Master's Thesis

# Improving Temporal Coherence of Image Features by Clustering Technique Learned from Moving Images

（動画から学習したクラスタリング手法による画像特徴の時間的なコヒーレンスの改良）

**Sainbayar Sukhbaatar**
(スフバータル　サインバヤル)

**Supervisor:　Professor Kazuyuki Aihara (合原一幸　教授)**

**January 28, 2013**

**Department of Mathematical Informatics**
**Graduate School of Information Science and Technology**
**The University of Tokyo**

# Abstract

Object recognition is difficult because the appearance of an object changes in many different ways. To recognize objects robustly, one needs representations that are constant despite those changes. Such invariant representations can be obtained by features with low sensitivity to various visual transformations.

Spatial pooling is a widely used technique for extracting invariant features from images. When the same feature is extracted from different locations of images, activations from nearby locations can be clustered together (i.e., added together) to create invariance to small spatial shifting. However, the invariance produced by spatial pooling is limited to spatial shifts. In addition, spatial pooling can only be applicable to convolutional features because spatial pooling makes clusters of features by using their spatial topography only.

In this thesis, we propose a novel pooling method, *auto-pooling*, that learns soft clustering of features from image sequences. Auto-pooling is trained to improve the temporal coherence of features, while keeping the information loss by pooling at minimum. Our method does not use spatial information, so it can be used with non-convolutional models too.

Experiments on images extracted from natural videos showed that our method can cluster similar features together. When trained by convolutional features, the auto-pooling outperformed the spatial pooling on an image classification task, even though the auto-pooling does not use the spatial topology of features.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Object recognition is one of the hardest problems in machine learning. Its main difficulty is that the appearance of an object changes over time in complex ways, which makes simple pattern matching methods useless. For robust object recognition, one needs representations that are constant over time, despite the changing appearance of an object. The goal of this thesis is to introduce a novel model that can learn such temporally coherent representations from image sequences.

Learning a temporally coherent representation is the same as learning features that are invariant to various transformations such as transitions and rotations. The concept of invariant features is not new, and dates back to Hubel and Wiesel's seminal work [7], where neurons in a cat's visual cortex are studied. They found cells with simple receptive fields that only responded to specific patterns (primarily to oriented edges) at specific locations. More importantly, however, they also found cells with more complex behavior. Those complex cells responded to specific patterns similar to simple cells, except they were invariant to small shifts and rotations.

Inspired by simple and complex cells, the spatial pooling step is introduced to computer vision architectures along with the convolution step [15]. In the convolution step, the same feature is applied to different locations of an image. Then in the pooling step, responses from a local neighborhood are pooled together (typically with a sum or a max operation) to create invariance to small spatial shifting. The essence of pooling is to make representations more abstract by discarding details that are irrelevant to object identification, such as the exact location of an object. Another merit of the pooling step is huge reduction of the dimension of data representations.

However, spatial pooling has several limitations. First, it only works with convolutional models, where features have a spatial structure. Second, spatial pooling only improves the invariance to spatial shifting. An ideal pooling should make features invariant to all types of transformations, and it should work with features with no spatial structure. Invariance to other types of transformations such as rotation and scaling is equally important in robust

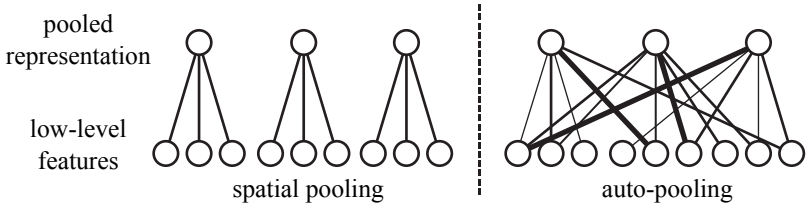


**Figure 1.1.** Tradional spatial pooling and auto-pooling

object recognition. Beside from two-dimensional simple transformations, it is impossible to create invariance to other transformations by convolution.

To make features invariant to complex transformations, it is necessary to learn the pooling step from data. Spatial pooling can be considered as clustering of features, where features close to each other are grouped together. Therefore, our goal is to learn clustering of features from image data. Here, we have to note that feature clustering is not restricted to "hard" clustering. A feature can belong to more than one cluster. Association between a cluster and a feature can be represented by a non-negative real value, which is called *strength*. Such clustering is often referred to as "soft" or "fuzzy" clustering.

In this thesis, we propose a novel pooling method, which we called *auto-pooling*. It learns to perform soft clustering of features through a training with image sequences (see Fig. 1.1). It tries to learn a pooling matrix, a non-negative matrix that represents associations between features and clusters (i.e., pooling regions). Our model achieves invariance by improving the temporal coherence of features, which is the same as making their changes slower. However, we should avoid zero variance because features should be informative about their inputs. This is achieved by minimizing the reconstruction error in the way to autoencoders [21, 23].

There are several advantages in auto-pooling over traditional spatial pooling. First, it produces invariance to all types transformations that present in natural videos. Second, auto-pooling is more biologically plausible model for complex cells because their connections are learned from image sequences rather than being manually defined by arbitrary heuristics. Third, auto-pooling can replace spatial pooling without modifying the low-level feature learning. Finally, it can be used with non-convolutional models because it does not use spatial information. This opens the door to utilize pooling in data types other than images, such as speech and text.

We will show the effectiveness of our model with two types of experiments. In the first experiment, we train an auto-pooling model with non-convolutional features. The goal of this experiment is to see whether similar features are being clustered together. We also defined and measured the invariance score of features before and after pooling. In the second experiment, we compared auto-pooling with traditional spatial pooling on a widely used image classification benchmark.

## 1.2 Outline of the thesis

Chapter 2 presents basics of feature extraction. We introduce sparse autoencoders as a representative of feature learning methods because their structure is similar to our model and we used them in our experiments. Also, convolution and spatial pooling are explained. In Chapter 3, we discuss studies related to our model, such as several variations of spatial pooling. In Chapter 4, we give details of our model. We also introduce a score for measuring invariance of features. Chapter 5 presents the results of our experiments on natural videos and image classification. Chapter 6 have discussions of our pooling method and Chapter 7 concludes this thesis.

# Chapter 2

# Background

## 2.1 Unsupervised image feature learning

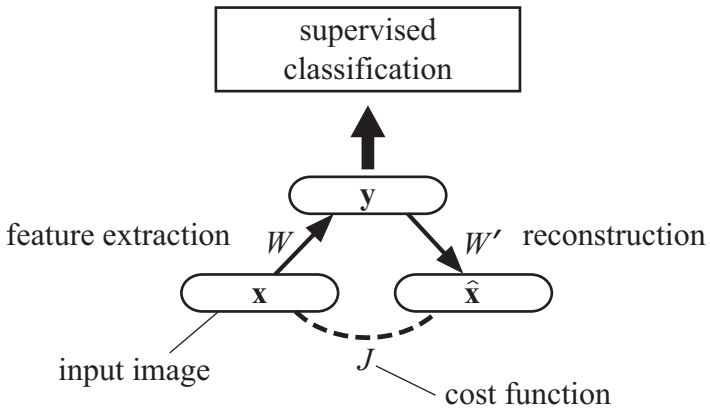


**Figure 2.1.** An autoencoder as a preprocessing step in supervised classification

Image classification is a supervised task, but unsupervised learning models can be helpful as a pre-processing step. Instead of using raw images in classification, it is better to use meaningful features extracted from images such as edges. Unsupervised models can learn such features from unlabeled data. Furthermore, unsupervised learning is much cheaper than supervised learning, because it does not require expensive labeling.

Learning useful features from natural images has been one of the hottest topics in machine learning. Several models have been proposed, such as sparse Restricted Boltzmann Machines [16], sparse autoencoders [21], denoising autoencoders [23] and independent component analysis [9]. Although those models are very different from each other, features learned by them all resemble to each other: oriented short edges at various locations. This is similar to simple cells in the visual cortex.

