Fractional Max-Pooling

Benjamin Graham
Dept of Statistics, University of Warwick, CV4 7AL, UK
b.graham@warwick.ac.uk

May 13, 2015

Abstract

Convolutional networks almost always incorporate some form of spatial pooling, and very often it is $\alpha \times \alpha$ max-pooling with $\alpha = 2$. Max-pooling act on the hidden layers of the network, reducing their size by an integer multiplicative factor α. The amazing by-product of discarding 75% of your data is that you build into the network a degree of invariance with respect to translations and elastic distortions. However, if you simply alternate convolutional layers with max-pooling layers, performance is limited due to the rapid reduction in spatial size, and the disjoint nature of the pooling regions. We have formulated a *fractional* version of max-pooling where α is allowed to take non-integer values. Our version of max-pooling is *stochastic* as there are lots of different ways of constructing suitable pooling regions. We find that our form of fractional max-pooling reduces overfitting on a variety of datasets: for instance, we improve on the state of the art for CIFAR-100 without even using dropout.

1 Convolutional neural networks

Convolutional networks are used to solve image recognition problems. They can be built by combining two types of layers:

- Layers of convolutional filters.
- Some form of spatial pooling, such as max-pooling.

Research focused on improving the convolutional layers has lead to a wealth of techniques such as dropout [10], DropConnect [12], deep networks with many small filters[2], large input layer filters for detecting texture [5], and deeply supervised networks [6].

By comparison, the humble pooling operation has been slightly neglected. For a long time 2×2 max-pooling (MP2 has been the default choice for building convolutional networks. There are many reasons for the popularity of MP2-pooling: it is fast, it quickly reduces the size of the hidden layers, and it encodes

a degree of invariance with respect to translations and elastic distortions. However, the disjoint nature of the pooling regions can limit generalization. Additionally, as MP2-pooling reduces the size of the hidden layers so quickly, stacks of back-to-back convolutional layers are needed to build really deep networks [7, 9, 11]. Two methods that have been proposed to overcome this problems are:

- Using 3×3 pooling regions overlapping with stride 2 [5].
- Stochastic pooling, where the act of picking the maximum value in each pooling region is replaced by a form of size-biased sampling [13].

However, both these techniques still reduce the size of the hidden layers by a factor of two. It seems natural to ask if spatial-pooling can usefully be applied in a gentler manner. If pooling was to only reduce the size of the hidden layers by a factor of $\sqrt{2}$, then we could use twice as many layers of pooling. Each layer of pooling is an opportunity to view the input image at a different scale. Viewing images at the 'right' scale should make it easier to recognize the tell-tale features that identify an object as belonging to a particular class.

The focus of this paper is thus a particular form of max-pooling that we call *fractional max-pooling* (FMP). The idea of FMP is to reduce the spatial size of the image by a factor of α with $1 < \alpha < 2$. Like stochastic pooling, FMP introduces a degree of randomness to the pooling process. However, unlike stochastic-pooling, the randomness is related to the choice of pooling regions, not the way pooling is performed inside each of the pooling regions.

In Section 2 we give a formal description of fractional max-pooling. Briefly, there are three choices that affect the way FMP is implemented:

- The pooling fraction α which determines the ratio between the spatial sizes of the input and the output of the pooling layer. Regular 2×2 max-pooling corresponds to the special case $\alpha = 2$.
- The pooling regions can either be chosen in a *random* or a *pseudorandom* fashion. There seems to be a trade off between the use of randomness in FMP and the use of dropout and/or training data augmentation. Random-FMP seems to work better on its own; however, when combined with 'too much' dropout or training data augmentation, underfitting can occur.
- The pooling regions can be either *disjoint* or *overlapping*. Disjoint regions are easier to picture, but we find that overlapping regions work better.

In Section 3 we describe how our convolutional networks were designed and trained. In Section 4 we give results for the MNIST digits, the CIFAR-10 and CIFAR-100 datasets of small pictures, handwritten Assamese characters and the CASIA-OLHWDB1.1 dataset of handwritten Chinese characters.

