УДК 004.93

V.V. Dudar, V.V. Semenov
Taras Shevchenko National University of Kyiv, Ukraine
64/13, Volodymyrska str., Kyiv, 01601

COLUMN DROP: MAKING CNNS INVARIANT TO IMAGE CROPPING

В.В. Дудар, В.В. Семенов
Київський національний університет імені Тараса Шевченка, Україна
вул. Володимирська 64/13, м. Київ 01601

COLUMN DROP: КРОК ДО ІНВАРІАНТНОСТІ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ДО ВИБОРУ ПІДЗОБРАЖЕННЯ

We introduce a new regularization technique column drop which uses inner structure of CNNs for classification to make its output invariant to random crops of input image. Use of this regularization eliminates need in data augmentation by random image cropping under some conditions on architecture of CNN. We show that application of column drop to pooling layers leads to improvement in generalization compared with use of dropout for pooling layers.

Keywords: convolutional neural net, invariance to image cropping, regularization, dropout

В статті описано новий метод регуляризації column drop для навчання згорткових нейронних мереж для класифікації, що робить їх інваріантними до вибору підзображення. Використання такої регуляризації відкидає необхідність в розширенні навчальної вибірки зображень за допомогою вибору випадкових підзображень, за певних умов на архітектуру мережі. Застосування column drop до pooling шарів мережі призводить до покращення точності класифікації на тестовій вибірці у порівнянні з використанням методу dropout для pooling шарів.

Ключові слова: згорткова нейронна мережа, інваріантність до вибору підзображення, регуляризація, dropout

Introduction

Convolutional neural networks are one of the most successful models in image recognition [1]. One of the main reasons CNNs show good performance is their inner structure that induces translation invariance of the model. However, overfitting effect is still significant, that's why various regularization techniques are used to improve generalization. The most popular regularization techniques are dropout [2] and data augmentation [1]. A simple way to increase size of the training set is to use random image cropping. Since convolutional neural network accepts only images of the fixed size as input, there are several ways to use this technique. The first one is to upsample cropped images to the initial image size, and feed these to the network during training. This approach could produce some artifacts with upsampling, which could hurt test set performance. The second approach is to use the image with black frame around the cropped image into the network. Again, the trained network will expect black frame to appear at the test time, which could decrease test set accuracy. The third possibility is to train CNN that accepts images of the smaller size, with cropped images as inputs. Then at the test time we need to select several subimages (usually its four corner subimages and one middle subimage) and average network predictions on them (this can be viewed as ensemble of five networks). This approach was used for AlexNet [3] training. It does not suffer from image distortions as previous approaches, but requires running resulting network several times at test time.

At this paper we show that a single CNN under some conditions on architecture can be viewed as ensemble of smaller networks that share parameters and computations and act on different subimages of the input. Using a simple regularization procedure, similar to Dropout, we can enforce each of these subnetworks to produce correct output on each training image, and use average of them at test time. These will make the single network trained with column drop regularization internally invariant to image cropping.

Column drop description

To illustrate the idea, we consider an Alex-net style CNN, which consists of several convolutional layers with nonlinear activation function, pooling layers, and fully connected layer with softmax nonlinearity (see Figure 1) (the same considerations also apply to other architecture types, such as DenseNet [4]).

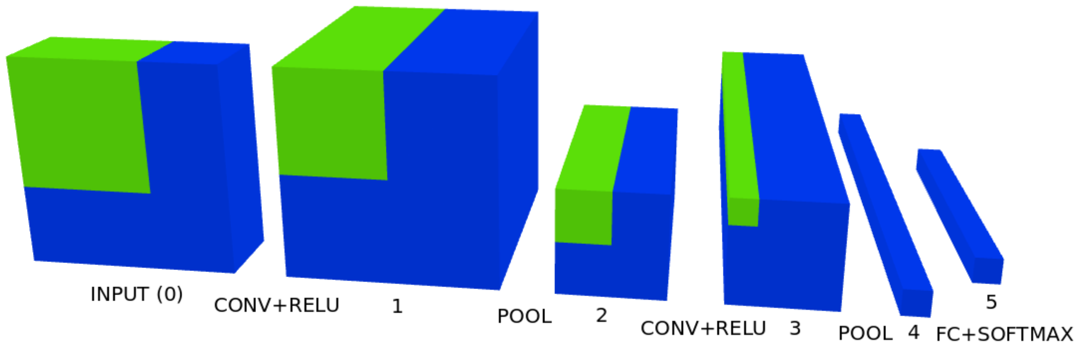

Fig. 1. Sample convolutional net

We assume that depth of tensor 3 is the same as length of vector 4 (so we have full average pooling before fully connected layer). Consider arbitrary single column in tensor 3 (with fixed spatial position, denoted by green on Figure). This column was obtained with convolution operation from previous layer, so it depends on green block in tensor 2, which in turn was obtained by average pooling so it depends on green block in tensor 1, which in turn depends on green block in the input image.

