a layer, we convolve kernels with the inputs and the rest is the same as original convolutional layers. In cases that kernel size equals to zero, structured DropConnect behaves exactly same as SpatialDropout. We have used the following probability function:

$$P_k(i, rate) = \begin{cases} (1 - rate) & i = n \\ rate/(k-1) & i \neq n \end{cases}$$

Here k denotes the original kernel size and $rate$ reference to a dropout hyper-parameter defining the rate of dropping each neuron. It's possible to use different probability for different kernel sizes less than original kernel size but to simplify the model we used the same probability. In figure 3 you can find a visualization of possible kernels that might be selected by the algorithm. Please note that in figure 3 the 3D cubes show the convolutional kernel while in figure 2 the 3D cubes show the activation map.

4.2 Inference

At the inference time, we need to compute the average of activation maps. We believe that the network can imitate the behavior of an ensemble of different

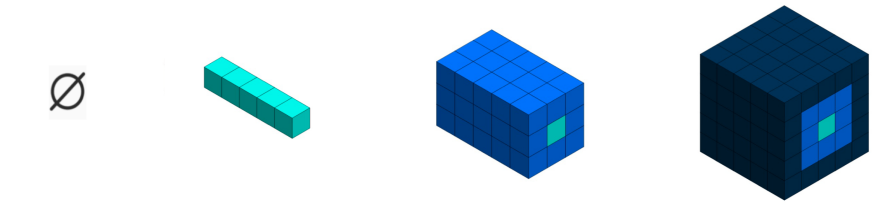


Figure 3: Different convolutional kernels that might be selected by the Structured DropConnect during the training

kernel sizes. Basically, using multi-scale inputs and taking the average over them is a common practice to increase the classification accuracy of the networks. We believe that this structured form of DropConnect can provide the same effect. To compute the average of different activation maps we need to use the following formula:

$$A_l^j = \sum_i^K p_k(i) \times a(W_{l,j}^i * A_{l-1}) \tag{1}$$

Where A_l^j shows the jth activation map of layer l and $W_{l,j}$ defines the weights of jth kernel of layer l. a reference to the activation function which is usually Relu and $p_k(i)$ shows the probability of selecting kernel with size i in a kernel with original size k. To simplify the equation we dropped the *rate* from the inputs of a function p While this is not a valid assumption, but to make the computations easier we assume that the following equation hold with equality:

$$\sum_i^K p_k(i) \times a(W_{l,j}^i * A_{l-1}) \approx a(\sum_i^K p_k(i) \times (W_{l,j}^i * A_{l-1})) \tag{2}$$

Moreover, because of the linear properties of convolution operation, we can further simplify the right-hand side of the above equation to:

$$A_l^j = a(\sum_i^K p_k(i) \times (W_{l,j}^i * A_{l-1})) = a((\sum_i^K p_k(i) \times W_{l,j}^i) * A_{l-1}) \tag{3}$$

Therefore, at the inference time, we only need to compute the weighted sum of different convolutional kernels that we used during training.

Due to the approximation that we have used for the inference, the computed activation map is not necessarily the exact result that we want, but the experimental result shows we can rely on this simplified model.

5 Experiments

To validate the effectiveness of structured DropConnect we took Resnet32[3] implementation from tensorflow repository [1] and trained the network on Cifar10 for 250 epochs. The optimization algorithm we have used was Adam [6] with learning rate multiplied by a factor of 0.1 at epochs: 100, 150, 200. The official implementation of DropBlock was not available so we have taken the implementation from one of available respositories[2]. We trained all the networks with Nvidia GTX 1080 Ti. The implementation of Structured DropBlock is available at: https://github.com/sajadn/Structured-DropConnect

We have divided the convolutional layers into three sections and applied the regularization on the last two sections. Since Resnet only has 3x3 kernels, therefore if we show our method gives improvement we can conclude that the same technique also helps networks with higher kernel sizes.

In [2], block_size=7 has been used to achieve the best results, but they only trained the network on Imagenet dataset with bigger image size (224 instead of 32). So we have trained Resnet with different block size to generate the best results of DropBlock. We also used the linear decay of keep_prob similar to [2]. In table 1 the results have been reported.