2 Fractional max-pooling

Each convolutional filter of a CNN produces a matrix of hidden variables. The size of this matrix is often reduced using some form of pooling. Max-pooling is

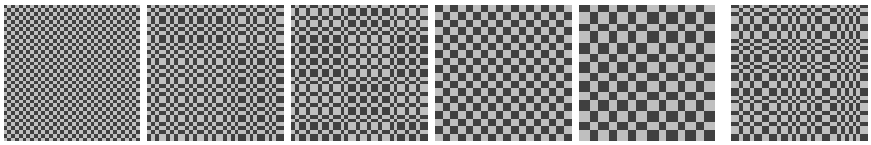

Figure 1: Left to right: A 36×36 square grid; disjoint pseudorandom FMP regions with $\alpha \in \{\sqrt[3]{2}, \sqrt{2}, 2, \sqrt{5}\}$; and disjoint random FMP regions for $\alpha = \sqrt{2}$. For $\alpha \in (1, 2)$ the rectangles have sides of length 1 or 2. For $\alpha \in (2, 3)$ the rectangles have sides of length 2 or 3. [Please zoom in if the images appear blurred.]

a procedure that takes an $N_{\text{in}} \times N_{\text{in}}$ input matrix and returns a smaller output matrix, say $N_{\text{out}} \times N_{\text{out}}$. This is achieved by dividing the $N_{\text{in}} \times N_{\text{in}}$ square into N_{out}^2 pooling regions $(P_{i,j})$:

$$P_{i,j} \subset \{1, 2, \ldots, N_{\text{in}}\}^2 \quad \text{for each} \quad (i, j) \in \{1, \ldots, N_{\text{out}}\}^2,$$

and then setting

$$\text{Output}_{i,j} = \max_{(k,l) \in P_{i,j}} \text{Input}_{k,l}.$$

For regular 2×2 max-pooling, $N_{\text{in}} = 2N_{\text{out}}$ and $P_{i,j} = \{2i-1, 2i\} \times \{2j-1, 2j\}$. In [5], max-pooling is applied with overlapping 3×3 pooling regions so $N_{\text{in}} = 2N_{\text{out}} + 1$ and the $P_{i,j}$ are 3×3 squares, tiled with stride 2. In both cases, $N_{\text{in}}/N_{\text{out}} \approx 2$ so the spatial size of any interesting features in the input image halve in size with each pooling layer. In contrast, if we take $N_{\text{in}}/N_{\text{out}} \approx \sqrt[n]{2}$ then the rate of decay of the spatial size of interesting features is n times slower. For clarity we will now focus on the case $N_{\text{in}}/N_{\text{out}} \in (1, 2)$ as we are primarily interested in accuracy; if speed is an overbearing concern then FMP could be applied with $N_{\text{in}}/N_{\text{out}} \in (2, 3)$.

Given a particular pair of values $(N_{\text{in}}, N_{\text{out}})$ we need a way to choose pooling regions $(P_{i,j})$. We will consider two type of arrangements, overlapping squares and disjoint collections of rectangles. In Figure 1 we show a number of different ways of dividing up a 36×36 square grid into disjoint rectangles. Pictures two, three and six in Figure 1 can also be used to define an arrangement of overlapping 2×2 squares: take the top left hand corner of each rectangle in the picture to be the top left hand corner of one of the squares.

To give a formal description of how to generate pooling regions, let $(a_i)_{i=0}^{N_{\text{out}}}$ and $(b_i)_{i=0}^{N_{\text{out}}}$ be two increasing sequence of integers starting at 1, ending with N_{in}, and with increments all equal to one or two (i.e. $a_{i+1} - a_i \in \{1, 2\}$). The regions can then be defined by either

$$P = [a_{i-1}, a_i - 1] \times [b_{j-1}, b_j - 1] \quad \text{or} \quad P_{i,j} = [a_{i-1}, a_i] \times [b_{j-1}, b_j]. \tag{1}$$

We call the two cases *disjoint* and *overlapping*, respectively. We have tried two different approaches for generating the integer sequence: using *random* sequences of numbers and also using *pseudorandom* sequences.

Figure 2: Top left, 'Kodak True Color' parrots at a resolution of 384×256. The other five images are one-eighth of the resolution as a result of 6 layers of average pooling using disjoint random FMP$\sqrt{2}$-pooling regions.