So, if number of convolutional and pooling layers in the network is small enough, then each block $1 \times 1 \times d$ in the tensor to which full average pooling is applied, depends on part of the input image.

At the same time, since we use convolutional and pooling layers, that share weights over different spatial regions of the input and intermediate tensors, so different $1 \times 1 \times d$ blocks compute the same mapping, applied to different subregions of the image. Strictly saying, it can be viewed as application of the mapping to the subimage padded with zeros, as it is shown of Figure 2

We can control size of the frame each tensor block depends on by changing number of convolutional and pooling layers in the network. We can adjust it in a way that each window covers subimage of the needed size. As we will see further, with column drop there is a tradeoff between window size, training accuracy and test accuracy: small window size corresponds to better generalization at the cost of more complicated training.

So if the network contains full average pooling before the fully connected plus softmax layer, then we can view the single CNN as ensemble of smaller networks (having shared architecture and weights) applied to different subimages padded with zeros. Thus our aim is to make each of these subnetworks to predict correct class for each element in the training set, and average their predictions to get class label at test time.

We can achieve that by applying such technique: at training time after the forward pass select random column of the final tensor, and copy it to the vector to feed into fully connected layer. At test time average all the columns into the vector, and apply fully connected layer.

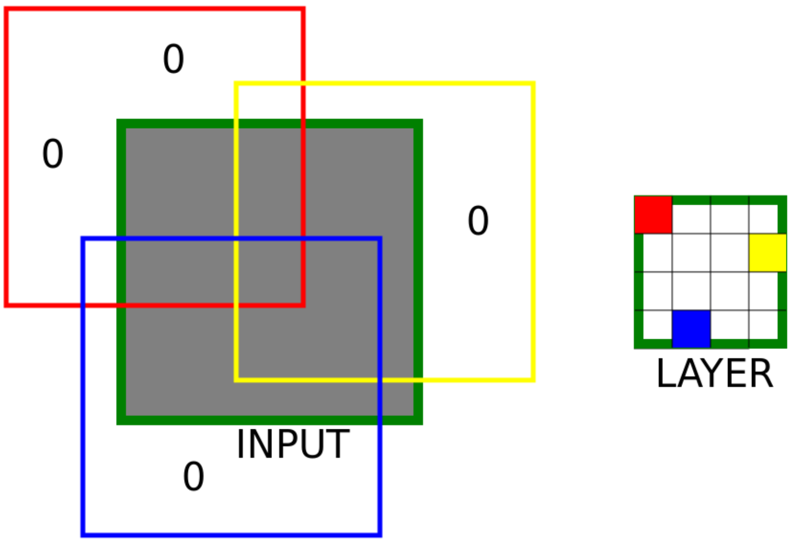

Fig. 2. Multiple receptive regions

We found that it is beneficial to average k randomly chosen columns instead of one to speed up training - it provides faster training while keeping generalization at similar level. But as k becomes bigger, generalization tends to become worse. Random columns to be pooled are separately selected for each element in minibatch during training. Procedure is shown on Figure 3

So main differences from dropout are that we drop columns instead of separate elements, and that we fix in advance number of columns to remain, instead of dropping them independently with some probability. This allows avoiding sampling bias at the stage of making transition to the test setting. There is no need to make additional multiplications at test stage, since expectation of average of k randomly selected columns equals to average of all columns (in fact this multiplication is implicit, since during training each present column is divided by k, and during testing by total number of columns).