Here, we will introduce only sparse autoencoders because we used them in our experiments for their simplicity. In addition, it is convenient to present autoencoders here because our pooling method is similar to them. However, our pooling method can be used with any feature learning model.

### 2.1.1 Autoencoders

Antoencoders are an unsupervised model for learning compact representations of data. They can be considered as a special type of traditional two-layer neural networks. The only difference is that autoencoders use the same data for both input and output rather than using separate data in training. This turns a supervised backpropagation algorithm into an unsupervised learning. Then, the goal of learning becomes to accurately reconstruct the input from hidden units as shown in Fig. 2.1.

In an autoencoder, input vector $\mathbf{x}_i$ is encoded by

$$\mathbf{y}_i = f(W\mathbf{x}_i + \mathbf{b}),$$

where $W$ is a encoding matrix and $\mathbf{b}$ is a bias vector for the hidden units. A widely used activation function is

$$f(\mathbf{x}) = (s(x_1), s(x_2), ..., s(x_d))^T,$$

where $d$ is the dimension of vector $\mathbf{x}$ and

$$s(x) = \frac{1}{1 + e^{-x}}.$$

This sigmoid activation function produces quasi-binary codes. Then in the decoding step, the input is reconstructed by

$$\hat{\mathbf{x}}_i = g(W'\mathbf{y}_i + \mathbf{b}'),$$

where $W'$ is a decoding matrix and $\mathbf{b}'$ is a bias vector for visible units. For real-valued data, the identity function

$$g(\mathbf{x}) = \mathbf{x}$$

is used as an activation function for visible units. The training of autoencoders is driven by a squared error cost function of

$$J = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2,$$

which can be minimized by a simple gradient descent algorithm.

### 2.1.2 Sparse Autoencoders

Normal autoencoders usually have fewer hidden units than visible units to avoid a trivial solution, in which each visible unit is represented by a single hidden unit. They are considered as a compression model because they try to learn compact representations of input data. However, it is known that such compression does not produce useful features when trained by natural images.

Sparse autoencoders, on the other hand, have more hidden units than visible units. When trained by small image patches, sparse autoencoders learn Gabor-like edge detectors.

They avoid the trivial solution by enforcing the following sparsity constraint on hidden units

$$\hat{\rho}_k = \rho,$$

where $\rho$ is a sparsity parameter and $\hat{\rho}_k$ is the average activation of $k$-th hidden unit, which is

$$\hat{\rho}_k = \frac{1}{N}\sum_{i=1}^{N} y_i^k.$$

This sparsity constraint is enforced by a cost function

$$J_{\text{sparse}} = \sum_{k=1}^{K} \text{KL}(\rho \parallel \hat{\rho}_k),$$

where Kullback-Leibler (KL) divergence is given by

$$\text{KL}(\rho \parallel \hat{\rho}_k) = \rho \log \frac{\rho}{\hat{\rho}_k} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_k}.$$

It is also practical to use the weight regularization cost

$$J_W = \frac{1}{2}\sum_{m}^{M}\sum_{k}^{K}(W_{km}^2 + {W'}_{mk}^2),$$

which keeps weights from taking too large values. The total cost function for sparse autoencoders is

$$J_{\text{total}} = J + \beta J_{\text{sparse}} + \alpha J_W,$$

where $\alpha$ and $\beta$ are parameters for controlling the weights of sparsity and weight regularization costs.

## 2.2 Convolution and Spatial Pooling

It is actually hard to learn features from images due to long computation time. Even for $32 \times 32$ small color images, the data dimension becomes 3072. As the image size increases, the computation time grows at least quadratically. Convolution is a widely used trick in those conditions to reduce the computation time.

When a feature learning model such as a sparse autoencoder is trained by natural images, the same local feature emerges at different locations. Therefore, it will be sufficient to learn local features at a single location and later duplicate them to all other locations. This is what convolutional models do to reduce the computation time. First, they are trained with small image patches to learn local features. Then, final global features are constructed by applying each local feature to every locations of images (see Fig. 2.2).

However, convolutional feature extraction results in large number of features. For each local feature, there are almost the same number of features as the number of pixels in images. Luckily, features extracted by convolutional models have a topographic structure.

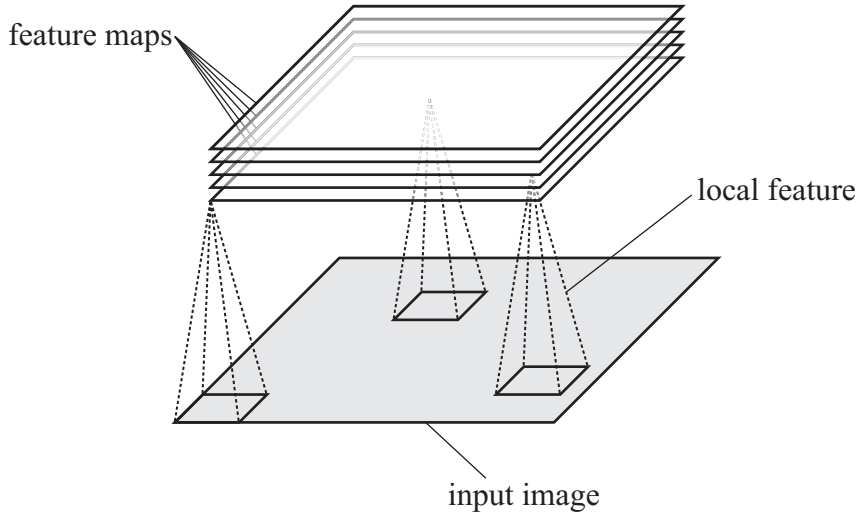


**Figure 2.2.** Convolutional feature extraction

They are organized into two-dimensional maps, each corresponding to a single local feature. This spatial structure can be exploited by pooling, where each map is divided into small squares by a grid. Then, activations in each square are combined by a pooling operation (see Fig. 2.3). Most frequently used pooling operations are sum and max operations (it is still in a debate which one is better).

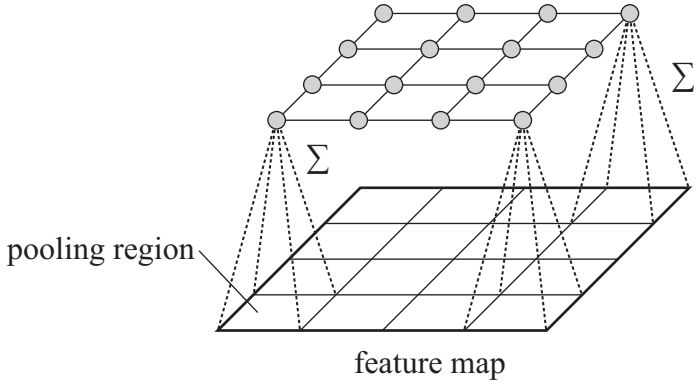


**Figure 2.3.** $4 \times 4$ spatial pooling

Spatial pooling significantly reduces the number of features, which is vital in convolutional models. Furthermore, features become more invariant to small spatial shifts after pooling, because features from a local neighborhood are pooled together. This spatial invariance of features significantly improves the object recognition accuracy.

# Chapter 3

# Related Work

Since the discovery of simple and complex cells, many methods have been proposed to imitate the invariance property of complex cells. Those methods often have two-layer structures, where the lower layer has units corresponding to simple cells, and the upper layer has units corresponding to complex cells. We can divide those methods into two categories based on whether the training of simple cells is dependent on complex cells.

In methods of the first category, the training of simple cells is completely independent from that of complex cells. Neocognitron [5] is one of the first models to introduce spatial pooling, which belongs to this category. Similar to the visual nervous system proposed by Hubel and Wiesel [7], neocognitrons have hierarchical structure of alternating layers of simple and complex cells. Each layer is composed from multiple two-dimensional maps. Simple cells in the same map all have the same receptive fields, only differing in their location. Receptive fields of complex cells, however, are fixed and restricted to a local neighborhood of simple cells.

