Learnable Pooling Regions for Image Classification

Mateusz Malinowski
Computer Vision and Multimodal Computing
Max Planck Institute for Informatics
Campus E1 4, 66123 Saarbrücken, Germany
mmalinow at mpi-inf.mpg.de

Mario Fritz
Computer Vision and Multimodal Computing
Max Planck Institute for Informatics
Campus E1 4, 66123 Saarbrücken, Germany
mfritz at mpi-inf.mpg.de

Abstract

From the early HMAX model to Spatial Pyramid Matching, pooling has played an important role in visual recognition pipelines. Spatial pooling, by grouping of local codes, equips these methods with a certain degree of robustness to translation and deformation yet preserving important spatial information. Despite the predominance of this approach in current recognition systems, we have seen little progress to fully adapt the pooling strategy to the task at hand. This paper proposes a model for learning task dependent pooling scheme – including previously proposed hand-crafted pooling schemes as a particular instantiation. In our work, we investigate the role of different regularization terms showing that the smooth regularization term is crucial to achieve strong performance using the presented architecture. Finally, we propose an efficient and parallel method to train the model. Our experiments show improved performance over hand-crafted pooling schemes on the CIFAR-10 and CIFAR-100 datasets – in particular improving the state-of-the-art to 56.29% on the latter.

1 Introduction

Spatial pooling plays a crucial role in modern object recognition and detection systems. Motivated from biology [Hubel and Wiesel, 1962, Fukushima and Miyake, 1982, LeCun et al., 1990, Riesenhuber and Poggio, 2009] and statistics of locally orderless images [Koenderink and Van Doorn, 1999], the spatial pooling approach has been found useful as an intermediate step of many today's computer vision methods. For instance, the most popular visual descriptors such as SIFT [Lowe, 2004] and HOG [Dalal and Triggs, 2005], which compute local histograms of gradients, can be in fact seen as a special version of the spatial pooling strategy. In order to form more robust features under translation or small object deformations, activations of the codes are pooled over larger areas in a spatial pyramid scheme [Lazebnik et al., 2006, Yang et al., 2009]. Unfortunately, this critical decision, namely the spatial division, is most prominently based on hand-crafted algorithms and therefore data independent.

Related Work As large amounts of training data is available to us today, there is an increasing interest to push the boundary of learning based approaches towards fully optimized and adaptive architectures where design choices, that would potentially constrain or bias a model, are kept to a minimum. Neural networks have a great tradition of approaching hierarchical learning problems and training intermediate representations [Ranzato et al., 2007, Le et al., 2012a]. Along this line, we propose a learnable spatial pooling strategy that can shape the pooling regions in a discriminative manner. Our architecture has a direct interpretation as a pooling strategy and therefore subsumes popular spatial pyramids as a special case. Yet we have the freedom to investigate different regularization terms that lead to new pooling strategies when optimized jointly with the classifier.

Recent progress has been made in learning pooling regions in the context of image classification using the Spatial Pyramid Matching (SPM) pipeline [Lazebnik et al., 2006, Yang et al., 2009]. Jia

and Huang [2011], Jia et al. [2012] and Feng et al. [2011] have investigated how to further liberate the recognition from preconceptions of the hand crafted recognition pipelines, and include the pooling strategy into the optimization framework jointly with the classifier. However, these methods still make strong assumptions on the solutions that can be achieved. For instance Jia and Huang [2011] optimizes binary pooling strategies that are given by the superposition of rectangular basis functions, and Feng et al. [2011] finds pooling regions by applying a linear discriminant analysis for individual pooling strategies and training a classifier afterwards. Also as opposed to Ranzato and Hinton [2010], we aim for discriminative pooling over large neighborhoods in the SPM fashion where the information about the image class membership is available during training.

Outline We question restrictions imposed by the above methods and suggest to learn pooling strategies under weaker assumptions. Indeed, our method discovers new pooling shapes that were not found previously as they were suppressed by the more restrictive settings.