Mathematical properties

Let's denote input image by I, assume the last tensor before full average pooling has dimensionality $N \times N$, mapping from a corresponding input window I_{ij} to column (i, j) of that tensor by $f\left(I_{ij}, W_{conv}\right)$ (function f is the same for all tensor columns, and shares the same convolutional parameters W_{conv}), and weights of the fully connected layer by W_{fc}. Then network output at test time is such:

$$y\left(I, W_{conv}, W_{fc}\right) = \text{softmax}\left(\frac{W_{fc}}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} f\left(I_{ij}, W_{conv}\right)\right)$$

Taking W_{fc} inside the sum we will get:

$$y\left(I, W_{conv}, W_{fc}\right) = \text{softmax}\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} W_{fc} f\left(I_{ij}, W_{conv}\right)\right)$$

indicating that output of the network before softmax nonlinearity is equal to average of smaller CNNs.

Cross-entropy error function for a single input image I with target vector t (we avoid summation over entire training set to keep notation uncluttered):

$$F\left(W_{conv}, W_{fc}\right) = -\sum_{c=1}^{C} t_c \ln\left(y_c\left(I, W_{conv}, W_{fc}\right)\right)$$

Cross-entropy error function F is convex [5] with respect to parameter W_{fc} which implies:

$$-\sum_{c=1}^{C} t_c \ln\left(\text{softmax}_c\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} W_{fc}\, f\left(I_{ij}, W_{conv}\right)\right)\right) \le$$
$$-\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{c=1}^{C} t_c \ln\left(\text{softmax}_c\left(W_{fc}\, f\left(I_{ij}, W_{conv}\right)\right)\right)$$

The right side of the inequality is the average of cross-entropy loss functions for each separate column in the last tensor. Thus when we randomly drop all columns except of one to find the gradient, we are implicitly applying stochastic gradient descent to the right term of the inequality (we are randomly choosing one of N^2 summands at each step). Thus we are minimizing function that is a majorant of the cross-entropy error function of the training set. This inequality guarantees that when we apply minimization procedure with column drop, error function for training set at test time will not exceed expectation of the error function with randomly dropped columns at training time.

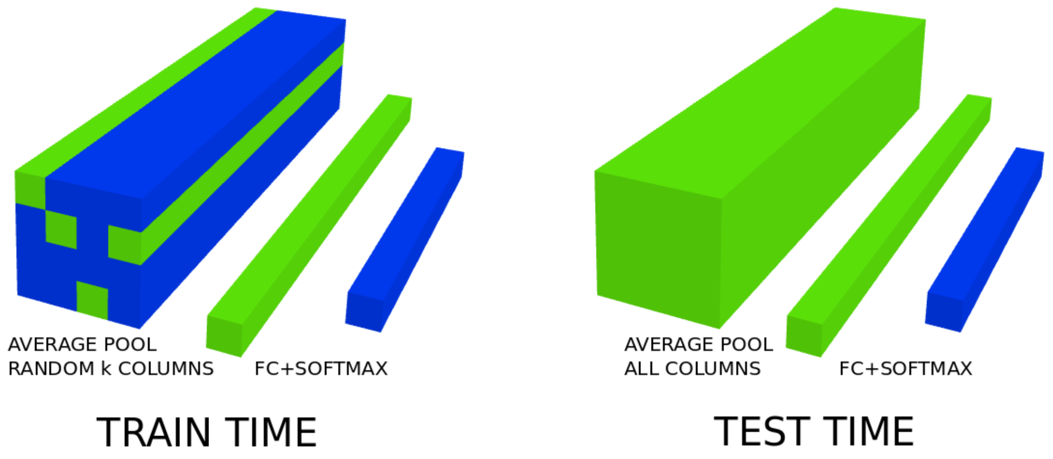

Fig. 3. Train and test time procedures

Let's consider the case we randomly select several columns that will be kept at each step. Easy to see that in case arbitrary function $g(\cdot)$ is convex, then such inequalities hold:

$$g\left(\frac{1}{M}\sum_{i=1}^{M} x_i\right) \le \frac{1}{M}\sum_{i=1}^{M} g\left(\frac{1}{M-1}\sum_{j=1, j\ne i}^{M} x_j\right) \le$$
$$\frac{1}{\left(2^M\right)}\sum_{i_1=1}^{M}\sum_{i_2=1, i_2\ne i_1}^{M} g\left(\frac{1}{M-2}\sum_{j=1, j\ne i_1, j\ne i_2}^{M} x_j\right) \ldots \le \frac{1}{M}\sum_{i=1}^{M} g\left(x_i\right)$$

Since cross-entropy error function $-\sum_{c=1}^{C} t_c \ln\left(\text{softmax}_c\left(\cdot\right)\right)$ is convex, and taking $W_{fc}\, f\left(I_{ij}, W_{conv}\right)$ as inputs, we will obtain that average cross entropy error function for N^2 models with single columns is a majorant for average error function of $\binom{N^2}{2}$ models with 2 selected columns, which in turn is a majorant of average error function of $\binom{N^2}{3}$ cross-entropy functions for 3 selected columns, and so on, that is in turn majorant for the error function of the model where we do not apply column drop.