Since neocognitrons, several similar methods have been proposed. While the structure of complex cells has been kept the same (as the spatial pooling step), training of simple cells have replaced by many different methods. There are several variations of spatial pooling differing in their complexity as shown in Fig. 3.1. The most simple spatial pooling is bag-of-features, a widely used technique in computer vision, in which spatial information is completely discarded. Actually, bag-of-features is equivalent to $1 \times 1$ spatial pooling.

A more complex spatial pooling method is spatial pyramids [14]. Instead of dividing feature maps into pooling regions by a single grid, spatial pyramids employ multiple grids

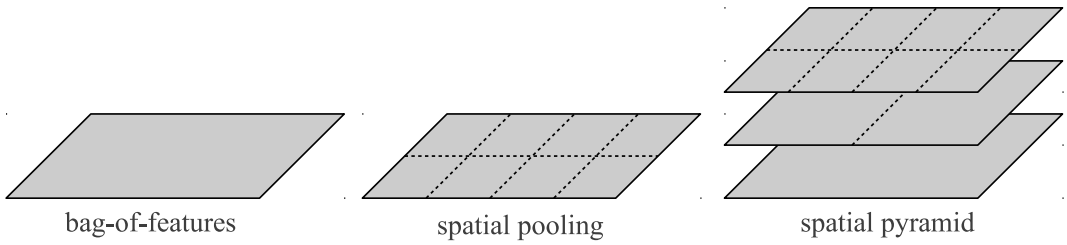


**Figure 3.1.** Different spatial pooling methods

with increasing granularity. This is the same as using multiple spatial pooling modules with different grids in parallel. As a result, it becomes possible to use both global and local features at the same time.

However, these spatial pooling methods all have fixed pooling regions defined by simple grids. Their structures are relatively simple compared to feature learning methods, and independent of training data.

There is a method [10] to choose spatial regions in more adaptive way. In this method, training starts with many pooling region candidates, and only a few of them are used in the final classification. This selection of pooling regions is achieved by supervised learning incorporated into the training of a classifier. Pooling region candidates are populated by random rectangular regions. Although this method learn pooling regions from data, it is still restricted to convolutional models. Also, pooling regions are limited to rectangular shapes. In addition, it is not suited for deep learning, where lower layers are trained in unsupervised way.

Another method that improved spatial pooling is proposed by Coates and Ng in [3], in which local features are clustered by their similarity. A similarity metric between features is defined from their energy correlations. Then, nearby features from the same cluster are pooled together to create rotation invariance. However, the invariance to spatial shifting is still achieved through the same spatial pooling, which restricts this model to convolutional models.

Methods [8, 20, 12] in the second category create invariance by placing features on a two-dimensional topographic map. During training, nearby features are constrained to be similar to each other. Then, invariant representations are achieved by clustering features in a local neighborhood. However, those methods fix clustering manually, which requires clusters to have the same size. Also, we cannot guarantee that the optimal feature clustering can be mapped into two-dimensional space. Moreover, those methods cannot be used with features already learned by another model.

Topographic ICA [8] is a method in the second category that extends Independent Component Analysis (ICA) [11]. ICA is a generative model that tries to represent the input by statistically independent components. When trained with image patches, components of an ICA resembles to Gabor filters. However, there is still dependence left between components learned by an ICA. Topographic ICA tries to remove this residual dependence by adding a layer of complex cells at top of an ICA. In a topographic ICA, simple cells (i.e. components) are organized into a two dimensional map. The main idea is that only nearby simple cells are allowed to have energy correlation between their activation. This topographic constraint produces a map of features where similar features are placed close to each other. Then complex cells simply pools over simple cells at a local neighborhood to create invariance. In contrast with spatial pooling, topographic ICA produces invariance to rotation and scaling as well.

Our pooling method belongs to the first category because low-level feature learning is independent from the pooling step. It is an advantage of auto-pooling, because models for feature learning are well studied, and we can choose one that is best fit to the data. Features can be extracted in a convolutional way like spatial pooling, but it is not required.

# Chapter 4

# Auto-pooling

## 4.1 Introduction

In this chapter, we introduce *auto-pooling*, a pooling method that performs **soft** clustering on a set of features, which can be learned by a separate model. An auto-pooling model can be trained in an unsupervised way to improve the temporal coherence of image features, thus make them more invariant to various transformations. Such abstraction by invariance is important in object recognition, where exact details like location and orientation are often irrelevant.

The goal of auto-pooling is to cluster similar features together, so that a small transformation of an object will not affect its pooled representation. Two features are considered similar if they are traversable by a small transformation such as shifting or rotation. For example, edge detectors with nearby locations and similar orientations are similar to each other. We use image sequences extracted from natural videos as the resource for learning similarity between features. They are rich in various complex transformations and easy to prepare. Moreover, image sequences are available to animals and humans as early as their birth, so it is biologically plausible to utilize them in learning of complex-cell-like invariant features.

Auto-pooling models can be viewed as a one-layer neural network. The visible units represent low-level image features, and the hidden units represent cluster centers. We will also refer clusters as pooling regions. Clustering is parameterized by the weights of a neural network, which are required to be non-negative values to represent the associations between features and clusters. A zero weight means that the feature does not belong to the cluster, and a large weight means the opposite.

Like autoencoders, auto-pooling consists from encoding and decoding modules. Both modules have linear activation functions, and share the same weights. The encoding module performs actual pooling by adding features weighted by the clustering parameters. The decoding module, on the other hand, reconstructs activities of features from its pooled representation, and only used during training.

There are two desirable conditions in good pooling methods.

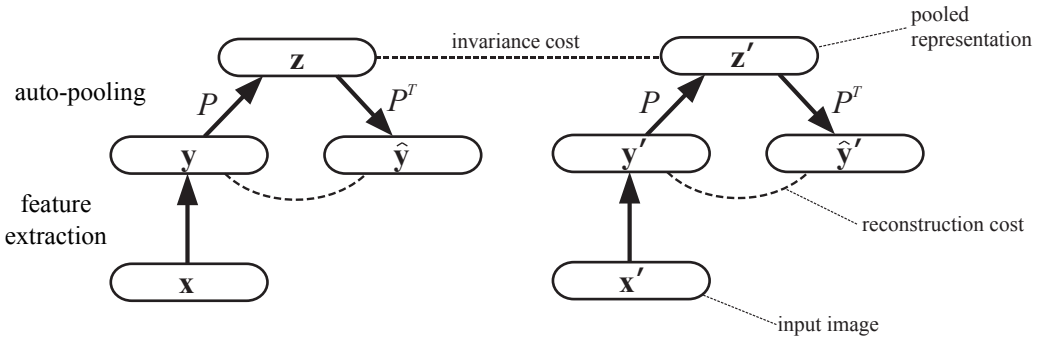


**Figure 4.1.** Auto-pooling

**Invariance property** Pooling should keep the temporal coherence of the original image sequence. In other words, if two images have the same object, which is likely true for images from a continues sequence, then their pooled representations should be the same. Another interpretation is that pooled representations should be slow, if they are extracted from a continues image sequence.

**Entropy property** The information loss die to pooling should be minimum. If pooled representations have too small information from the input images, then classification will be difficult. In other words, the cross entropy between inputs and their pooled representations should be large.

These two properties are in tension to each other. The invariance property can be fulfilled with a simple solution of constant representations with zero variances. However, the entropy property does not favor this trivial solution because variables with zero variances also have zero entropy. The balance of these two properties is the key to invariant abstract features.

Auto-pooling favors the invariance property by minimizing the distance between pooled representations extracted from consecutive frames. The entropy property, however, is not straightforward to obtain. Inspired by autoencoders, we approach this problem by reconstructing feature activities back from its pooled representation. If the information loss due to pooling were small, it would be possible to reconstruct the original input with small error. Auto-pooling minimizes this reconstruction error during training.

Auto-pooling can be considered as a generalization of the traditional sum-pooling method. However, the main advantage of auto-pooling is that it does not use spatial information. Instead, they learn to pool similar features together using image sequences.