The generality that we are aiming for comes at the price of a high dimensional parameters space. This manifests in a complex optimization problem that is more demanding on memory requirements as well as computations needs, not to mention a possibility of over-fitting. Therefore, we also discuss two approximations to our method. First approximation introduces a pre-pooling step and therefore reduces the spatial dimension of the codes. The second approximation divides the codes into a set of smaller batches (subset of codes) that can be optimized independently and therefore in parallel.

Finally, we evaluate our method on the CIFAR-10 and show strong improvements over hand-crafted pooling schemes in the regime of small dictionaries where our more flexible model shows its capability to make best use of the representation by exploring spatial pooling strategies specific to each coordinate of the code. Despite the diminishing return, the performance improvements persist up to largest codes we have investigated. We also show strong classification performance on the CIFAR-100 dataset where our method outperforms, to the best of our knowledge, the state-of-the-art.

2 Method

As opposed to the methods that use fixed spatial pooling regions in the object classification task [Lazebnik et al., 2006, Yang et al., 2009] our method jointly optimizes both the classifier and the pooling regions. In this way, the learning signal available in the classifier can help shaping the pooling regions in order to arrive at better pooled features.

2.1 Parameterized pooling operator

The simplest form of the spatial pooling is computing histogram over the whole image. This can be expressed as $\Sigma(\boldsymbol{U}) := \sum_{j=1}^{M} \boldsymbol{u}_j$, where $\boldsymbol{u}_j \in \mathbb{R}^K$ is a code (out of M such codes) and an index j refers to the spatial location that the code originates from[1]. A code is an encoded patch extracted from the image. The proposed method is agnostic to the patch extraction method and encoding scheme. Since the pooling approach looses spatial information of the codes, Lazebnik et al. [2006] proposed to first divide the image into subregions, and afterwards to create pooled features by concatenating histograms computed over each subregion. There are two problems with such an approach: first, the division is largely arbitrary and in particular independent of the data; second, discretization artifacts occur as spatially nearby codes can belong to two different regions as the 'hard' division is made.

In this paper we address both problems by using a parameterized version of the pooling operator

$$\Theta_{\boldsymbol{w}}(\boldsymbol{U}) := \sum_{j=1}^{M} \boldsymbol{w}_j \circ \boldsymbol{u}_j \tag{1}$$

where $\boldsymbol{a} \circ \boldsymbol{b}$ is the element-wise multiplication. Standard spatial division of the image can be recovered from Formula 1 by setting the vectors $\boldsymbol{w}_j$ either to a vector of zeros $\mathbf{0}$, or ones $\mathbf{1}$. For instance, features obtained from dividing the image into 2 subregions can be recovered from Θ by

[1] That is $j = (x, y)$ where x and y refer to the spatial location of the center of the extracted patch.

concatenating two vectors: $\sum_{j=1}^{\frac{M}{2}} \mathbf{1} \circ \boldsymbol{u}_j + \sum_{j=\frac{M}{2}+1}^{M} \mathbf{0} \circ \boldsymbol{u}_j$, and $\sum_{j=1}^{\frac{M}{2}} \mathbf{0} \circ \boldsymbol{u}_j + \sum_{j=\frac{M}{2}+1}^{M} \mathbf{1} \circ \boldsymbol{u}_j$, where $\left\{1, ..., \frac{M}{2}\right\}$ and $\left\{\frac{M}{2}+1, ..., M\right\}$ refer to the first and second half of the image respectively.

In general, let $\mathfrak{F} := \{\Theta_{\boldsymbol{w}}\}_{\boldsymbol{w}}$ be a family of the pooling functions given by Eq. 1, parameterized by the vector $\boldsymbol{w}$, and let $\boldsymbol{w}^{*,l}$ be the 'best' parameter chosen from the family $\mathfrak{F}$ based on the initial configuration l and a given set of images.[2] First row of Figure 2 shows four initial configurations that mimic the standard 2-by-2 spatial image division. Every initial configuration can lead to different $\boldsymbol{w}^{*,l}$ as it is shown in Figure 2. Clearly, the family $\mathfrak{F}$ contains all possible 'soft' and 'hard' spatial divisions of the image, and therefore can be considered as their generalization.

