CS245
Principles of
Data Science

Term Project 3 Report
Feature Encoding for Image Classification

Instructor:
Prof.Li Niu

Author:	Student ID:
Yan Shuo	515120910083
Guan Shuwen	516021910603
Wang Lingdong	516030910113

Shanghai Jiaotong University

April 5, 2019

Abstract

This report mainly focuses on feature encoding method in terms of image classification. Firstly, we obtain SIFT features through descriptors extraction and feature encoding. Then, we use several neural network models to attain deep learning features of images. Finally we test our feature's quality by feeding them into Support Vector Machine for image classification and get the result of accuracy.

Contents

1 Image descriptors extraction

This experiment is based on AWA2 dataset. When processing original image set, we find Image collie_10718.jpg is grey-value while other images are all in form of RGB. Therefore, we transform collie_10718.jpg into RGB format for coherence. Besides, every image is resized to 224 $\times$224 to reduce computation time and storage consumption. We choose this size because deep learning features that the data website gives officially is 224 $\times$224.

1.1 Extract SIFT descriptors of keypoints

A SIFT descriptor of a keypoint is a 3-D spatial histogram of the image gradients. The gradient at each pixel is regarded as a sample of a three-dimensional elementary feature vector, formed by the pixel location and the gradient orientation.

In this project, we extract SIFT descriptors of keypoints using function in opencv-python library.

For each keypoint, a 16$\times$16 neighbourhood around is taken. It is devided into 16 sub-blocks of 4 $\times$ 4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor

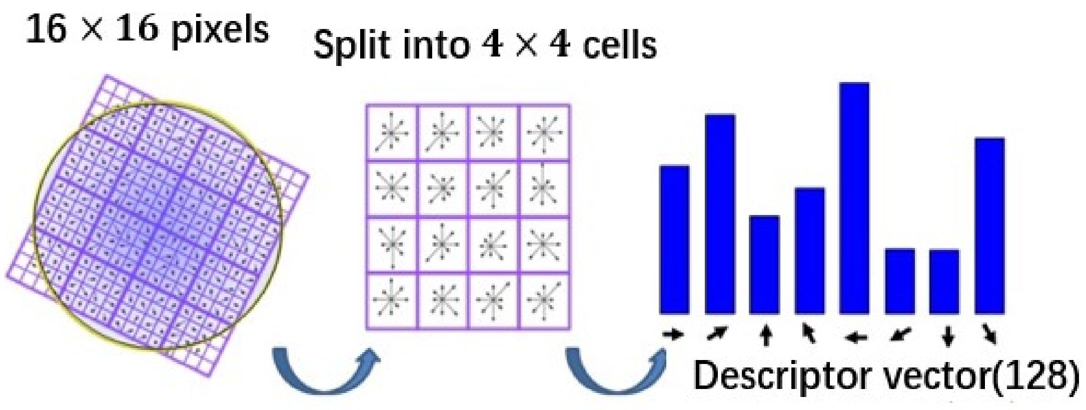


Figure 1: Process of extracting descriptors

For each picture, the number of descriptors extracted is 6763 and dimension of descriptors is 128. Take the picture antelope_10001.jpg as an example, sift descriptors of keypoints are extracted shown below.

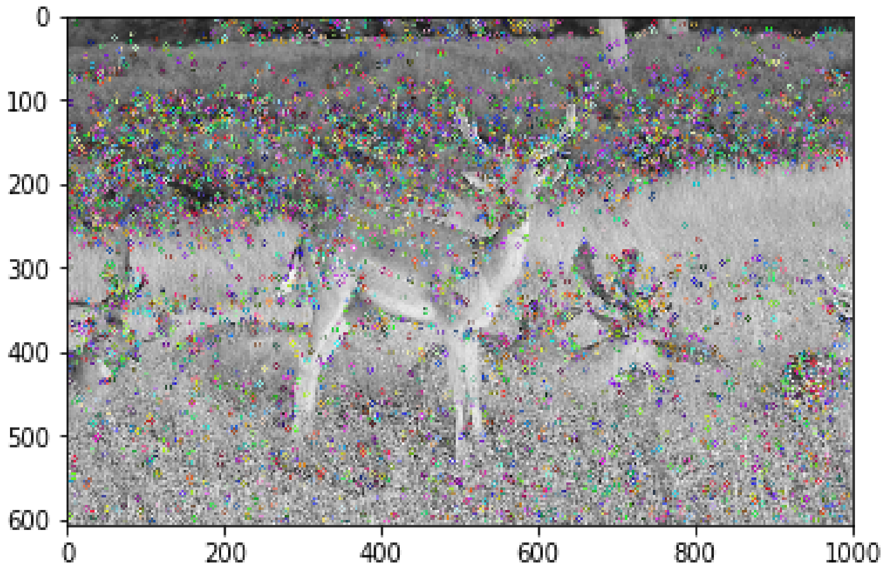


Figure 2: antelope_10001.jpg extracting SIFT result

1.2 Extract proposals of images

We use selective search to extract proposals from each image. Take image antelope_10001.jpg as an example, result of extracting proposals is shown below:

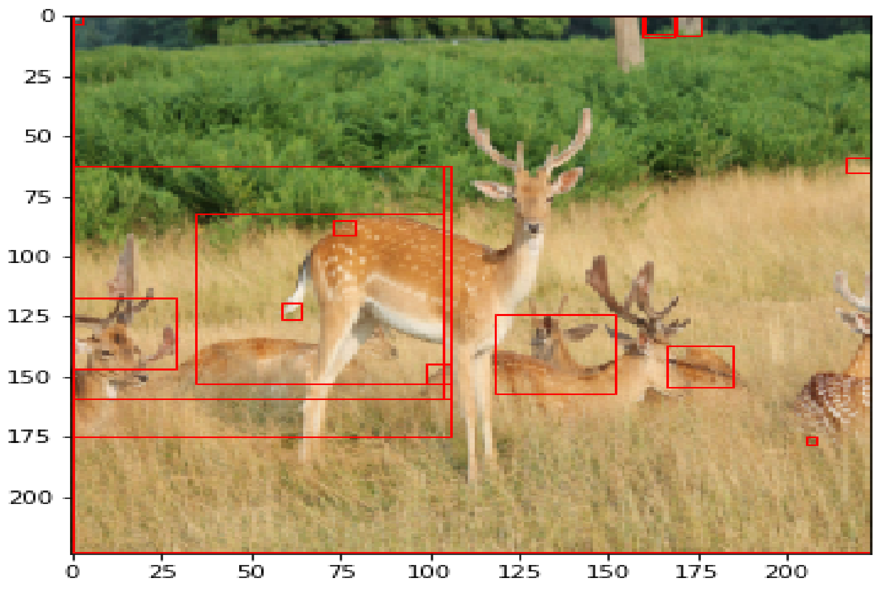


Figure 3: antelope_10001.jpg extracting proposals result

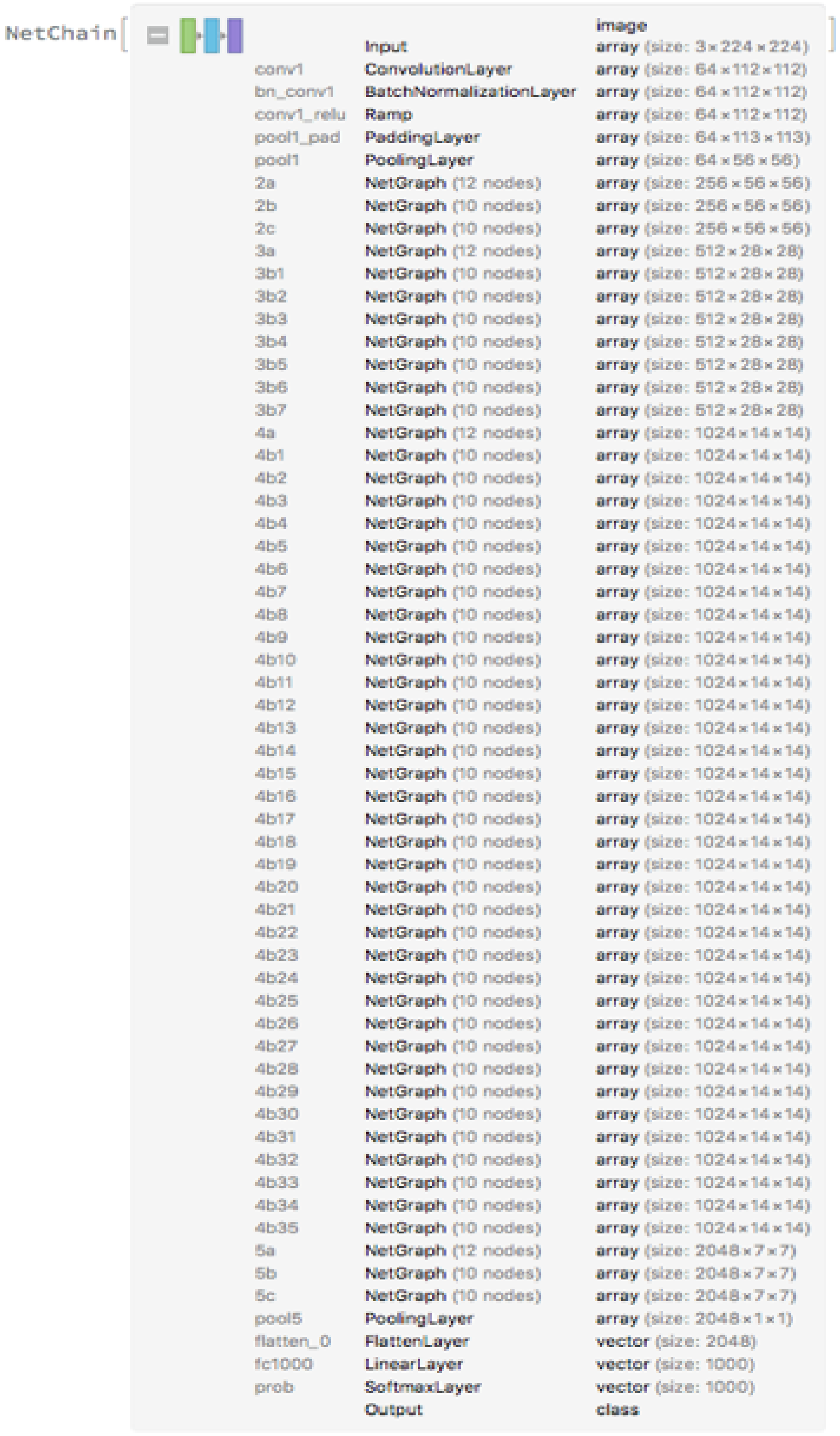
We randomly select ten proposals with size in 1000-10000. Then we utilize neural networks to get deep learning features of these proposals.