## 4.2 Details of implementation

Instead of image sequences, it is convenient to use images pairs taken from consecutive video frames as training data. Such image pairs can be written as

$$X = \{\mathbf{x}_1, \mathbf{x}'_1, \mathbf{x}_2, \mathbf{x}'_2, ..., \mathbf{x}_N, \mathbf{x}'_N\}.$$

If an object is present in image $\mathbf{x}_i$, then it is likely to be present in $\mathbf{x}'_i$ too, because $\mathbf{x}_i$ and $\mathbf{x}'_i$ are taken from frames with a very small time difference (33ms in the case of a video with 30 frames per second). Before the training of an auto-pooling model, image features should be already learned by a some other model. Let us assume that feature extraction is done by

$$\mathbf{y}_i = f(\mathbf{x}_i), \quad \mathbf{y}'_i = f(\mathbf{x}'_i).$$

If features are extracted by an autoencoder or a restricted Boltzmann Machine, the function $f$ will be a sigmoid function. However, it is possible to use auto-pooling with any feature extractor $f$, as long as $\mathbf{y}_i$ and $\mathbf{y}'_i$ are non-negative.

The traditional spatial sum-pooling algorithm can be expressed as

$$\mathbf{z}_i = P\mathbf{y}_i, \quad \mathbf{z}'_i = P\mathbf{y}'_i,$$

where $P$ is a pooling matrix, and its elements are 0 or 1 depending on the topology of feature maps. For example, the spatial pooling shown in Fig. 1.1 can be expressed as

$$P = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

where columns represent features and rows represent pooling regions. Auto-pooling generalizes this traditional pooling by allowing the elements of $P$ to take any non-negative real-values. This turns hard clustering by spatial pooling into more adaptive soft clustering.

Our main goal is to learn pooling parameters $P_{ij}$, without using the explicit spatial information. The training of auto-pooling is driven by two cost functions. The first cost function is

$$J_1 = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{z}_i - \mathbf{z}'_i\|_2^2,$$

which tries to obtain the invariance property by minimizing the distance between pooled representations $\mathbf{z}_i$ and $\mathbf{z}'_i$. However, there is a trivial solution of $P = 0$ if we use only this cost function.

The second cost function corresponds to the entropy property, and encourages pooled representations to be more informative of their inputs. Input $\mathbf{y}_i$ and $\mathbf{y}'_i$ are reconstructed from their pooled representations by

$$\hat{\mathbf{y}}_i = P^T\mathbf{z}_i, \quad \hat{\mathbf{y}}'_i = P^T\mathbf{z}'_i$$

using the same parameters as the encoding module. The reconstruction error is minimized by the cost function of

$$J_2 = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2 + \|\mathbf{y}'_i - \hat{\mathbf{y}}'_i\|_2^2).$$

The final cost function of auto-pooling is

$$J = \lambda J_1 + J_2,$$

where parameter $\lambda \geq 0$ controls the weight of the invariance cost function. Larger $\lambda$ will make features more invariant by discarding more information from the original input.

Auto-pooling is similar to autoencoders, which are used in feature learning. In auto-pooling, the input is reconstructed in the same way as auto-encoders. Also, the reconstruction cost function $J2$ is exactly same to the cost function of autoencoders. However, there are several important differences between them. First, parameters of auto-pooling are restricted to non-negative values. Second, activation functions of auto-pooling are linear and have no biases. Third, auto-pooling has an additional cost function for temporal coherence.

Auto-pooling can be trained by a simple gradient descent algorithm, which iteratively minimizes the cost function using its gradient information. The update rule for each iteration is

$$P \leftarrow P - \gamma \frac{\partial J}{\partial P},$$

where the partial derivation is given by

$$\begin{aligned}\frac{\partial J}{\partial P} = \frac{1}{N}\sum_{i=1}^{N}[&\mathbf{z}_i(\hat{\mathbf{y}}_i - \mathbf{y}_i)^T + \mathbf{z}'_i(\hat{\mathbf{y}}'_i - \mathbf{y}'_i)^T \\ &+ P(\hat{\mathbf{y}}_i - \mathbf{y}_i)\mathbf{y}_i^T + P(\hat{\mathbf{y}}'_i - \mathbf{y}'_i)\mathbf{y}_i'^T \\ &+ \lambda(\mathbf{z}_i - \mathbf{z}'_i)(\mathbf{y}_i - \mathbf{y}'_i)^T].\end{aligned}$$

Here $\gamma \geq 0$ is a learning rate. In practice, it is faster to divide training data to smaller batches, and update parameters for each batch. Also the use of momentum in the gradient descent shortens the training time [22].

## 4.3 Invariance Score

To evaluate our model, we define a score for measuring the invariance of features. A simple metric for measuring invariance of a feature is its change between two subsequent frames, which is

$$G = \frac{1}{T}\sum_{t=1}^{T} \|g(\mathbf{x}_t) - g(\mathbf{x}_{t+1})\|_2.$$

Here, $g(\mathbf{x}) := f(\mathbf{x})$ if we are measuring invariance of raw features, and $g(\mathbf{x}) := Pf(\mathbf{x})$ if we are measuring invariance of pooled representations. If a feature is invariant, then $G$ would be small.

However, we need an additional term to prevent features from cheating to reduce $G$. If a feature simply have the same activation all the time, then $G$ would become zero. An ideal invariant feature should take the same value only if stimuli are from subsequent

frames. If stimuli are chosen randomly, then an invariant feature should take different values because it is likely that the inputs contain different objects. Therefore, the average distance between two random frames

$$H = \frac{1}{T}\sum_{t=1}^{T} \|g(\mathbf{x}_t) - g(\mathbf{x}_{\sigma(t)})\|_2$$

should be large. Here $\sigma$ is a random permutation of $\{1, 2, ..., T\}$. The invariance score is defined by

$$F = \frac{H}{G}, \tag{4.1}$$

which will be large only for invariant features. Later, we will use this score to compare the invariance of features before and after pooling.

# Chapter 5

# Experiments

## 5.1 Pooling of image features

In this experiment, we trained an auto-pooling model with features learned from small image patches. Our goal is to demonstrate that auto-pooling can cluster similar features together, and thus increase their invariance. Invariance score of features is measured before and after pooling. The training consisted from three steps: data preparation, feature learning and pooling.

### Step 1: Data preparation

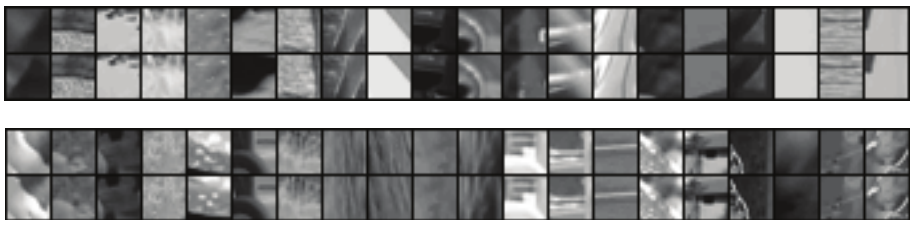

**Figure 5.1.** Image pairs (each column) extracted from natural videos

We prepared small image pairs from natural videos. The videos used in this dataset are obtained from `http://www.vimeo.com` (all videos had Creative Commons license). To improve the diversity of the dataset, we included videos with natural scenes, animals (e.g., frogs, cats, birds, etc.) and artificial objects (e.g., cars, planes, etc.). Total of 44 videos are used in the dataset, and their sizes were 640×340 pixels. Frame rates of the videos were between 24 and 30 frames per second. We prepared image pairs from the videos by the following procedure:

1. Select a random video from the repository
2. Choose two consecutive frames $\mathbf{s}^{(t)}$, $\mathbf{s}^{(t+1)}$ randomly from the selected video.
3. If $\|\mathbf{s}^{(t)} - \mathbf{s}^{(t+1)}\|_1 > \varepsilon_1$, then discard those frames and go back to the previous, because it is likely that those frames are not from a continues sequence. Here $\varepsilon_1$ is a some small number.