2.2 Learnable pooling regions

In SPM architectures the pooling weights $\boldsymbol{w}$ are designed by hand, here we aim for joint learning $\boldsymbol{w}$ together with the parameters of the classifier. Intuitively, the classifier during training has access to the classes that the images belong to, and therefore can shape the pooling regions. On the other hand, the method aggregates statistics of the codes over such learnt regions and pass them to the classifier allowing to achieve higher accuracy. Such joint training of the classifier and the pooling regions can be done by adapting the backpropagation algorithm [Bishop, 1999, LeCun et al., 1998], and so can be interpreted as a densely connected multilayer perceptron [Collobert and Bengio, 2004, Bishop, 1999].

Consider a sampling scheme and an encoding method producing M codes each K dimensional. Every coordinate of the code is an input layer for the multilayer perceptron. Then we connect every j-th input unit at the layer k to the l-th pooling unit a_l^k via the relation $w_{lj}^k u_j^k$. Since the receptive field of the pooling unit a_l^k consists of all codes at the layer k, we have $a_l^k := \sum_{j=1}^{M} w_{lj}^k u_j^k$, and so in the vector notation

$$\boldsymbol{a}_l := \sum_{j=1}^{M} \boldsymbol{w}_j^l \circ \boldsymbol{u}_j = \Theta_{\boldsymbol{w}^l}(\boldsymbol{U}) \tag{2}$$

Next, we connect all pooling units with the classifier allowing the information to circulate between the pooling layers and the classifier.

Although our method is independent of the choice of a dictionary and an encoding scheme, in this work we use K-means with triangle coding $f_k(\boldsymbol{x}) := \max\{0, \mu(\boldsymbol{z}) - z_k\}$ [Coates et al., 2011].

Similarly, every multi-class classifier that can be interpreted in terms of an artificial neural network can be used. In our work we employ logistic regression. This classifier is connected to the pooling units via the formula

$$J(\boldsymbol{\Theta}) := -\frac{1}{D} \sum_{i=1}^{D} \sum_{j=1}^{C} \mathbf{1}\{y^{(i)} = j\} \log p(y^{(i)} = j | \boldsymbol{a}^{(i)}; \boldsymbol{\Theta}) \tag{3}$$

where D denotes the number of all images, C is the number of all classes, $y^{(i)}$ is a label assigned to the i-th input image, and $\boldsymbol{a}^{(i)}$ are responses from the 'stacked' pooling units $[\boldsymbol{a}_l]_l$ for the i-th image[3]. We use the logistic function to represent the probabilities: $p(y = j|\boldsymbol{x}; \boldsymbol{\Theta}) := \frac{\exp(\boldsymbol{\theta}_j^T \boldsymbol{x})}{\sum_{l=1}^{C} \exp(\boldsymbol{\theta}_l^T \boldsymbol{x})}$. Since the classifier is connected to the pooling units, our task is to learn jointly the pooling parameters $\boldsymbol{W}$ together with the classifier parameters $\boldsymbol{\Theta}$, where $\boldsymbol{W}$ is the matrix containing all pooling weights.

Finally, we use standard gradient descent algorithm that updates the parameters using the following fixed point iteration

$$\boldsymbol{X}^{t+1} := \boldsymbol{X}^t - \gamma \nabla J(\boldsymbol{X}^t) \tag{4}$$

where in our case $\boldsymbol{X}$ is a vector consisting of the pooling parameters $\boldsymbol{W}$ and the classifier parameters $\boldsymbol{\Theta}$. In practice, however, we employ a quasi-Newton algorithm LBFGS[4].

[2] We will show the learning procedure that can select such parameter vectors in the following subsection.