We will say that the sequences are *random* if the increments are obtained by taking a random permutation of an appropriate number of ones and twos. We will say that the sequences are *pseudorandom* if they take the form

$$a_i = \text{ceiling}(\alpha(i+u)), \qquad \alpha \in (1,2), \text{ with some } u \in (0,1).$$

Below are some patterns of increments corresponding to the case $N_{\text{in}} = 25$, $N_{\text{out}} = 18$. The increments on the left were generated 'randomly', and the increments on the right come from pseudorandom sequences:

211112112211112122	112112121121211212
111222121121112121	212112121121121211
121122112111211212	211211212112121121

Although both types of sequences are irregular, the pseudorandom sequences generate much more stable pooling regions than the random ones. To show the effect of randomizing the pooling regions, see Figure 2. We have taken a picture, and we have iteratively used disjoint random pooling regions to reduce the size of the image (taking averages in each pooling region). The result is that the scaled down images show elastic distortion. In contrast, if we use pseudorandom pooling regions, the resulting image is simply a faithfully scaled down version of the original.

3 Implementation

The networks are trainined using an implementation of a sparse convolutional network [3]. What this means in practice is that we can specify a convolutional

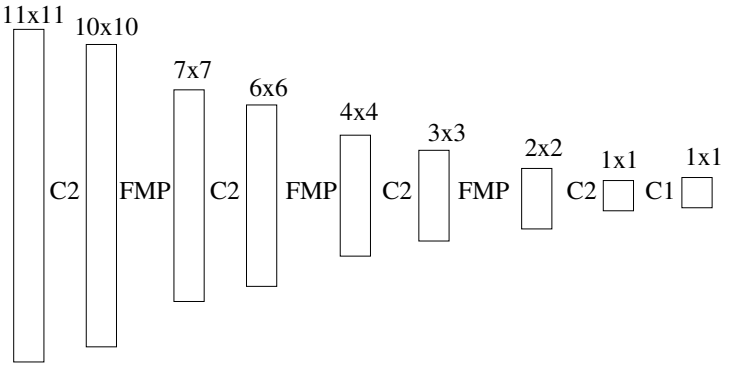

Figure 3: Layer sizes for a tiny FMP$\sqrt{2}$ network. The fractions $\frac{3}{2}$, $\frac{6}{4}$ and $\frac{10}{7}$ approximate $\sqrt{2}$.

network in terms of a sequence of layers, e.g.

$$10C2 - FMP\sqrt{2} - 20C2 - FMP\sqrt{2} - 30C2 - FMP\sqrt{2} - 40C2 - 50C1 - \text{output}.$$

The spatial size of the input layer is obtained by working from right to left: each C2 convolution increases the spatial size by one, and FMP$\sqrt{2}$ layers increase the spatial size by a factor of $\sqrt{2}$, rounded to the nearest integer; see Figure 3. The input layer will typically be larger than the input images—padding with zeros is automatically added as needed. Fractional max-pooling could also easily be implemented for regular convolutional neural network software packages.

For simplicity, all the networks we use have a linearly increasing number of filters per convolutional layer. We can therefore describe the above network using the shorthand form

$$(10nC2 - FMP\sqrt{2})_3 - C2 - C1 - \text{output},$$

$10n$ indicates that the number of filters in the n-th convolutional layer is $10n$, and the subscript 3 indicates three pairs of alternating C2/FMP layers. When we use dropout, we use an increasing amount of dropout the deeper we go into the network; we apply 0% dropout in the first hidden layer, and increase linearly to 50% dropout in the final hidden layer. We use leaky rectified linear units.