4. Extract $32 \times 32$ patches from a random location of the image pair

$$\mathbf{s}^{(t)} \rightarrow \mathbf{x}$$
$$\mathbf{s}^{(t+1)} \rightarrow \mathbf{x}'$$

5. If $\min(\sigma(\mathbf{x}), \sigma(\mathbf{x}')) < \varepsilon_2$ for a small constant $\varepsilon_2$, then go back to step 4 to exclude blank patches such as part of sky. Here $\sigma(\mathbf{x})$ is the standard deviation of the elements of $\mathbf{x}$.

6. If $\sigma(\mathbf{x} - \mathbf{x}') < \varepsilon_3$, then go back to step 4 to skip identical patches.

7. Rescale patches $\mathbf{x}$, $\mathbf{x}'$ to $16 \times 16$ and convert them to gray-scale.

8. Finally, all patches are contrast normalized and whitened, a process known to improve feature learning [2]. For details about the whitening process, see Appendix A.

The final dataset contained 100 thousand pairs of small image patches. Some of the image pairs are shown in Fig. 5.1.

### Step 2: Unsupervised feature learning from image patches

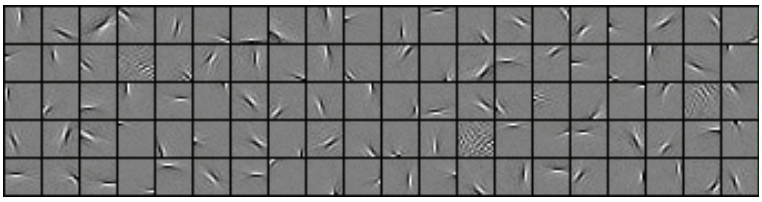

**Figure 5.2.** Features learned from image patches by a sparse autoencoder

Since our pooling method is completely independent of the low-level feature learning, we can use any model for feature learning. Here, we used a sparse autoencoder for its simplicity and ease of training. The image pair dataset is used in the training, but the pairing relationship of patches was ignored in this case.

The sparse autoencoder had 256 visible units, one for each pixel in image patches. The hidden layer had 400 units, so it learned sparse over-complete representations. The sparsity parameter were set to 2% and its weight $\beta$ were set to 5. The training continued for 3000 epochs, with batch size of 1000 and learning rate of 0.1. A subset of learned features is shown in Fig. 5.2. As expected, most of the features were edge detectors, similar to simple cells of the visual cortex.

### Step 3: Clustering of features by auto-pooling

After obtaining image features, we trained a auto-pooling model on top of them. It had 50 hidden units, each corresponding to a cluster of features. The training continued for 50 epochs with batch size of 1000 and learning rate of 0.003. When the training finished, pooling matrix $\mathbf{P}$ became a sparse matrix with few non-zero elements. A large $P_{ij}$ means that $j$-th feature belongs to $i$-th cluster. However, a feature can belong to more than one clusters because the only restriction on pooling matrix $\mathbf{P}$ is non-negativity.

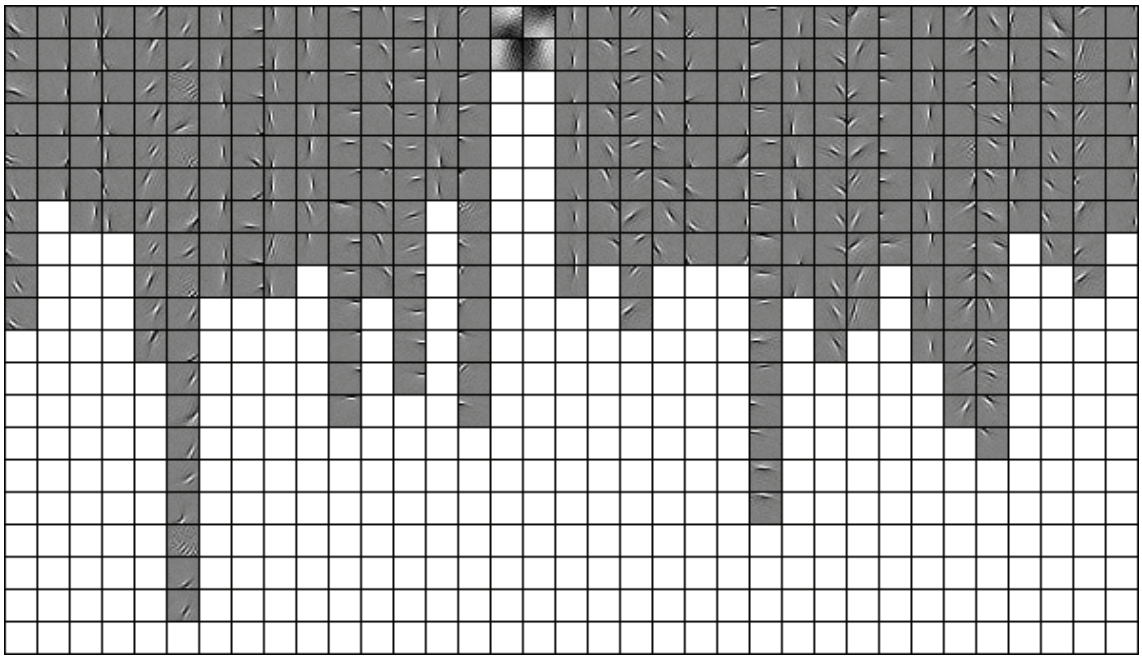

**Figure 5.3.** Clusters (columns) of features learned from the image pair dataset

Since auto-pooling perform soft clustering, it is hard to visualize the clustering result. For simplicity, however, we used a small threshold $\varepsilon$ to show some learned clusters in Fig. 5.3. For $i$-th cluster, we showed features with $P_{ji} > \varepsilon$ in a single column. It is evident that similar features are clustered together. Also, one can see that the size of cluster varies greatly depending on the nature of its features.

**Figure 5.4.** Diversity of edge detectors in clusters

To display clusters more clearly, we have shown some clusters of edge detectors in more detail in Fig. 5.4, in which edge detectors are replaced by corresponding thin lines, so we can see them together. This clearly shows the diversity of edge detectors inside each cluster. It is important to note that there is variance in orientations beside from locations.

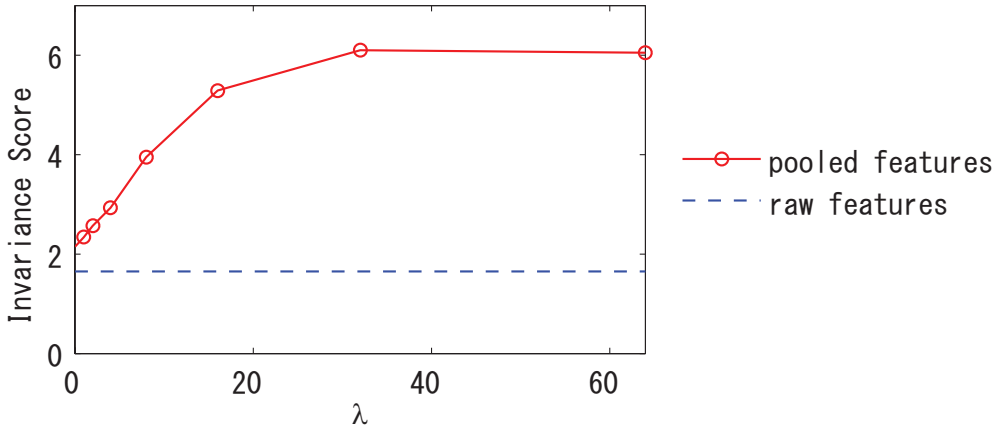


**Figure 5.5.** Invariance scores of features before and after the pooling

Next, we measured the effect of clustering on image features using our invariance score. Fig. 5.5 shows the scores of features before and after clustering, measured at various values of parameter $\lambda$, which controls the weight of the invariance cost function. The invariance score is significantly improved after the clustering, especially for large $\lambda$. It is not surprising because larger $\lambda$ puts more importance on the invariance cost, which makes pooled representations less likely to change their activations between consecutive frames. As a result, term $G$ in Eq. 4.1 becomes small.