[3] Providing the codes $\boldsymbol{U}^{(i)}$ are collected from the i-th image and $\boldsymbol{a}_l^{(i)} := \Theta_{\boldsymbol{w}^l}(\boldsymbol{U}^{(i)})$ then $\boldsymbol{a}^{(i)} := [\boldsymbol{a}_l^{(i)}]_l$.

[4] The algorithm, developed by Mark Schmidt, can be downloaded from the following webpage: http://www.di.ens.fr/ mschmidt/Software/minFunc.html

2.3 Regularization terms

In order to improve the generalization, we introduce regularization of our network as we deal with a large number of the parameters. For the classification $\boldsymbol{\Theta}$ and pooling parameters $\boldsymbol{W}$, we employ a simple L_2 regularization terms: $||\boldsymbol{\Theta}||_{l_2}^2$ and $\sum_k ||\boldsymbol{W}^k||_{l_2}^2$. We improve the interpretability of the pooling weights as well as to facilitate a transfer among models by adding a projection onto a unit cube. To reduce quantization artifacts of the pooling strategy as well as to ensure smoothness of the output w.r.t. small translations of the image, the model penalizes weights whenever the pooling surface is non-smooth. This can be done by measuring the spatial variation, that is $||\nabla_x \boldsymbol{W}^k||_{l_2}^2 + ||\nabla_y \boldsymbol{W}^k||_{l_2}^2$ for every layer k. This regularization enforces soft transition between the pooling subregions.

Every regularization term comes with its own hyper-parameter set by cross-validation. The overall objective that we want to optimize is

$$\begin{aligned}
&\underset{\boldsymbol{W},\boldsymbol{\Theta}}{\text{minimize}}\, J_{\text{R}}(\boldsymbol{\Theta}, \boldsymbol{W}) := \\
&-\frac{1}{D}\sum_{i=1}^{D}\sum_{j=1}^{C} \mathbf{1}\{y^{(i)}=j\} \log p(y^{(i)}=j|\boldsymbol{a}^{(i)};\boldsymbol{\Theta}) \\
&+\frac{\alpha_1}{2}||\boldsymbol{\Theta}||_{l_2}^2 + \frac{\alpha_2}{2}||\boldsymbol{W}||_{l_2}^2 \\
&+\frac{\alpha_3}{2}\left(||\nabla_x \boldsymbol{W}||_{l_2}^2 + ||\nabla_y \boldsymbol{W}||_{l_2}^2\right) \\
&\\
&\text{subject to } \boldsymbol{W} \in [0,1]^{K\times M\times L}
\end{aligned} \tag{5}$$

where $\boldsymbol{a}_l$ is the l-th pooling unit described by Formula 2, and $||\boldsymbol{W}||_{l_2}$ is the Frobenius norm.

2.4 Approximation of the model

The presented approach is demanding to train in the means of the CPU time and memory storage when using high dimensional representations. That is, the number of the pooling parameters to learn grows as $K \times M \times L$, where K is dimensionality of codes, M is the number of patches taken from the image and L is the number of pooling units. Therefore, we propose two approximations to our method making the whole approach more scalable towards bigger dictionaries. However, we emphasize that learnt pooling regions have very little if any overhead compared to standard spatial division approaches at test time.

First approximation does a fine-grained spatial partition of the image, and then pools the codes over such subregions. This operation, we call it a pre-pooling step, reduces the number of considered spatial locations by the factor of the pre-pooling size. For instance, if we collect M codes and the pre-pooling size is S per dimension, then we reduce the number of codes to a number $\frac{M}{S^2}$. The pre-pooling operation fits well into our generalization of the SPM architectures as by choosing $S := \frac{M}{2}$ we obtain a weighted quadrants scheme. Moreover, the modeler has the option to start with the larger S when little data is available and gradually decreases S as more parameters can be learnt using more data.

The second approximation divides a K dimensional code into $\frac{K}{D}$ batches, each D dimensional (where $D \leq K$ and K is divisible by D). Then we train our model on all such batches in parallel to obtain the pooling weights. Later, we train the classifier on top of the concatenation of the trained, partial models. As opposed to Le et al. [2012b] our training is fully independent and doesn't need communication between different machines.

