Protected Pooling Method of Sparse Coding in Visual Classification[*]

Zhichen Zhao, Huimin Ma, and Xiaozhi Chen

Department of Electrical Engineering, Tsinghua University, 100084, China
zhaozc10@tsinghua.mail.edu.cn, mhmpub@tsinghua.edu.cn

Abstract. Sparse Coding, a popular feature coding method, has shown superior performance in visual recognition tasks. Different pooling methods, such as average pooling and max pooling, are commonly employed after feature coding. However, it has not been explained clearly what characteristic accounts for the success of pooling method. In this paper, a new pooling method, namely protected pooling, is proposed. Our method produces features putting more emphasis on weak codes. What's more, we prove that all other pooling methods follow the same rules. Experiments on Scene 15, Caltech-101 and Flowers 17 demonstrate our improvements.

1 Introduction

Feature coding, which transforms local features into a more seperatable feature space, is a common step in visual recognition tasks. The original feature coding is VQ coding (Vector Quantization Coding[8], also called Hard Coding). Later, yang[1] has introduced a helpful solution called SC (Sparse Coding). The latter has been proved to be more powerful and employed to achieve excellent performance on various datasets. Thus, we choose SC to demonstrate our proposed pooling method.

Pooling operation can capture spatial structures to a certain extent and reduce dimensionality of features. The pooling features are commonly fed into a classifier (e.g. SVM) to learn data patterns. The code produced by average pooling is the concatenation of sums of feature responses within the same grid. Max pooling keeps only the strongest response within each grid. Boureau etc. have explained the difference between these two methods[2]. Besides, more pooling methods, such as MaxExp, Gamma and AxMin[3] have been proposed and proved to be useful.

In this paper, we are hammering at finding out the critical reason that affects results of pooling method. Also, we compared different pooling methods on datasets, inspected the property of them. At last, we raised a new pooling method to improve the accuracy, and carried out some verified experiments to prove our improvement.

[*] This work was supported by National Science Foundation of China(No.61171113).

2 Recent Works

2.1 Primary Pooling Methods

For present a image, we need to compute a single feature vector to classify. Let $\mathbf{U}$ (a matrix) be the result after the coding. We compute the image feature by using pooling function

$$\mathbf{z} = \mathcal{F}(\mathbf{U}) \tag{1}$$

where $\mathcal{F}$ is defined on each column of $\mathbf{U}$. Different pooling functions construct different image statistics. In VQ coding, we always use the *averaging* pooling

$$\mathbf{z} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{u}_m \tag{2}$$

which is also known as *histogram*. People names it according to its most obvious feature: counting numbers of descriptors belong to some "word", just like a histogram using on codebook. Of course this function can be used in SC, but in SC there is a more efficient pooling called *max* pooling

$$\mathbf{z}_j = \max\{|\mathbf{u}_{1j}|, ...|\mathbf{u}_{Mj}|\} \tag{3}$$

where $\mathbf{z}_j$ is the j-th element of $\mathbf{z}$, $\mathbf{u}_{ij}$ is the (i, j) entry of $\mathbf{U}$. Max pooling is thought to be supported by biophysical evidence in visual cortex. In most applications, max pooling has been found to be superior than many other pooling methods including average pooling.

2.2 Recent Pooling Methods

Likelihood based pooling methods have recently shed new light on the role of the pooling step in Bag-of-Words. Under the hypothesis of Bernoulli distribution, the mean of max pooling could be expressed as $1-(1-\alpha)^M$[2], Estimating α as the average of mid-level feature activations. Inspired by this, a pooling method named *MaxExp* is proposed, defined by

$$\mathbf{z} = 1 - (1 - \frac{1}{M}\sum_{m=1}^{M} \mathbf{u}_m)^{\widehat{M}}, \widehat{M} = |M| \tag{4}$$

M is the number of elements. To using this method we should confirm that $0 \leq \mathbf{u}_m \leq 1$ or the result will be totally wrong because there may appear negative elements in $\mathbf{z}$. This constraint is always satisfied in SC coding.