At the same time, however, the increase of $\lambda$ diminishes the role of the reconstruction cost, which was keeping pooled representations more informative about their inputs. Thus, too large $\lambda$ makes pooled representations over-invariant, having constant activations all the time. This reduces $H$ in Eq. 4.1. We can see this side effect of large $\lambda$ from Fig. 5.5, where the invariance score stopped its growth at large $\lambda$.

## 5.2 Image Classification

In this experiment, we aimed to show the effectiveness of our pooling method by applying it to an image classification task. There are four steps in the training. The first three steps are the same as the previous experiment, except that we used a convolutional model to compare our method with the traditional spatial pooling method. The fourth step is training of a classifier.

### Step 1: Data preparation

We used two types of datasets for this experiment: an image pair dataset and a labeled image dataset. The labeled image dataset is for unsupervised feature learning and supervised classification. The image pair dataset is for only the training of auto-pooling.

We used CIFAR10 dataset [13] in the image classification, which contains labeled images from ten categories (some samples are shown in Fig. 5.6). The dataset was divided

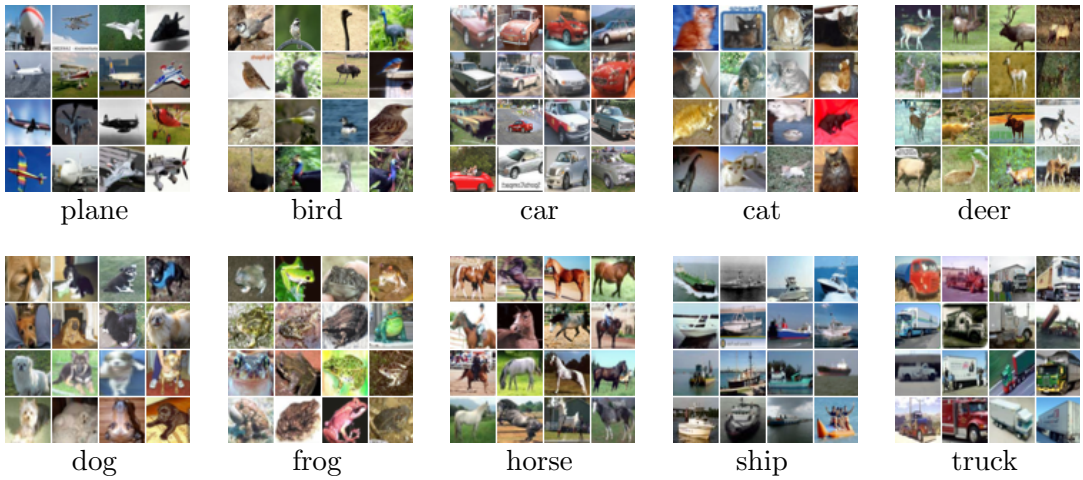


**Figure 5.6.** Sample images from the ten categories of CIFAR10 dataset

into two parts: a training data containing 50 thousand images, and a test data containing 10 thousand images, each evenly distributed among the ten categories. All images were 32×32 pixels in size, and had three-color channels, with the total dimension of 3072.

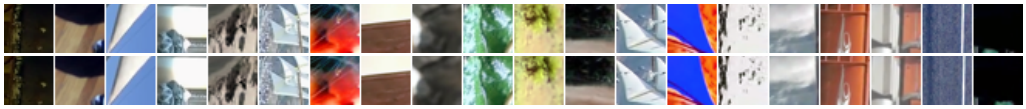

**Figure 5.7.** Samples from the image pair dataset prepared from natural videos

For the training of auto-pooling, we used an image pair dataset prepared in the same way as the previous experiment. The only difference was that the patches were 32×32 color images as the same as images in CIFAR10 dataset. Some samples from the image pair dataset are shown in Fig. 5.7. It is evident that there are qualitative differences between images from CIFAR10 and image pairs from natural videos. While images from CIFAR10 are relatively complex and always have an object at their centers, images from videos often show only a part of an object, and have much less details.

### Step 2: Unsupervised feature learning

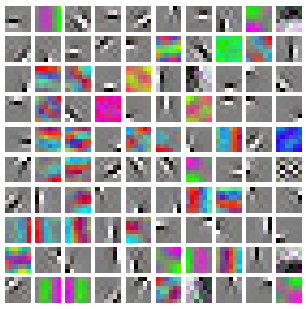

**Figure 5.8.** Features learned from small image patches

In the feature learning step, we used a convolutional model. We trained a sparse autoencoder with 100 hidden units on 6×6 small patches extracted from images in CIFAR10 dataset. Learned image features are shown in Fig. 5.8. Convolutional feature extraction was done with one pixel stride. That means we extracted 100 features from every 6×6 patches of 32×32 images. This resulted in 100 feature maps of $27 \times 27$.

### Step 3: Clustering of features by auto-pooling

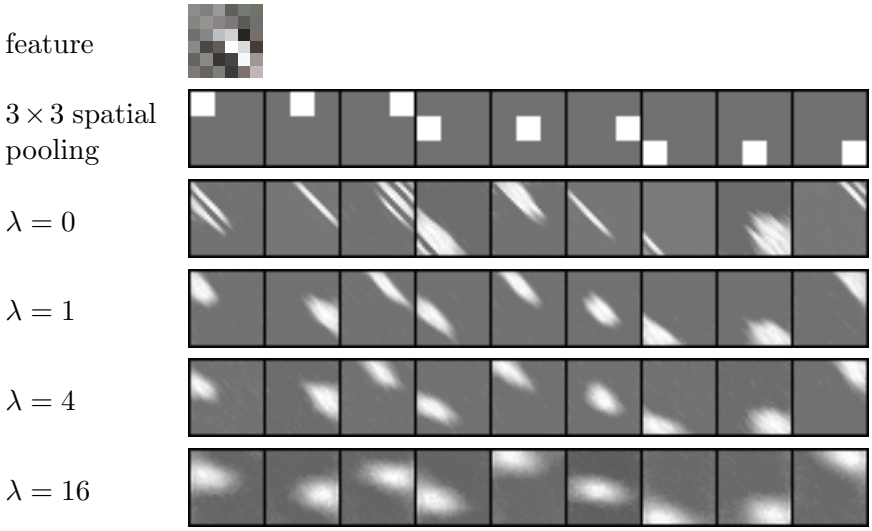


**Figure 5.9.** Shape of pooling regions

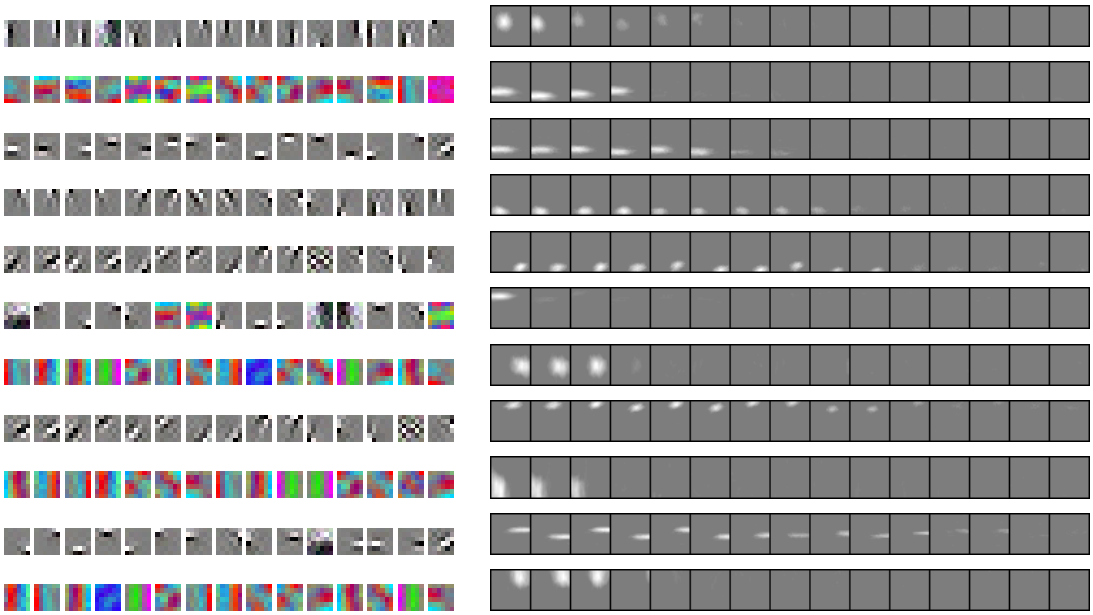

**Figure 5.10.** Pooling regions learned by auto-pooling. The right side shows pooled regions on feature maps, and the left side shows corresponding features