Since the ordering of the codes is arbitrary, we also consider D dimensional batches formed from the permuted version of the original codes, and combine them together with the concatenated batches to boost the classification accuracy (we call this approximation redundant batches). Given a fixed sized dictionary, this approximation performs slightly better, although it comes at the cost of increased number of features due to the redundant batches.

Finally, our approximations not only lead to a highly parallel training procedure with reduced memory requirements and computational demands, but also have shown to greatly reduce the number of required iterations as they tend to converge roughly 5 times faster than the full model on large dictionaries.

3 Experimental Results

We evaluate our method on the CIFAR-10 and CIFAR-100 datasets [Krizhevsky and Hinton, 2010]. Furthermore, we provide insights into the learnt pooling strategies as well as investigate transfer between datasets. In this section we describe our experimental setup, and present our results on both datasets.

3.1 CIFAR-10 and CIFAR-100 datasets

The CIFAR-10 and CIFAR-100 datasets contain 50000 training color images and 10000 test color images from respectively 10 and 100 categories, with 6000 and 600 images per class respectively. All images have the same size: 32×32 pixels, and were sampled from the 80 million tiny images dataset [Torralba et al., 2008].

3.2 Evaluation pipeline

In this work, we follow the Coates and Ng [2011] pipeline. We extract normalized and whitened 6×6 patches from images using a dense, equispaced grid with a unit sample spacing. As the next step, we employ the K-means assignment and triangle encoding [Coates and Ng, 2011, Coates et al., 2011] to compute codes – a K-dimensional representation of the patch. We classify images using either a logistic regression, or a linear SVM in the case of transferred pooling regions. Optionally we use two approximations described in subsection 2.4. As we want to be comparable to Coates et al. [2011], who use a spatial division into 2-by-2 subregions which results in $4 \cdot K$ pooled features, we use 4 pooling units. Furthermore, we use standard division (first row of Figure 2) as an initialization of our model.

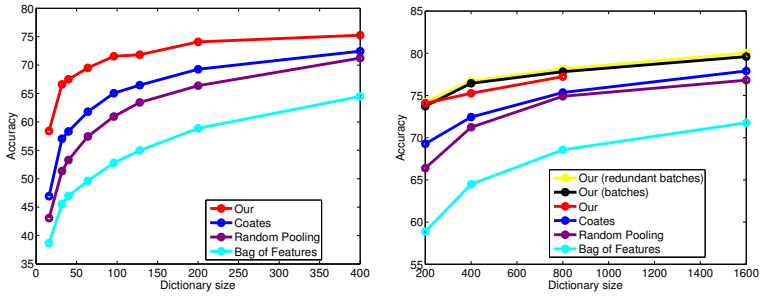
To learn parameters of the model we use the limited-memory BFGS algorithm (details are described in subsection 2.2), and limit the number iterations to 3000. After the training, we can also concatenate the results of the parameterized pooling operator $[\Theta_{\boldsymbol{w}_l}(\boldsymbol{U})]_{l=1}^{4}$. This yields a $4 \cdot K$ dimensional feature vector that can be again fed into the classifier, and trained independently with the already trained pooling regions. We call this procedure transfer of pooling regions.

The reason behind the transfer is threefold. Firstly, we can combine partial models trained with our approximation in batches to a full, originally intractable, model[5]. Secondly, the transfer process allows to combine both the codes and the learnt model from the dictionaries of different sizes. Lastly, it enables training of the pooling regions together with the classifier on one dataset, and then re-train the classifier alone on a target dataset. To transfer the pooling regions, we tried logistic regression classifier and linear SVM showing that both classifying procedures can benefit from the learnt pooling regions. However, since we achieve slightly better results for the linear SVM (about 0.5% for bigger dictionaries), only those results are reported. Similarly, we don't notice significant difference in the classification accuracy for smaller dictionaries when the pre-pooling is used (with the pre-pooling size $S := 3$), and therefore all experiments refer only to this case. Finally, we select hyper-parameters of our model based on the 5-fold cross-validation.