Another method, called *Gamma*, was introduced from a phenomenon that a given visual word appears in an image more often than is statistically expected[4]. Gamma acts on average pooling to improve the similarity of image signatures belonging to each class of objects and it is defined by:

$$\mathbf{z} = (\frac{1}{M}\sum_{m=1}^{M} \mathbf{u}_m)^{\gamma} \tag{5}$$

The correction factor $0 < \gamma < 1$ is usually chosen by cross-validation. It is closely related to MaxExp. It is not difficult for us to find that these two methods both promote the function curve beyond $y = x$ (fig. 1). This property will be discussed later and our following method takes this more completely.

Knoiusz[3] introduced a close approximation of MaxExp that has a parameter β accounting for the interdependence of descriptors. The method, named Approximate Pooling *(AxMin)*, is defined by:

$$\mathbf{z} = \min[1, \beta(\frac{1}{M}\sum_{m=1}^{M}\mathbf{u}_m)] \tag{6}$$

AxMin pooling implies that the confidence in the visual word can increase until it reaches the saturation threshold (full confidence). Once reached, any strong variations have no effect. This means elements have different status according to their value, and big enough elements needs less details than small ones.

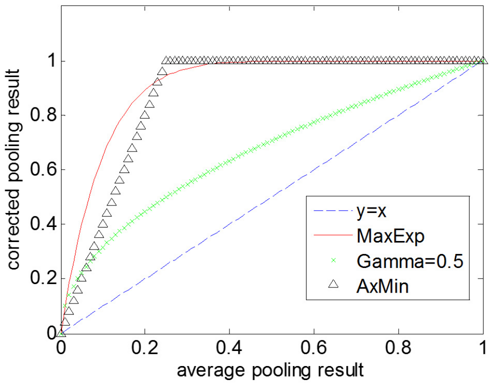

Fig. 1. A figure of these three method: MaxExp, Gamma and AxMin. They are all functions using average pooling result as independent variable and promote data bigger than $y = x$, the reason will be explained in section 3.2.

All these three methods are effective in application. For AxMin, Knoiusz made improvement by taking only the top n largest elements to calculate average each column while ignoring others. It is obvious that max pooling is its special case where n=1, so it equals to taking self-adaption between 1 and M. In section 3, we will discuss what influences performance among these methods and design a new method to obtain better pooling features.

3 Methods

Through the discuss above, we found that all the three methods, MaxExp, Gamma and AxMin, boost average pooling codes in certain ways (Fig 1). This enlightens us to find out the critical factor that affects pooling method and it is exactly our stand point. In this section, we will give discussion on characteristics of pooling and feature coding, then find out the factor behind different methods. First, we will begin from average and max pooling.

3.1 Behind Average and Max Pooling

During researching on the average pooling and max pooling. An interesting phenomenon drew our attention: the data of $\mathbf{z}$ after max pooling always has smaller variance and its minimum except zero is bigger than that of average pooling. The fact reminded us that a very small data nearby zero will be baneful. we found that data in average pooling has a bigger gap, which makes tiny numbers approach zero after normalization. Too tiny elements are easy to be ignored and message will be lost. The advantage of max pooling is the protection of tiny data and figuring out if there exists the word in codebook correctly. Notice that max pooling indeed reduces the variance but variance is not the critical factor. More insights will be seen through experiments in section 4.

Max pooling reaches a better result. It could be seen as a improvement of average pooling, though there is no functional relationship between average and max pooling. It also gives an important thinking that if we could boost these tiny elements without changing the order in data, we could probably obtain a more accurate result.

3.2 Theory with Pooling Methods

With the help of analysis above, a new way comes to be reasonable: Let average pooling result be our input, we could find a function to modify it as

$$\mathbf{y} = \mathcal{F}(\frac{1}{M}\sum_{m=1}^{M}\mathbf{u}_m) \tag{7}$$

Where $\mathbf{y}$ is a change over $\mathbf{z}$, average could give basic messages we need, so usage of the function will modify data as we expect. Of course we need to hold most properties obtained by average pooling such as order and existence of messages. To modify correctly, this function should have properties below

- $\mathcal{F}(0) = 0$ and $\mathcal{F}(1) = 1$.
  Means no extra messages or change of the boundary.
- $\forall x_1 \leq x_2, \mathcal{F}(x_1) \leq \mathcal{F}(x_2)$.
  A monotone increasing function will not change the order of data, so does the status behind them.
- $\mathcal{F}''(x) \leq 0, \forall x \in [0, 1]$.
  A concave function is required for boosting weak codes.