We trained an auto-pooling model with features extracted in a convolutional way from

the image pair dataset. First, we used only one feature map to show the effect of $\lambda$ in pooling regions. Nine different pooling regions learned at various values of $\lambda$ are shown in Fig. 5.9. $P_{ij} = 0$ is shown in gray and large $P_{ij}$ is shown in white. When there is no temporal coherence learning (i.e., when $\lambda = 0$), the learned pooling regions (i.e., white areas in Fig. 5.9) were various in size and often divided into separate parts, which is obviously not good pooling. However, when auto-pooling tries to improve the temporal coherence of features (i.e., when $\lambda > 0$), divided regions are merged into a single continues region. As $\lambda$ increased, the pooling regions became more smoother, larger and likely to overlap with each other. Compared to $3 \times 3$ spatial pooling, the pooling regions learned by auto-pooling clearly show dependency on a feature (i.e., by having the same orientation).

When all 100 maps are used, it produced very large (exceeding 100 gigabytes) training data for auto-pooling. Luckily, the training took only a few hours because we implemented our algorithm on a graphic card (Tesla K20c) using CUDAMat library [18]. Some of the learned feature clusters are visualized in Fig. 5.10. For each cluster, we showed 15 feature maps with the largest pooling area (i.e. $\max_k(\sum_{j \in S_k} P_{ij})$, where $S_k$ is the set of features in $k$-th feature map). Local features corresponding to the feature maps are also shown in Fig. 5.10.

Unlike spatial pooling, each cluster learned by auto-pooling extended to multiple feature maps. Pooling regions (i.e., white areas in Fig. 5.10) of those maps usually have the same continues spatial distribution, which will create spatial invariance in the same way as spatial pooling. If we observe carefully those pooling regions, however, we can see the small variance in their locations. This location variance is inversely related to the location variance of corresponding local features. For example, if there is a edge detector in the lower part of a $6 \times 6$ local feature, corresponding pooling regions will have upper position in $27 \times 27$ feature maps.

Beside from pooling by spatial areas, auto-pooling also succeeded in clustering similar local features. In some clusters, edge detectors of similar orientations are grouped together. This will make pooled representations invariant to small rotations, which is a clear advantage over traditional spatial pooling. In addition, clustering of local features only differing in their locations will reduce the redundancy created by convolutional feature extraction.

### Step 4: Classification

We compared our pooling method with traditional spatial pooling on a classification task, in which a supervised classifier is trained by pooled representations of labeled images. For auto-pooling, we varied the number of clusters from 400 to 2500. For spatial pooling, we can only change the grid size. However, it is possible to use spatial pyramid to produce better results. We denote a spatial pyramid that used $2 \times 2$ and $3 \times 3$ grids by $2 \times 2 + 3 \times 3$.

In classification, we trained a linear SVM with pooled representations. The results are shown in Table 5.1. We trained the classifier with two training data: a full data with 5000 examples per class, and a smaller one with 1000 examples per class. Since

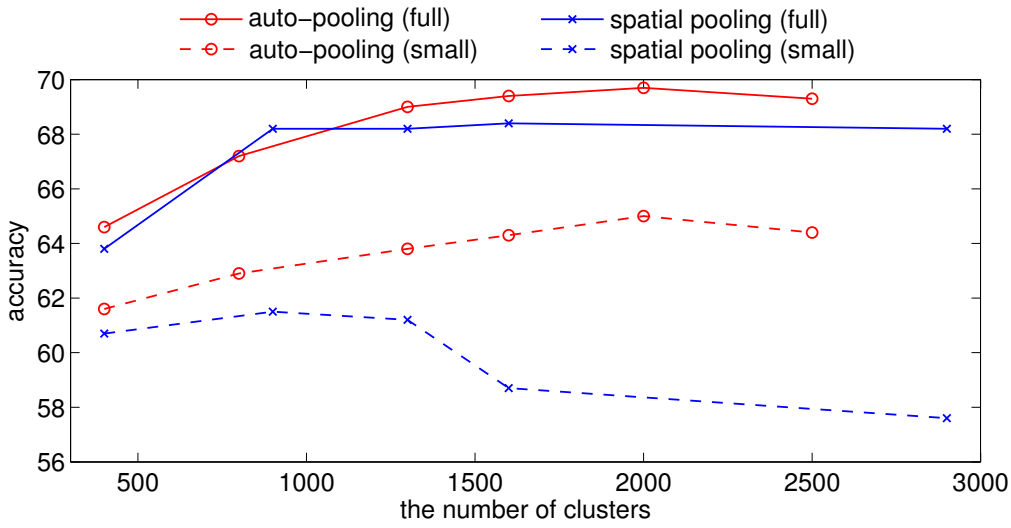


**Figure 5.11.** Classification accuracy

the number of features is an important factor in classification, we plotted the accuracy of the two pooling methods against the number of clusters in Fig. 5.11. Auto-pooling outperformed traditional spatial pooling for the most time. Especially for small training data, the difference between the two pooling methods was substantial. This indicates that auto-pooling is better at generalization, which is the main goal of invariant features. The spatial pooling, on other hand, shows the sign of over-fitting when its pooling regions are increased.

**Table 5.1.** Classification accuracy on CIFAR10 (AP=auto-pooling, SP=spatial pooling)

| **Pooling methods** | **Accuracy (full)** | **Accuracy (small)** |
|---|---|---|
| AP (400 clusters) | 64.6% | 61.6% |
| AP (800 clusters) | 67.2% | 62.9% |
| AP (1300 clusters) | 69.0% | 63.8% |
| AP (1600 clusters) | 69.4% | 64.3% |
| AP (2000 clusters) | 69.7% | 65.0% |
| AP (2500 clusters) | 69.3% | 63.5% |
| SP ($2 \times 2$) | 63.8% | 60.7% |
| SP ($3 \times 3$) | 68.2% | 61.5% |
| SP ($2 \times 2 + 3 \times 3$) | 68.2% | 61.2% |
| SP ($4 \times 4$) | 68.4% | 58.7% |
| SP ($2 \times 2 + 3 \times 3 + 4 \times 4$) | 68.2% | 57.6% |

# Chapter 6

# Discussions

## 6.1 Auto-pooling and complex cells

Computers already overtook humans on the processing of concrete information such as manipulations of numbers or transformations of images at the pixel level. However, they always struggle when it comes to incomplete information and abstract concepts. Humans and animals are superior at making abstraction from concrete information. For example, they can recognize many different faces as a single concept, and this abstraction is often more important than exact details.

To build computer vision that can recognize objects robustly, many have studied the brain to understand how it processes visual stimuli. Although the brain was too complex to understand, they discovered some of its important properties. The visual cortex had a hierarchical structure [4, 19] consisting from more than ten layers. At the bottom of that hierarchy lied the primary visual area (V1), the best understood part of the visual cortex. The information processing in V1 starts with simple cells followed by complex cells, which shows small invariance to spatial shifts and rotations [7].