Model	Accuracy
Resnet32 + DropBlock (bs = 3)	0.9293
Resnet32 + DropBlock (bs = 5)	0.9308
Resnet32 + DropBlock (bs = 7)	**0.9319**

Table 1: Accuracy of Resnet32 with DropBlock using different block size (bc)

Therefore we conclude that the hyperparameters proposed for Imagenet classification in the original dropblock paper, is also the best proposals for Cifar10 classification.

To find the best scheduling for the probability of keeping a unit in activation maps we have explored two different scheduling. One of them is linear scheduling and another one is piece-wise scheduling. In piece-wise scheduling, we drop the keep_prob by two percents after epochs: 40, 70, 100, 150, 200. In table 2 you can see that the linear scheduling proposed by [2] works better for our structured DropConnect so we use that as the best results of Structured DropConnect.

We have also experimented with other proposed Dropout variations. You can find the final results in table 3. The results are an average of 4 different networks trained from random initialization.

You can see that the results of Structured DropConnect are the best. However, the difference between DropBlock and Structured DropConnect is not that considerable and it might be due to the noise in the results. In order to validate the effect of this technique, we need to do experiments over bigger datasets like

[1] https://github.com/tensorflow/models/tree/master/official/resnet
[2] https://github.com/DHZS/tf-dropblock

Model	Accuracy
Resnet32 + Structured DropConnect (linear scheduling)	0.9317
Resnet32 + Structured DropConnect (piece-wise scheduling)	**0.9321**

Table 2: Accuracy of Resnet32 regularized with Structured DropConnect using different scheduling techniques

Model	Accuracy
Resnet32	0.9259
Resnet32 + Dropout	0.9283
Resnet32 + Spatial Dropout	0.9302
Resnet32 + DropBlock (bs = 7)	0.9319
Resnet32 + Structured DropConnect	**0.9321**

Table 3: Accuracy of Resnet32 with different dropout techniques

Imagenet. This experiment validates that the idea of Structured DropConnect make sense and it is at least as good as DropBlock.

We have also explored the idea of combining DropBlock and Structured DropConnect. In table 4 you can find the results with different block sizes. Please note that since we have used two regularization technique simultaneously, the drop probability decrease to half.

Model	Accuracy
Resnet32 + Structured DropConnect + DropBlock (bs = 3)	0.9338
Resnet32 + Structured DropConnect + DropBlock (bs = 5)	0.9329
Resnet32 + Structured DropConnect + DropBlock (bs = 7)	**0.9347**

Table 3: Accuracy of Resnet32 using both DropBlock and Structured DropBlock

6 Conclusion

In this work, we proposed the structured DropConnect which is randomly selecting different kernel sizes during the training. Using experiments on Cifar10 dataset we have shown that the proposed method has a power to regularize CNNs. We believe this gain is due to the fact that network at the test time behaves like an ensemble of different networks with different kernel sizes.

7 Future Work

- We have used the same sized kernels for different channels, but we can add more noise using different kernel size in various channels.
- We used the same mask for all the training case in mini-batch, however, in Dropout and DropConnect this mask for each training sample is different. One future direction might be making the Structured DropConnect for data points in a mini-batch differently.
- At the inference time, we have used the equation 3, but with Relu as an activation function this equation is not exactly equal to the expected value we want. DropConnect [1] used moment matching to solve this problem. We can apply the same algorithm to this work.
- Combination of DropBlock and Structured DropConnect make sense since one of them take advantage of activating different spatial thinned networks and another one leverage the power of different kernel sizes.

References

[1] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[2] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10750–10760, 2018.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Michael G Hluchyj and Mark J Karol. Shuffle net: An application of generalized perfect shuffles to multihop lightwave networks. *Journal of Lightwave Technology*, 9(10):1386–1397, 1991.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, René Vidal, and Vittorio Murino. Curriculum dropout. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3544–3552, 2017.

[8] Hengyue Pan, Hui Jiang, Xin Niu, and Yong Dou. Dropfilter: A novel regularization method for learning convolutional neural networks. *arXiv preprint arXiv:1811.06783*, 2018.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[11] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[13] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.