3.3 Evaluation of our method on small dictionaries

Figure 1(a) shows the classification accuracy of our full method against the baseline [Coates and Ng, 2011]. Since we train the pooling regions without any approximations in this set of experiments the results are limited to dictionary sizes up to 800. Our method outperforms the approach of Coates by 10% for dictionary size 16 (our method achieves the accuracy 57.07%, whereas the baseline only 46.93%). This improvement is consistent up to the bigger dictionaries although the margin is getting

[5]The reader can find details of such approximation in subsection 2.4.



(a) (b)

Figure 1: Figure 1(a) shows accuracy of the classification with respect to the number of dictionary elements on smaller dictionaries. Figure 1(b) shows the accuracy of the classification for bigger dictionaries when batches, and the redundant batches were used. Experiments are done on CIFAR-10.

smaller. Our method is about 2.5% and 1.88% better than the baseline for 400 and 800 dictionary elements respectively.

3.4 Scaling up to sizable dictionaries

In subsection 2.4 we have discussed the possibility of dividing the codes into low dimensional batches and learning the pooling regions on those. In the following experiments we use batches with 40 coordinates extracted from the original code, as those fit conveniently into the memory of a single, standard machine (about 5 Gbytes for the main data) and can all be trained in parallel.

Besides a reduction in the memory requirements, the batches have shown multiple benefits in practice due to smaller number of parameters. We need less computations per iterations as well as observe faster convergence. Figure 1(b) shows the classification performance for larger dictionaries where we examined the full model [Our], the baseline [Coates], random pooling regions (described in subsection 3.5), bag of features, and two possible approximation - the batched model [Our (batches)], and the redundantly batched model [Our (redundant batches)].

Our test results are presented in Table 1. When comparing our full model to the approximated versions with batches for dictionaries of size 200, 400 and 800, we observe that there is almost no drop in performance and we even slightly improve for the bigger dictionaries. We attribute this to the better conditioned learning problem of the smaller codes within one batch. With an accuracy for the batched model of 79.6% we outperform the Coates baseline by 1.7%. Interestingly, we gain another small improvement to 80.02% by adding redundant batches which amounts to a total improvement of 2.12% compared to the baseline. Our method performs comparable to the pooling strategy of Jia and Huang [2011] which uses more restrictive assumptions on the pooling regions and employs feature selection algorithm.

Method	Dict. size	Features	Acc.
Jia	1600	6400	80.17%
Coates	1600	6400	77.9%
Our (batches)	1600	6400	79.6%
Our (redundant)	1600	12800	80.02%

Table 1: Comparison of our methods against the baseline [Coates and Ng, 2011] and Jia and Huang [2011] with respect to the dictionary size, number of features and the test accuracy on CIFAR-10.

To the best of our knowledge Ciresan et al. [2012] achieves the best results on the CIFAR-10 dataset with an accuracy 88.79% with a method based on a deep architecture – different type of architecture to the one that we investigate in our study.

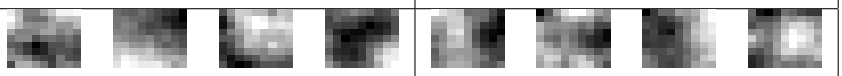
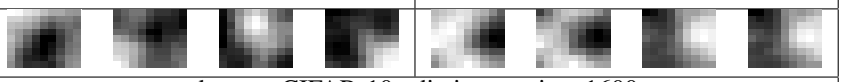
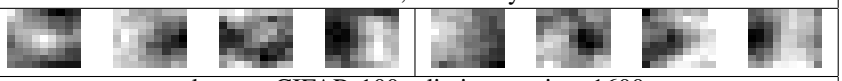
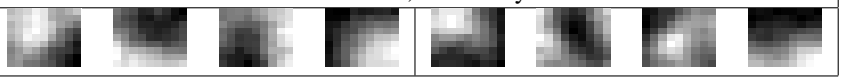
regularization	pooling weights
dataset: CIFAR-10 ; dictionary size: 200	
Coates (no learn.)	
l2	
smooth	
smooth & l2	
dataset: CIFAR-10 ; dictionary size: 1600	
smooth & batches	
dataset: CIFAR-100 ; dictionary size: 1600	
smooth & batches	