Thus if we train the model where only 1 column is kept at each step, this guarantees that models where we keep two or more random columns are also implicitly trained. From the other side training with such regularization could be successful only with bigger models, so balance between these factors should be found.

Application to inner pooling layers

We found that it is beneficial to apply column drop regularization also to inner pooling layers of CNN. In this case we lose theoretical properties derived in the previous section: the model we use at test time is not exactly equal to the average of models at training time, thus there are no guarantees that test model will perform better than any of the training models (here we have the same situation as with dropout).

We experimented also with application of column drop after convolutional layers of the neural network: it shows worse generalization compared with dropout applied after same layers.

Thus recommended way to use column drop is to apply it to all pooling layers of the network, and use dropout after convolutional layers.

Experiments

To test proposed regularization we train a variant of DenseNet on Cifar-10 [6] dataset, that consists of 32×32 color natural images. We follow setting of the paper [4] and use bottleneck convolutional layers of such structure: BN-Relu-Conv1×1-BN-Relu-Conv3×3 (here BN stands for batch normalization [8], Relu is Rectified linear unit [9]) and concatenation of result with input tensor. We use 4 dense blocks, separated by average pooling layers with sizes 2, 2, 2 and 4. Numbers of bottleneck layers in each dense block are written as a list in the table 1. We also use initial convolution with kernel 3×3 and depth 2 * growth rate (growth rate and bottleneck depth are specified in the table 1).

We use transition layers for pooling with such sequence of layers: BN-Relu-Conv1×1-Drop-AveragePool with pooling fraction 0.5. Here Drop could refer to dropout or column drop, depending on the setting.

Table 1. Classification results.

Network architecture	Regularization / Augmentation	Train error	Test error	Train accuracy	Test accuracy
Dense-BN, [1,1,1,1] Bottleneck depth: 16 Growth rate: 16 15388 parameters	-	0.21	0.65	93.56%	79.59%
	Random crop	0.35	0.51	87.91%	**82.91%**
	Column drop 0.2	0.33	0.50	88.38%	82.60%
	Drop 0.2	0.34	0.51	88.12%	82.63%
	Column drop 0.5	0.55	0.64	80.06%	77.45%
	Drop 0.5	0.53	0.63	81.37%	78.19%
Dense BN, [1,1,1,1] Bottleneck depth: 32 Growth rate: 32 57406 parameters	-	0.00	0.73	100%	83.11%
	Random crop	0.09	0.47	96.94%	86.74%
	Column drop 0.2	0.05	0.42	98.86%	**87.13%**
	Drop 0.2	0.03	0.48	99.49%	86.27%
	Column drop 0.5	0.22	0.42	92.05%	86.00%
	Drop 0.5	0.17	0.44	94.44%	86.52%
Dense BN, [1,1,1,1] Bottleneck depth: 64 Growth rate: 64 221362 parameters	-	0.00	0.48	100%	86.44%
	Random crop	0.00	0.43	99.97%	89.86%
	Column drop 0.2	0.00	0.38	100%	90.36%
	Drop 0.2	0.00	0.45	100%	88.59%
	Column drop 0.5	0.03	0.33	99.40%	**90.64%**
	Drop 0.5	0.01	0.51	99.96%	87.84%
Dense-BN, [2,2,2,2] Bottleneck depth: 16 Growth rate: 16 30002 parameters	-	0.07	0.80	100%	82.77%
	Random crop	0.18	0.38	93.39%	**87.85%**
	Column drop 0.2	0.08	0.44	97.54%	86.40%
	Drop 0.2	0.07	0.46	98.03%	86.43%
	Column drop 0.5	0.23	0.43	91.58%	85.94%
	Drop 0.5	0.21	0.46	92.91%	85.72%

To make unified comparison with dropout, we apply probabilistic dropping to our procedure also: at training time we drop columns of the tensor with probability p, and multiply values by $1/(1-p)$ and average pool all columns, and test time we just apply average pooling to all columns.