There are several methods that imitated the invariant behavior of complex cells, which we introduced in Chapter 2. However, they cannot give an explanation to the learning of complex cells because they fixes synapses of complex cells. Auto-pooling, on the other hand, learns its parameters from image sequences. It showed that complex-cell-like invariance can be learned by maximizing the temporal coherence of features and their cross entropy with input data.

## 6.2 Auto-pooling and deep learning

In recent years, deep learning models with deep hierarchical structures similar to the visual cortex attracted great attentions for their success in object recognition [6, 17]. In most cases, deep learning models consist of alternating layers of feature extraction and pooling. It is shown that cells at higher layers can learn to respond to an abstract concept such as cats or human bodies, even though the model is trained in a completely unsupervised way [1].

Although spatial pooling is widely used in convolutional deep learning models, there is a problem when it is used in higher layers. The spatial structure in convolutional features decreases with the number of layers. Every time features are pooled together, the size of feature maps gets smaller and the number of maps increases. Therefore, it might not be optimal to use spatial pooling at higher layers. We provided an alternative pooling method that does not rely on the spatial information. Even at the lowest level where spatial pooling is more suited, auto-pooling produced better results.

# Chapter 7

# Conclusion

We proposed a novel pooling method that can generalize traditional spatial pooling to transformations other than spatial shifting. Auto-pooling tries to make features more temporally coherent, having slow changing activation when presented with a continuous image sequence. The information loss due to pooling is kept minimum using the same cost function as autoencoders. The main advantage of our method is that it **learns** to cluster features, rather than relying on manual heuristic spatial division, in an unsupervised manner.

When trained on image features learned by a sparse autoencoder, auto-pooling successfully clustered similar features together. Most of the clusters consisted of edge detectors with nearby locations and similar orientations. In addition, auto-pooling significantly improved the invariance score of features. We showed the effectiveness of our method by comparing it with traditional spatial pooling on a real-world image classification task. We trained a classifier by pooled representations produced from convolutional features. Our pooling method gave better classification results, even though spatial pooling had the advantage of using spatial information of features.

## 7.1 Future Work

In our experiments, the advantage of auto-pooling over spatial pooling was mainly restricted to learning of rotation invariance. This is because auto-pooling is applied to low-level features, which were mostly edge detectors with the size. Therefore, the only possible variance besides spatial shifting was rotation. We believe that if we use auto-pooling instead of spatial pooling in deep architectures, we can create invariance to more complex transformations such as three-dimensional rotations and distortions. As mentioned before, auto-pooling can cluster features without using explicit spatial information. This makes it possible to utilize pooling in data without spatial structure, such as speech and text.

# Acknowledgments

First of all, I would like to thank my adviser Professor Kazuyuki Aihara for accepting me in his group and providing me with all resources necessary for my master's study. I am also grateful for Project Associate Professor Takaki Makino, who has been giving me valuable instructions throughout my study. I would like to express my appreciation to all members of Aihara-Suzuki-Kohno-Kobayashi Laboratories, especially to Taichi Kiwaki for his helpful discussions and valuable technical supports. Finally, I have to note that my thesis was impossible without the financial support from the Japanese Government represented by MEXT.

# Bibliography

[1] A. Coates, A. Karpathy, and A. Ng. Emergence of object-selective features in unsupervised feature learning. In *Advances in Neural Information Processing Systems 25*, pages 2690–2698. 2012.

[2] A. Coates, H. Lee, and A. Ng. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[3] A. Coates and A. Ng. Selecting receptive fields in deep networks. In *Advances in Neural Information Processing Systems 24*, pages 2528–2536. 2011.

[4] D.J. Felleman and D.C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1(1):1–47, 1991.

[5] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[6] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[7] D. Hubel and T. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106, 1962.

[8] A. Hyvärinen, P. Hoyer, and M. Inki. Topographic independent component analysis. *Neural Computation*, 13(7):1527–1558, 2001.

[9] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4):411–430, 2000.

[10] Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *Computer Vision and Pattern Recognition, 2012 IEEE Conference on*, pages 3370–3377. IEEE, 2012.

[11] C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991.

[12] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. Le-Cun. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009 IEEE Conference on*, pages 1605–1612. IEEE, 2009.

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

[14] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Conference on*, pages 2169–2178. IEEE, 2006.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems 20*, pages 873–880. MIT Press, 2008.

[17] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[18] V. Mnih. CUDAMat: a CUDA-based matrix class for Python. Technical report, Department of Computer Science, University of Toronto, 2009.

[19] D. Mumford. On the computational architecture of the neocortex. *Biological Cybernetics*, 66(3):241–251, 1992.

[20] S. Osindero, M. Welling, and G. Hinton. Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–414, 2006.

[21] M. Ranzato, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19*, pages 1137–1144. MIT Press, 2007.

[22] D. Rumelhart, G. Hintont, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[23] P. Vincent, H. Larochelle, Y. Bengio, and P.A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103. ACM, 2008.

# Appendix A

# Whitening

Whitening is a widely used pre-processing step in image processing. It removes simple dependencies among image pixels. Nearby pixels in an image are likely to have the same value. After whitening, however, it becomes impossible to predict the pixel's value from its neighbors. Samples of whitened images are shown in Fig. A.1. Whitening is useful because it allows following learning models to focus on more interesting higher order dependencies.

In a strict sense, whitening is a decorrelation method that transforms a set of variables into a new set variables by a simple matrix multiplication. Resulting new variables will be uncorrelated, and their covariance matrix will equal to an identity matrix. The process is named "whitening" because whitened data can be considered as a white noise.

Lets assume that input data has zero mean, and represented by a matrix $X$, where columns correspond to data samples and rows correspond to variables. Then, whitened data $Y$ is will be calculated by

$$Y = WX,$$

where $W$ is a whitening matrix.

Whitening matrix $W$ can be found by a simple procedure. Since any rotation of $W$ will be also a solution, we restrict $W$ by

$$W = W^T.$$

From the definition of whitening, $Y$ should satisfy

$$\begin{aligned} YY^T &= I \\ WXX^TW^T &= I \\ W^TWXX^TW^T &= W^T \\ W^2XX^TW^T &= W^T \\ W^2XX^T &= I \\ W^2 &= (XX^T)^{-1} \\ W &= (XX^T)^{-\frac{1}{2}}. \end{aligned}$$

**Figure A.1.** Sample images from CIFAR10 dataset before and after whitening

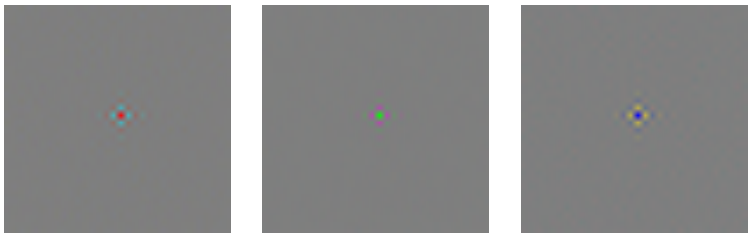


**Figure A.2.** Whitening filters

If we apply singular-value-decomposition to $XX^T$, we will have

$$XX^T = VDV^T,$$

where $V$ is a some orthogonal matrix and $D$ is a diagonal matrix. By using this equation, we can find $W$ by

$$\begin{aligned} W &= (XX^T)^{-\frac{1}{2}} \\ &= (VDV^T)^{-\frac{1}{2}} \\ &= (VD^{-1}V^T)^{\frac{1}{2}} \\ &= VD^{-\frac{1}{2}}V^T. \end{aligned}$$

Since $D$ is a diagonal matrix, it is easy to calculate $D^{-\frac{1}{2}}$.

Once we have whitening matrix $W$, we can whiten data by a simple matrix multiplication. The rows of $W$ can be viewed as filters that are applied to images. However, they usually have the same spatial structure, only differing in locations. Sample whitening filters from each color channel are shown in Fig. A.2.