Table 2: Visualization of different pooling strategies obtained for different regularizations, datasets and dictionary size. Every column shows the regions from two different coordinates of the codes. First row presents the initial configuration also used in standard hand-crafted pooling methods. Brighter regions denote larger weights.

3.5 Random pooling regions

Our investigation also includes results using random pooling regions where the weights for the parameterized operator (Eq. 2) were sampled from normal distribution with mean 0.5 and standard deviation 0.1, that is $\boldsymbol{w}_j^l \sim \mathcal{N}(0.5, 0.1)$ for all l. This notion of the random pooling differs from the Jia et al. [2012] where random selection of rectangles is used. The experiments show that the random pooling regions can compete with the standard spatial pooling (Figure 1(a) and 1(b)) on the CIFAR-10 dataset, and suggest that random projection can still preserve some spatial information. This is especially visible in the regime of bigger dictionaries where the difference is only 1.09%. The obtained results indicate that hand-crafted division of the image into subregions is questionable, and call for a learning-based approach.

3.6 Investigation of the regularization terms

Our model (Eq. 5) comes with two regularization terms associated with the pooling weights, each imposing different assumptions on the pooling regions. Hence, it is interesting to investigate their role in the classification task by considering all possible subsets of $\{\text{l2}, \text{smooth}\}$, where "l2" and "smooth" refer to $||\boldsymbol{W}||_{l_2}^2$ and $\left(||\nabla_x \boldsymbol{W}||_{l_2}^2 + ||\nabla_y \boldsymbol{W}||_{l_2}^2\right)$ respectively.

Table 3 shows our results on CIFAR-10. We choose a dictionary size of 200 for these experiments, so that we can evaluate different regularization terms without any approximations. We conclude that the spatial smoothness regularization term is crucial to achieve a good predictive performance of our method whereas the l2-norm term can be left out, and thus also reducing the number of hyper-parameters. Based on the cross-validation results (second column of Table 3), we select this setting for further experiments.

Regularization	CV Acc.	Test Acc.
free	68.48%	69.59%
l2	67.86%	68.39%
smooth	73.36%	73.96%
l2 + smooth	70.42%	70.32%

Table 3: We investigate the impact of the regularization terms on the CIFAR-10 dataset with dictionary size equals to 200. Term "free" denotes the objective function without the l2-norm and smoothness regularization terms. The cross-validation accuracy and test accuracy are shown.

3.7 Experiments on the CIFAR-100 dataset

Although the main body of work is conducted on the CIFAR-10 dataset, we also investigate how the model performs on the much more demanding CIFAR-100 dataset with 100 classes. Our model with the spatial smoothness regularization term on the 40 dimensional batches achieves 56.29% accuracy. To our best knowledge, this result consitutes the state-of-the-art performance on this dataset, outperforming Jia and Huang [2011] by 1.41%, and the baseline by 4.63%.

Method	Dict. size	Features	Acc.
Jia	1600	6400	54.88%
Coates	1600	6400	51.66%
Our (batches)	1600	6400	56.29%

Table 4: The classification accuracy on CIFAR-100, where our method is compared against the Coates and Ng [2011] (we downloaded the framework from https://sites.google.com/site/kmeanslearning, we also use 5-fold cross-validation to choose hyper-parameter C) and Jia and Huang [2011] (here we refer to the NIPS 2011 workshop paper).