In all cases we train network for 200 epochs with SGD with learning rate 0.1, momentum 0.9, quadratic weight decay with coefficient 0.0005. We decrease learning rate with factor 0.97 after each epoch.

Results of classification for different architectures are summarized in the table 1

Analysis of experiments

The first 3 network architectures used have property that each column of the last tensor before full average pooling depends on the subset of input. In particular, corner columns of the tensor depend on 23*23 corner squares of the input (that has spatial dimensionality 32*32). For these architectures column drop regularization shows similar or better results than random crop augmentation and dropout for pooling layers.

For the fourth architecture, that has 2 bottleneck layers in each dense block, each column of the final tensor depends on the whole input image, that’s why random crop augmentation shows better results than column drop and dropout. But these regularizations are still improving generalization compared with the case no augmentation with random cropping is used.

So, as expected, column drop improves results in case number of pooling and convolutional layers of the network is small enough to guarantee that columns of the last tensor depend on subinput, which is not the case for state-of-art architectures.

If CNN is deep enough, this method does not alleviate need in data augmentation with random image cropping. In this case column drop still can be used for pooling layers in combination with data augmentation of input by cropping and dropout after convolutional layers to produce even better results.

Conclusion

Proposed regularization method column drop can be used to improve generalization of convolutional neural networks. It has nice theoretical interpretation in terms of making output of CNN invariant to random image cropping. When applied to pooling layers it shows superior performance compared with dropout – widely used regularization method.

Drawback of this method is that it holds its theoretical properties of making output invariant to image crops only under certain conditions on CNN architecture: if receptive field of the columns of the last tensor does not cover entire input image.

Acknowledgments

We gratefully acknowledge the support of NVIDIA with the donation of the Titan X Pascal GPU used for this research.

References

1. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press, http: //www.deeplearningbook.org
2. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15:1929–1958.
3. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.
4. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
5. Bishop C.M. Pattern Recognition and Machine Learning. – Springer Science + Business Media 2006. – 703 p.
6. Krizhevsky A., Learning Multiple Layers of features from tiny images, 2009.
7. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015
8. X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In AISTATS, 2011.

РЕЗЮМЕ

В.В. Дудар, В.В. Семенов

Column Drop: крок до інваріантності згорткових нейронних мереж до вибору підзображення

У статті описано новий метод регуляризації згорткових нейронних мереж Column Drop. Даний метод базується на загальновідомому методі регуляризації Dropout, який покращує точність класифікації нейронних мереж за рахунок представлення нейронної мережі як ансамблю мереж (що містять підмножини нейронів початкової мережі) і тренування випадково вибраних підмереж ансамблю. В даній роботі показано, що згорткова мережа може розглядатись як ансамбль підмереж, що мають спільні параметри, але діють на різні частини зображення. Тренуючи випадково вибрані підмережі, ми досягаємо того, що кожна з підмереж дає правильну класифікацію вхідного зображення. В тестовому режимі відбувається усереднення підмереж, за рахунок чого і досягається покращення точності на тестовій вибірці.

Механізм застосування Column Drop такий: під час тренування випадковим чином з певною ймовірністю видаляються стовпчики тензорів нейронної мережі (кожен стовпчик має фіксовану просторову позицію), стовпчики що залишились, домножуються на коефіцієнт таким чином, щоб математичне очікування кожного елемента тензора було сталим. У тестовому режимі стовпчики тензорів не видаляються.

Запропонований алгоритм робить мережу інваріантною до вибору випадкового підзображення за умови що стовпці останнього тензору мережі (перед повним пулінгом) залежать лише від підзображень вхідного зображення, що виконується для неглибоких мереж.

Для тестування даного підходу були проведені порівняння точностей на тестовій вибірці CIFAR-10 для декількох конфігурацій згорткових мереж, які були натреновані без регуляризації, з регуляризацією dropout, запропонованим методом column drop, та без регуляризації але з розширенням навчальної вибірки за допомогою вибору випадкових підзображень. Результати тестування показали, що Column Drop показує кращі результати ніж Dropout та розширення навчальної вибірки за допомогою вибору підзображень, за умови що стовпці останнього тензора мере-жі залежать від частини вхідного зображення.

Якщо ця умова не виконується, тоді Column Drop може бути скомбінований з розширенням навчальної вибірки чи іншими методами регуляризації для досягнення кращих результатів.

Надійшла до редакції 20.09.2018