In fact, there are already methods satisfying the conditions above. We could easily check out that MaxExp, Gamma and AxMin are all up to these rules. This truth also confirms our theory. Notice even though a concave function could be helpful, the degree should be controlled, otherwise it will bring bad consequence finally. As the goal of designing above rules is to protect the weak codes which may contain discriminative information, we define all these methods "Protected Pooling Methods".

3.3 Improve the Method Thoroughly

Among MaxExp, Gamma and AxMin, AxMin@n[3] performs best in many occasions. It finds where the data should be protected and where the details of order are not important. However, using two straight lines is two simple to make full use of the above properties and the roughness around intersection point also brings errors. Further more, parameter β is difficult to control in SC because data in $\mathbf{U}$ are too small. According to those points, we improve AxMin@n in 3 views:

– use a concave function to replace straight line, it is supposed to protect weak codes and it is more smooth around intersection point.
– use sum operation instead of average to be input. Data in SC are small while M is big(nearly 2000). Division on average result may lose precision. In fact, AxMin@n is a ideal model for its using on n largest numbers, we will also use this improvement.
– change parameter β to be horizontal, the place we choose where to cut off should be decided by the value of funtion rather than slope. What's more, a horizontal parameter is easier to tune.

Above all, we choose $\log_2(x+1)$ to be the concave function. Using all the improvement, the final pooling method we use is

$$\mathbf{z} = \min[\log_2(\sum_{m=1}^{N} \mathbf{u'}_m + 1), \beta] \tag{8}$$

Where $\mathbf{u'}$ has been sorted from large to small, β is the parameter controlling position to cut and N is the number of top largest elements we used. we call this function as *protected-AxMin@n(p-AxMin@n)*. And this, is our **Protected Pooling method**.

As so far, the whole method has been introduced. To check the accuracy of our improvements, in the next section, we will carry out experiments to confirm our theory and compare pooling methods in different datasets.

4 Experiments

Firstly we analyze the influence of different factors such as the completeness of information and the degree of boosting weak codes. We design an experiment using Scene 15[11] dataset. A codebook with 1K words is trained and SVM is used as classifier throughout all experiments. Table 1 is our results.

Table 1. Classification rate compared on Scene 15

Pooling Method	classification rate
bool	75.35± 0.64
average	76.30± 0.87
sum	78.96± 0.53
ex-variance	77.66± 0.47
re-variance	77.21± 0.56

The method "bool" refers to a method that operate on average result. Set all non-zero elements in **z** to be 1. Notice that it floats all the details in **z** but could get a near accuracy with average pooling. This verified our theory that the completeness of messages is the most important point. Ex-variance expands the variance, re-variance reduces the variance, this two results implies variance is not the critical factor. Moreover, we could find sum pooling performs better than average pooling just because of its protection on precision. A division operation may be usual but in SC we should be careful.

4.1 Results on Scene 15

We tried our algorithm on the Scene 15 dataset. This dataset contains totally 4485 images about 15 categories, including living room, kitchen and so on. we took 100 images per class for training and the left for testing (the same as yang[1]). Comparison results are shown in Table 2. It could be seen easily that with or without @n, protected(p-) methods always perform better than previous methods.

Table 2. Classification rate compared on Scene 15

Pooling Method	classification rate
max[1]	80.28± 0.93
max	81.30± 0.49
Gamma	80.62± 0.99
AxMin	82.13± 0.92
p-AxMin	82.59± 0.81
AxMin@n	82.42± 0.77
p-AxMin@n	**82.79± 0.85**

4.2 Results on Caltech101

The Caltech-101 dataset[9] contains 101 classes, we follow yang's condition to experiment. We take 30 images per class to train and the size of codebook is 1K.

Notice that AxMin@n has a lower accuracy compared with max pooling. Max pooling could be regarded as a method to protect weak codes from average pooling though there is no explicit mapping function. MaxExp, Gamma and

Table 3. Classification rate compared on Caltech101

Pooling Method	classification rate
max[1]	73.20± 0.54
max	**72.32± 0.61**
AxMin@n	71.60± 0.97
p-AxMin@n	72.28± 0.56