3.8 Transfer of the pooling regions between datasets

Beyond the standard classification task, we also examine if the learnt pooling regions are transferrable between datasets. In this scenario the pooling regions are first trained on the source dataset and then used on the target dataset to train a new classifier. We use dictionary of 1600 with 40-dimensional batches. Our results (Table 5) suggest that the learnt pooling regions are indeed transferable between both datasets. While we observe a decrease in performance when learning the pooling strategy on the less diverse CIFAR-10 dataset, we do see improvements for learning on the richer CIFAR-100 dataset. We arrive at a test accuracy of 80.35% which is an additional improvement of 0.75% and 0.18% over our best results (batch-based approximation) and Jia and Huang [2011] respectively.

Source	Target	Accuracy
CIFAR-10	CIFAR-100	52.86%
CIFAR-100	CIFAR-10	80.35%

Table 5: We train the pooling regions on the 'Source' dataset. Next, we use such regions to train the classifier on the 'Target' dataset where the test accuracy is reported.

3.9 Visualization and analysis of pooling strategies

Table 2 visualizes different pooling strategies investigated in this paper. The first row shows the widely used rectangular spatial division of the image. The other visualizations correspond to pooling weights discovered by our model using different regularization terms, datasets and dictionary size.

The second row shows the results on CIFAR-10 with the "l2" regularization term. The pooling is most distinct from the other results, as it learns highly localized weights. This pooling strategy has also performed the worst in our investigation (Table 3).

The "smooth" pooling performs the best. Visualization shows that weights are localized but vary smoothly over the image. The weights expose a bias towards initialization shown in the first row. All methods with the spatial smoothness regularization tend to focus on similar parts of the image, however "l2 & smooth" is more conservative in spreading out the weights.

The last two rows show weights trained using our approximation by batches. From visual inspection, they show a similar level of localization and smoothness to the regions obtained without approximation. This further supports the use of our approximation into independent batches.

4 Conclusion

In this paper we propose a flexible parameterization of the pooling operator which can be trained jointly with the classifier. In this manner, we study the effect of different regularizers on the pooling regions as well as the overall system. Our study show the importance of the smooth regularization term in the presented architecture. To be able to train the large set of parameters we propose approximations to our model allowing efficient and parallel training without loss of accuracy.

Our experiments show there is a room to improve the classification accuracy by advancing the spatial pooling stage. The presented method outperforms a popular hand-crafted pooling based method and previous approaches to learn pooling strategies. While our improvements are consistent over the whole range of dictionary sizes that we have investigated, the margin is most impressive for small codes where we observe improvements up to 10% compared to the baseline of Coates. Finally, our method achieves an accuracy of 56.29% on CIFAR-100, which is to the best of our knowledge the new state-of-the-art on this dataset.

As we believe that our method is a good framework for further investigations of different pooling strategies and in order to speed-up progress on the pooling stage we will make our code publicly available at time of publication.

References

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990.

M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2009.

J. J. Koenderink and A. J. Van Doorn. The structure of locally orderless images. *International Journal of Computer Vision*, 31(2):159–168, 1999.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.

M. A. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *CVPR*, 2007.

Q. V. Le, M. A. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *ICML*, 2012a.

Y. Jia and C. Huang. Beyond spatial pyramids: Receptive field learning for pooled image features. In *NIPS Workshop on Deep Learning*, 2011.

Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *CVPR*, 2012.

J. Feng, B. Ni, Q. Tian, and S. Yan. Geometric lp-norm feature pooling for image classification. In *CVPR*, 2011.

M. A. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *CVPR*, 2010.

C. M. Bishop. *Neural Network for Pattern Recognition*. Oxford University Press, 1999.

Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. *Neural networks: Tricks of the trade*, pages 546–546, 1998.

R. Collobert and S. Bengio. Links between perceptrons, mlps and svms. In *ICML*, 2004.

A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. A. Ranzato, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. 2012b.

A. Krizhevsky and G. Hinton. Convolutional deep belief networks on cifar-10. Technical report, 2010.

A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *PAMI*, 2008.

A. Coates and A. Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, 2011.

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.