Reading Notes: Higher-order Integration of Hierarchical Convolutional Activations for Fine-grained Visual Categorization

theppsh

November 20, 2018

1 Introduction

This paper [1] incorporates the *kernel representations* into Convolutional Neural Networks (CNN) for the fine-grained visual categorization (FGVC). I think this trick can be taken into some object detectors by modifying something... And it might lead to higher performance, so I step into this article for more consideration.

In general, taking the higher-oder statistics into CNN by kernel trick can help find the discriminative information between two categories, and benifits the final object classification or detection, and it seems to be fashion that incorporating this traditional method into deep learning recently. Thus, the authors leverage serveral sets of convolutional filters to get an approximation of r-order polynomial kernels.

2 Matching kernel and polynomial predictor

Assume the output from some specific convolutional layers be tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{K\times M\times N}$, and tensor $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}_p\}_{p\in\Omega}$, where $\boldsymbol{x}_p \in \mathbb{R}^K$ represents the descriptor at a particular position p over the set Ω of valid spatial locations with $|\Omega| = M \times N$.

2.1 Review kernel trick

Kernel tricks are widely used in the conventional machine learning algorithms, for instance, the *kernel PCA*, *kernel fisher analysis*, and *kernel SVM*, etc. The intuitive thought of kernelization is to map the original low-dimensional data to a high-dimensional or infinite-dimensional space by a function $\phi : \boldsymbol{x} \in \mathbb{R}^{low} \rightarrow \boldsymbol{y} \in \mathbb{R}^{high}, low \lneq high$, thus the non-linear-discriminative data can be converted to be linearly separable. The most fascinating is most methods leverage *kernel trick* to get the pair-wise similarity graph of the original data rather than calculate the high-dimensional data directly, so it's effective and efficient.

Consider the matching scheme $\mathcal{K}$ for activation sets $\boldsymbol{\mathcal{X}}$ and $\bar{\boldsymbol{\mathcal{X}}}$ from two images, in which the set similarity is measured by aggregating all the pairwise similarities among the local descriptors:

$$\mathcal{K}(\boldsymbol{\mathcal{X}}, \bar{\boldsymbol{\mathcal{X}}}) = g(\{k(\boldsymbol{x}_p, \boldsymbol{x}_q)\}_{p,q\in\Omega}) = \psi(\boldsymbol{\mathcal{X}})^T \psi(\bar{\boldsymbol{\mathcal{X}}}), \tag{1}$$

where $k(\cdot)$ is some kernel function to obtain the similarity between $\boldsymbol{x}_p$ and $\boldsymbol{x}_q$, $g(\cdot)$ is a global sum-pooling function in this paper, $\psi(\boldsymbol{\mathcal{X}})$ and $\psi(\bar{\boldsymbol{\mathcal{X}}})$ are the vector representations for the two sets individually. It is worth noting that the construction of $\psi(\cdot)$ presented above is decomposed into two steps in CNNs: feature mapping and feature aggregating. The mapping step maps each local descriptor $\boldsymbol{x} \in \mathbb{R}^K$ into $\phi(\boldsymbol{x}) \in \mathbb{R}^D$. The feature aggregating produces an image-level representations $\psi(\boldsymbol{\mathcal{X}}) \in \mathbb{R}^D$ by applying global sum-pooling function on the sets $\{\phi(\boldsymbol{x}_p)\}_{p\in\Omega}$ after the mapping step.

Note that Eqn.(1) can be rewritten as:

$$\begin{aligned}
\mathcal{K}(\boldsymbol{\mathcal{X}}, \bar{\boldsymbol{\mathcal{X}}}) &= (\sum_p \phi(\boldsymbol{x}_p))^T (\sum_q \phi(\boldsymbol{x}_q)) \\
&= \sum_p (\phi(\boldsymbol{x}_p)^T \sum_q \phi(\boldsymbol{x}_q)) \\
&= \sum_p \sum_q \phi(\boldsymbol{x}_p)^T \phi(\boldsymbol{x}_q)
\end{aligned} \tag{2}$$

Since we use the global sum-pooling function as $g(\cdot)$, thus $\psi(\boldsymbol{\mathcal{X}})$ satisfies:

$$\psi(\boldsymbol{\mathcal{X}}) = \sum_p \phi(\boldsymbol{x}_p). \tag{3}$$

The term $\phi(\boldsymbol{x}_p)^T \sum_q \phi(\boldsymbol{x}_q)$ in Eqn.(2) can be interpreted as the linear predictor of $\phi(\boldsymbol{x}_p)$, i.e. we regard $\sum_q \phi(\boldsymbol{x}_q)$ as the weights. Inspired by the above linear predictor, we define a function $f(\cdot)$ to project $\phi(\boldsymbol{x}_p) \in \mathbb{R}^D$ to $\mathbb{R}$:

$$f(\boldsymbol{x}_p) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}_p) \rangle, \tag{4}$$

where $\langle \cdot, \cdot \rangle$ is the dot product. To capture the complex and high-order information among parts, use the following polynomial predictor:

$$\begin{aligned}
f(\boldsymbol{x}_p) &= \sum_{k=1}^{K} w_k x_p^k + \sum_{r=2}^{R} \sum_{k_1,\ldots,k_r} \mathcal{W}^r_{k_1,\ldots,k_r} (\prod_{s=1}^{r} x_p^{k_s}) \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \langle \boldsymbol{\mathcal{W}}^r, \otimes_r \boldsymbol{x}_p \rangle \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \sum_{k_1,\ldots,k_r} \prod_{s=1}^{r} \sqrt[r]{\mathcal{W}^r_{k_1,\ldots,k_r}} x_p^{k_s} \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \prod^{r} \langle \sqrt[r]{\mathcal{W}^r_{k_1,\ldots,k_r}} \cdot \mathbf{1}, \boldsymbol{x}_p \rangle
\end{aligned} \tag{5}$$

where $\boldsymbol{w}$ is the paramter of 1-order predictor, $\boldsymbol{\mathcal{W}}^r \in \mathbb{R}^{K^r}$ is r-order tensor which contains the weights of degree-r variable combination in $\boldsymbol{x}$, $\otimes_r \boldsymbol{x} = \underbrace{\boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x}}_{r}$,

and $\otimes$ denotes the *kronecker product*. $\mathbf{1} \in \mathbb{R}^K$ is a vector whose entries equal to 1. Note that function $f(\cdot)$ is just a mapping for $\boldsymbol{x}$, it shares some ideas like kernel function in Eqn.(1), but actually they are totally different. Obviously, take an example of the equation:

$$\sum_p \sum_q w_p x_p w_q x_q = \langle \boldsymbol{w}, \boldsymbol{x} \rangle^2 = \prod^2 \langle \boldsymbol{w}, \boldsymbol{x} \rangle$$

2.2 Tensor learning

The r-order tensor $\boldsymbol{\mathcal{W}}^r \in \mathbb{R}^{K^r}$ in Eqn.(5) can be approximated by rank-one tensor decomposition: Assume r vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{K_1} \in \mathbb{R}^{K_r}$, and their outer product is $K_1 \times \cdots, \times K_r$ rank-one tensor which satisfies $(\boldsymbol{u}_1 \otimes, \cdots, \otimes \boldsymbol{u}_r)_{k_1,\cdots,k_r} = (\boldsymbol{u}_1)_{k_1} \cdots (\boldsymbol{u}_r)_{k_r}$. The rank-one decomposition for such tensor $\boldsymbol{\mathcal{W}}$ is defined as $\boldsymbol{\mathcal{W}} = \sum_{d=1}^{D} \alpha^d \boldsymbol{u}_1^d \otimes \cdots \otimes \boldsymbol{u}_r^d$, where α^d is the weight for d-th rank-one tensor, D is the rank of tensor. We then apply the rank-one approximation for each r-order tensor $\boldsymbol{\mathcal{W}}^r$ and present the following alternative form of polynomial predictor in Eqn.(5):

$$\begin{aligned}
f(\boldsymbol{x}_p) &= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \langle \sum_{d=1}^{D^r} \alpha^{r,d} \boldsymbol{u}_1^{r,d} \otimes \cdots \otimes \boldsymbol{u}_r^{r,d}, \otimes_r \boldsymbol{x}_p \rangle \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \sum_{d=1}^{D^r} \alpha^{r,d} \langle \boldsymbol{u}_1^{r,d} \otimes \cdots \otimes \boldsymbol{u}_r^{r,d}, \otimes_r \boldsymbol{x}_p \rangle \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \sum_{d=1}^{D^r} \alpha^{r,d} \sum_{k_1,\cdots,k_r} \prod_{s}^{r} u_{s,k_s}^{r,d} x_p^{k_s} \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \sum_{d=1}^{D^r} \alpha^{r,d} \prod_{s}^{r} \langle \boldsymbol{u}_s^{r,d}, \boldsymbol{x}_p \rangle \\
&= \langle \boldsymbol{w}, \boldsymbol{x}_p \rangle + \sum_{r=2}^{R} \langle \boldsymbol{\alpha}^r, \boldsymbol{z}_p^r \rangle
\end{aligned} \tag{6}$$

where $\boldsymbol{u}_1^{r,d}, \cdots, \boldsymbol{u}_r^{r,d} \in \mathbb{R}^K$ are parameters can be learned end-to-end. D^r represents the minimal rank of the r-order tensor $\boldsymbol{\mathcal{W}}^r$, or the hyper-paramter corresponding to the output channels of a particular CNN. $\boldsymbol{\alpha}^r = [\alpha^{r,1}, \cdots, \alpha^{r,D^r}]^T$ is the associated weight vector of all D^r rand-one tensors. Vector $\boldsymbol{z}_p^r \in \mathbb{R}^{D^r}$ whose d-th element satisfies:

$$(\boldsymbol{z}_p^r)_d = \prod_{s=1}^{r} \langle \boldsymbol{u}_s^{r,d}, \boldsymbol{x}_p \rangle \tag{7}$$

which characterizes the degree-r variable interactions under a single rank-one tensor basis.