3.1 Model averaging

Each time we apply an FMP network, either for training or testing purposes, we use different random or pseudorandom sequences to generate the pooling regions. An FMP network can therefore be thought of as an ensemble of similar networks, with each different pooling-region configuration defining a different member of the ensemble. This is similar to dropout [10]; the different values the dropout mask can take define an ensemble of related networks. As with

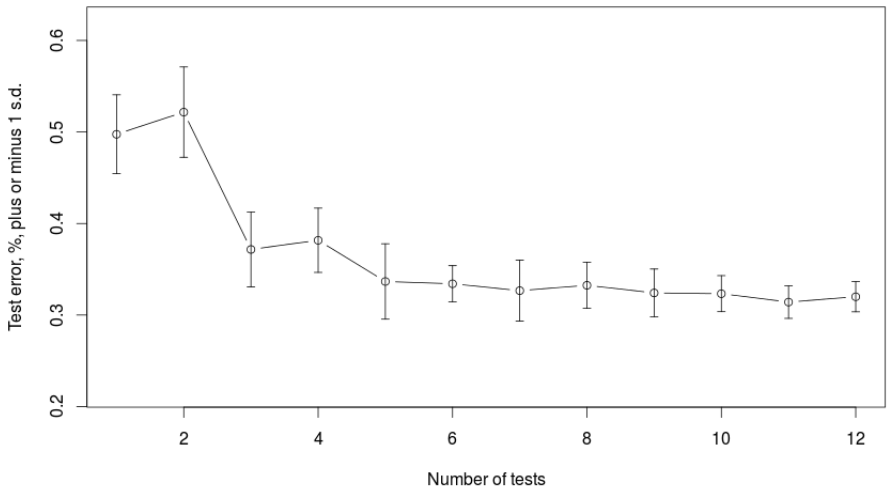

Figure 4: The effect of repeat testing for a single MNIST trained FMP network.

dropout, model averaging for FMP networks can help improve performance. If you classify the same test image a number of times, you may get a number of different predictions. Using majority voting after classifying each test image a number of times can substantially improve accuracy; see Figure 4.

4 Results

4.1 Without training set augmentation or dropout

To compare the different kinds of fractional max-pooling, we trained FMP networks on the MNIST[1] set of digits and the CIFAR-100 dataset of small pictures [4]. For MNIST we used a small FMP network:

$$\text{input layer size } 36 \times 36 : \qquad (32nC2 - FMP\sqrt{2})_6 - C2 - C1 - \text{output},$$

and for CIFAR-100 we used a larger network:

$$\text{input layer size } 94 \times 94 : \qquad (64nC2 - FMP\sqrt[3]{2})_{12} - C2 - C1 - \text{output}.$$

Without using training data augmentation, state-of-the-art test errors for these two datasets are 0.39% and 34.57%, respectively [6]. Results for the FMP networks are in Table 1. Using model averaging with multiplicity twelve, we find that random overlapping FMP does best for both datasets. For CIFAR-100, the improvement over method using regular max-pooling is quite substantial.

[1] `http://yann.lecun.com/exdb/mnist/`

Dataset and the number of repeat tests	pseudorandom disjoint	random disjoint	pseudorandom overlapping	random overlapping
MNIST, 1 test	0.54	0.57	**0.44**	0.50
MNIST, 12 tests	0.38	0.37	0.34	**0.32**
CIFAR-100, 1 test	31.67	32.06	**31.2**	31.45
CIFAR-100, 12 tests	28.48	27.89	28.16	**26.39**

Table 1: MNIST and CIFAR-100 % test errors.

To give an idea about network complexity, the CIFAR-100 networks have 12 million weights, and were trained for 250 repetitions of the training data (18 hours on a GeForce GTX 780). We experimented with changing the number of hidden units per layer for CIFAR-100 with random overlapping pooling:

- Using '$16nC2$' (0.8M weights) gave test errors of 42.07% / 34.87%.
- Using '$32nC2$' (3.2M weights) gave test errors of 35.09% / 29.66%.
- Using '$96nC2$' (27M weights) combined with dropout and a slower rate of learning rate decay gave test errors of 27.62% / 23.82%.

4.2 Assamese handwriting

To compare the effect of training data augmentation when using FMP pooling versus MP2 pooling, we used the The Online Handwritten Assamese Characters Dataset [1]. It contains 45 samples for each of 183 Indo-Aryan characters. 'Online' means that each pen stroke is represented as a sequence of (x, y) coordinates. We used the first 36 handwriting samples as the training set, and the remaining 9 samples for a test set. The characters were scaled to fit in a box of size 64×64. We trained a network with six layers of 2×2 max pooling,

$$32nC3 - MP2 - (C2 - MP2)_5 - C2 - \text{output}$$

and an FMP network using 10 layers of random overlapping FMP$\sqrt{2}$ pooling,

$$(32nC2 - FMP\sqrt{2})_{10} - C2 - C1 - \text{output}.$$

We trained the networks without dropout, and either

- no training data augmentation,
- with the characters shifted by adding random translations, or
- with affine transformations, using a randomized mix of translations, rotations, stretching, and shearing operations.