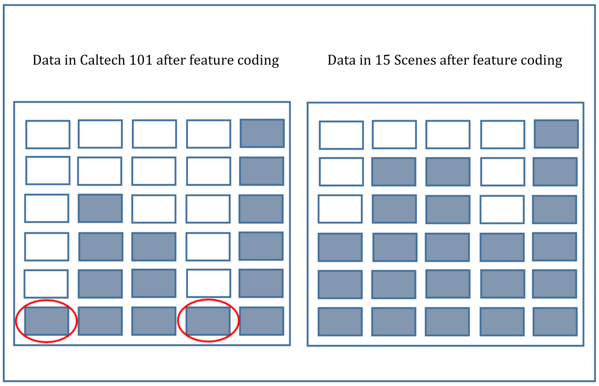
Data in Caltech 101 after feature coding
Data in 15 Scenes after feature coding

Fig. 2. Data in two datasets after feature coding. White elements are zero ones, black elements are nonzero ones. Max pooling is especially effectively to "isolated point".

AxMin do the same thing. Using these methods can insure better accuracy than average pooling, but AxMin@n not always obtain better accuracy than max pooling because they both protect weak codes just in different ways.

Further on, we will find out the cause of different display on Caltech101 and Scene 15. The data $\mathbf{u}$ (has been sorted from small to large) is showed in Figure 2. of two different datasets. Data in red circle shows there are many "isolated point". Using AxMin@n, we can not protected them to the maximum extent. However it is easy for max pooling. This phenomenon also agrees our theory.

4.3 Results on Flowers 17

The dataset Flowers 17[12] consists of 17 classes of flowers with 80 images of each. Characteristic has changed from a problem to another, but it doesn't affect the comparison with different pooling methods. The size of codebook in Flower 17 is also 1K and results are showed in Table 4

Notice that flowers have their own characteristics. because of the low baseline, our method could get more gain from other methods. With results on 3 different datasets our theory proves to be correct, and our method is efficient. So we can draw the conclusion that the protection of weak codes is the critical factor in pooling.

Table 4. classification rate compared on Flowers 17

Pooling Method	classification rate
max	67.29± 0.87
AxMin@n	67.78± 1.10
p-AxMin@n	**68.76± 0.71**

5 Conclusion

We have analyzed several popular pooling methods and proposed a hypothesis that protecting weak codes is the critical factors for pooling operation. The design of a proper pooling strategy is supposed to follow three rules: maintaining the completeness of information, preserving magnitude order and boosting weak codes. Experiments have demonstrated improvements over previous pooling methods. The proposed pooling rules are independent of feature coding stage, so combining both feature coding and pooling to design a unified coding-pooling strategy is supposed to be promising, which is our future work.

References

1. Yang, J., Yu, K., Gong, Y.: Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In: CVPR (2009)
2. Boureau, Y., Ponce, J., LeCun, Y.: A Theoretical Analysis of Feature Pooling in Visual Recognition. In: ICML (2010)
3. Koniusz, P., Yan, F., Mikolajczyk, K.: Comparison of Mid-Level Feature Coding Approaches And Pooling Strategies in Visual Concept Detection. CVIU 117(5), 479–492 (2013)
4. Jegou, H., Douze, M., Schmid, C.: On the Burstiness of Visual Elements. In: CVPR (2009)
5. Yu, K., Zhang, T., Gong, Y.: Nonlinear Learning using Local Coordinate Coding. In: NIPS (2009)
6. Zhou, X., Yu, K., Zhang, T., Huang, T.S.: Image Classification Using Super-Vector Coding of Local Image Descriptors. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part V. LNCS, vol. 6315, pp. 141–154. Springer, Heidelberg (2010)
7. Wang, J., Yang, J., Yu, K., Lv, F., Huang, T.S., Gong, Y.: Locality-constrained Linear Coding for Image Classification. In: CVPR (2010)
8. Chatfield, K., Lempitsky, V., Vedaldi, A., Zisserman, A.: The devil is in the details: an evaluation of recent feature encoding methods. In: BMVC (2011)
9. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In: GMBV (2004), `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`
10. Chang, C.C., Lin, C.J.: LIBSVM-a library for support vector machines
11. Lazebnik, S., Schmid, C., Ponce, J.: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In: Lazebnik, S., Schmid, C., Ponce, J. (eds.) CVPR (2006), `http://www-cvr.ai.uiuc.edu/ponce_grp/data/`
12. Nilsback, M.-E., Zisserman, A.: A Visual Vocabulary for Flower Classification. In: CVPR (2006), `http://www.robots.ox.ac.uk/~vgg/data/flowers/17/`