That paper costs a lot of time to describe the final function $f(\boldsymbol{x}_p)$ in Eqn.(6), and I think they will implement $f(*)$ in CNN fashion, but it seems they just use $\boldsymbol{z}_p^r$ incorporated in the final models.

The implementation embedded in CNN can be described as follows: firstly, for each s, we can deploy $\{\boldsymbol{u}_s^{r,d}\}_{d=1,\cdots,D^r}$ as a set of D^r 1×1 convolutional

filters to generate the set of feature maps $\boldsymbol{\mathcal{Z}}_s^r$ of dimension $D^r \times M \times N$. To get the high-order information, we apply element-wise multiplication denoted as $\odot$, and get the output $\boldsymbol{\mathcal{Z}}^r \in \mathbb{R}^{D^r \times M \times N}$ with high-order statistics information as follows:

$$\boldsymbol{\mathcal{Z}}^r = \boldsymbol{\mathcal{Z}}_1^r \odot \cdots \odot \boldsymbol{\mathcal{Z}}_r^r, \tag{8}$$

then sum-pooling function $g(\cdot)$ is used to generate the final output $\boldsymbol{y}^r \in \mathbb{R}^{D^r}$ by sum-up all the local descriptors in $\boldsymbol{\mathcal{Z}}^r$:

$$\boldsymbol{y}^r = g(\boldsymbol{\mathcal{Z}}^r) = \sum_{p \in \Omega} \boldsymbol{z}_p^r, \tag{9}$$

referring to Eqn.(9), the derivatives for $\boldsymbol{x}_p$ and each degree-r convolutional filter $\boldsymbol{u}_s^{r,d}$ in back propagation process can be achieved by:

$$\frac{\partial \ell}{\partial \boldsymbol{z}_p^r} = \frac{\partial \ell}{\partial \boldsymbol{y}^r} \tag{10}$$

from Eqn.(7) and Eqn.(8), we denote $\boldsymbol{z}_p^r, p \in \Omega$ as:

$$\boldsymbol{z}_p^r = \boldsymbol{z}_{p,1}^r \odot \cdots \odot \boldsymbol{z}_{p,r}^r, \tag{11}$$

where we define $\boldsymbol{z}_{p,s}^r \in \mathbb{R}^{D^r}$ whose d-th entry is:

$$(\boldsymbol{z}_{p,s}^r)_d = \langle \boldsymbol{u}_s^{r,d}, \boldsymbol{x}_p \rangle, \tag{12}$$

notice the symbol $\boldsymbol{z}_{p,d}^r$ is different from term $(\boldsymbol{z}_p^r)_d$ in Eqn.(7). Thus the derivative of loss ℓ respect to $\boldsymbol{z}_{p,s}^r$ is:

$$\frac{\partial \ell}{\partial \boldsymbol{z}_{p,s}^r} = \frac{\partial \ell}{\partial \boldsymbol{z}_p^r} \odot \frac{\partial \boldsymbol{z}_p^r}{\partial \boldsymbol{z}_{p,s}^r} = \frac{\partial \ell}{\partial \boldsymbol{z}_p^r} \odot \boldsymbol{z}_{p,1}^r \odot \cdots \odot \boldsymbol{z}_{p,s-1}^r \odot \boldsymbol{z}_{p,s+1}^r \odot \cdots \odot \boldsymbol{z}_{p,r}^r \tag{13}$$

For the later convenience of derivation, define a matrix $\boldsymbol{U}_s^r = [\boldsymbol{u}_s^{r,1}, \cdots, \boldsymbol{u}_s^{r,D^r}]^T \in \mathbb{R}^{D^r \times K}$. Thus

$$\boldsymbol{z}_{p,s}^r = \boldsymbol{U}_s^r \boldsymbol{x}_p, \tag{14}$$

therefore, the derivative of ℓ respect to $\boldsymbol{x}_p$ can be obtained by:

$$\frac{\partial \ell}{\partial \boldsymbol{x}_p} = 1 + \sum_{r=2}^{R} \sum_{s=1}^{r} (\boldsymbol{U}_s^r)^T \frac{\partial \ell}{\partial \boldsymbol{z}_{p,s}^r}, \tag{15}$$

notice that we regard $\boldsymbol{z}_p^1$ as $\boldsymbol{x}_p$, thus the term 1 in Eqn.(15) represents the derivative of order-one interactions in $\boldsymbol{x}_p$. Use the chain rules, the same procedure can be easily adopted into the derivative respect to $\boldsymbol{U}_s^r$:

$$\frac{\partial \ell}{\partial \boldsymbol{U}_s^r} = \sum_{p \in \Omega} \frac{\partial \ell}{\partial \boldsymbol{z}_{p,s}^r} (\boldsymbol{x}_p)^T \tag{16}$$

Of course, all the operation can be easily extended to matrix multiplication rather than single sample $\boldsymbol{x}_p$, you just need to be careful when programing. Actually, It is worth noting that some frameworks such PyTorch have implemented the derivative procedure.

3 Experiments

To be continued...

References

[1] Sijia Cai, Wangmeng Zuo, and Lei Zhang. Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 511–520, 2017.