Pooling method	None	Translations	Affine
6 layers of MP2	14.1	4.6	1.8
10 layers of FMP($\sqrt{2}$), 1 test	1.9	1.3	0.9
10 layers of FMP($\sqrt{2}$), 12 tests	**0.7**	**0.8**	**0.4**

Table 2: Assamese % test error with different type of data augmentation.

See Table 2. In a sense, max-pooling and training data augmentation are two different ways of encoding our apriori knowledge that the meaning of handwriting is generally invariant under certain kinds of minor distortions. Interestingly, the FMP network without data augmentation does better than the MP2 network with training data augmentation, suggesting that FMP is a better way of encoding that information.

4.3 Online Chinese handwriting

The CASIA-OLHWDB1.1 database contains online handwriting samples of the 3755 isolated GBK level-1 Chinese characters [8]. There are approximately 240 training characters, and 60 test characters, per class. A test error of 5.61% is achieved using 4 levels of MP2 pooling [2].

We used the representation for online characters described in [3]; the characters were drawn with size 64×64 and additional features measuring the direction of the pen are added to produce an array of size $64 \times 64 \times 9$. Using 6 layers of 2×2 max-pooling, dropout and affine training data augmentation resulted in a 3.82% test error [3]. Replacing max-pooling with pseudorandom overlapping FMP:

$$(64nC2 - FMP\sqrt{2})_7 - (C2 - MP2 - C1)_2 - C2 - C1 - \text{output}$$

results in test errors of 3.26% (1 test) and 2.97% (12 tests).

4.4 CIFAR-10 with dropout and training data augmentation

For CIFAR-10 we used dropout and extended the training set using affine transformations: a randomized mix of translations, rotations, reflections, stretching, and shearing operations. We also added random shifts to the pictures in RGB colorspace. For a final 10 training epochs, we trained the network without the affine transformations.

For comparison, human performance on CIFAR-10 is estimated to be 6%[2]. A recent Kaggle competition relating to CIFAR-10 was won with a test error of 4.47%[3] using the same training data augmentation scheme, and architecture

$$(300nC2 - 300nC2 - MP2)_5 - C2 - C1 - \text{output}.$$

[2] `http://karpathy.ca/myblog/?p=160`
[3] `https://www.kaggle.com/c/cifar-10/`

Using a pseudorandom overlapping pooling FMP network

$$(160nC2 - FMP\sqrt[3]{2})_{12} - C2 - C1 - \text{output}.$$

we obtained test errors of 4.50% (1 test), 3.67% (12 tests) and 3.47% (100 tests).

5 Conclusions

We have trained convolutional networks with fractional max-pooling on a number of popular datasets and found substantial improvements in performance. Overlapping FMP seems to be better than disjoint FMP. Pseudorandom pooling regions seem to do better than random pooling regions when training data augmentation is used. It is possible that random pooling might regain the upperhand if we fine-tuned the amount of dropout used.

Looking again at the distortions created by random pooling in Figure 2, note that the distortion is 'decomposable' into an x-axis distortion and a y-axis distortion. It might be interesting to explore pooling regions that cannot be written using equation 1, as they might encode more general kinds of distortion into the resulting convolutional networks.

References

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649, 2012.

[3] Ben Graham. Spatially-sparse convolutional neural networks. 2014.

[4] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-Supervised Nets, 2014.

[7] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *ICLR*, 2014.

[8] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang. CASIA online and offline Chinese handwriting databases. In *Proc. 11th International Conference on Document Analysis and Recognition (ICDAR), Beijing, China*, pages 37–41, 2011.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. 2014.

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Lecun, and Rob Fergus. Regularization of Neural Networks using DropConnect, 2013. JMLR W&CP 28 (3) : 1058–1066, 2013.

[13] Matthew D. Zeiler and Rob Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *ICLR 2013*.