1.3 Attain deep learning features

We have experimented with five models, ResNet152, Inception_v3, AlexNet, VGG19, VGG19_bn.

1. ResNet152

   This network is built on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or short-cuts to jump over some layers. In project 1&2, deep learing features we use are extracted by ResNet101. We want to try a slightly different model and therefore adopt ResNet152. Detailed layers are shown below.



```
NetChain[                                              image
                        Input                          array (size: 3×224×224)
          conv1         ConvolutionLayer               array (size: 64×112×112)
          bn_conv1      BatchNormalizationLayer        array (size: 64×112×112)
          conv1_relu    Ramp                           array (size: 64×112×112)
          pool1_pad     PaddingLayer                   array (size: 64×113×113)
          pool1         PoolingLayer                   array (size: 64×56×56)
          2a            NetGraph (12 nodes)            array (size: 256×56×56)
          2b            NetGraph (10 nodes)            array (size: 256×56×56)
          2c            NetGraph (10 nodes)            array (size: 256×56×56)
          3a            NetGraph (12 nodes)            array (size: 512×28×28)
          3b1           NetGraph (10 nodes)            array (size: 512×28×28)
          3b2           NetGraph (10 nodes)            array (size: 512×28×28)
          3b3           NetGraph (10 nodes)            array (size: 512×28×28)
          3b4           NetGraph (10 nodes)            array (size: 512×28×28)
          3b5           NetGraph (10 nodes)            array (size: 512×28×28)
          3b6           NetGraph (10 nodes)            array (size: 512×28×28)
          3b7           NetGraph (10 nodes)            array (size: 512×28×28)
          4a            NetGraph (12 nodes)            array (size: 1024×14×14)
          4b1           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b2           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b3           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b4           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b5           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b6           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b7           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b8           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b9           NetGraph (10 nodes)            array (size: 1024×14×14)
          4b10          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b11          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b12          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b13          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b14          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b15          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b16          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b17          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b18          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b19          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b20          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b21          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b22          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b23          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b24          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b25          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b26          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b27          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b28          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b29          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b30          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b31          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b32          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b33          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b34          NetGraph (10 nodes)            array (size: 1024×14×14)
          4b35          NetGraph (10 nodes)            array (size: 1024×14×14)
          5a            NetGraph (12 nodes)            array (size: 2048×7×7)
          5b            NetGraph (10 nodes)            array (size: 2048×7×7)
          5c            NetGraph (10 nodes)            array (size: 2048×7×7)
          pool5         PoolingLayer                   array (size: 2048×1×1)
          flatten_0     FlattenLayer                   vector (size: 2048)
          fc1000        LinearLayer                    vector (size: 1000)
          prob          SoftmaxLayer                   vector (size: 1000)
                        Output                         class                    ]
```

Figure 4: Summary of ResNet-152 model we use

We remove the last three layers of the trained net so that the net produces a vector representation of a proposal.

2. Inception_v3

   Inception v3 is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. We originally want to print summary of this model, but the image is too large, we decide not to present it in the report.

3. AlexNet

   Process of AlexNe is shown below. It contains two parts, feature extraction and SVM classification. We use first part to get extracted features.

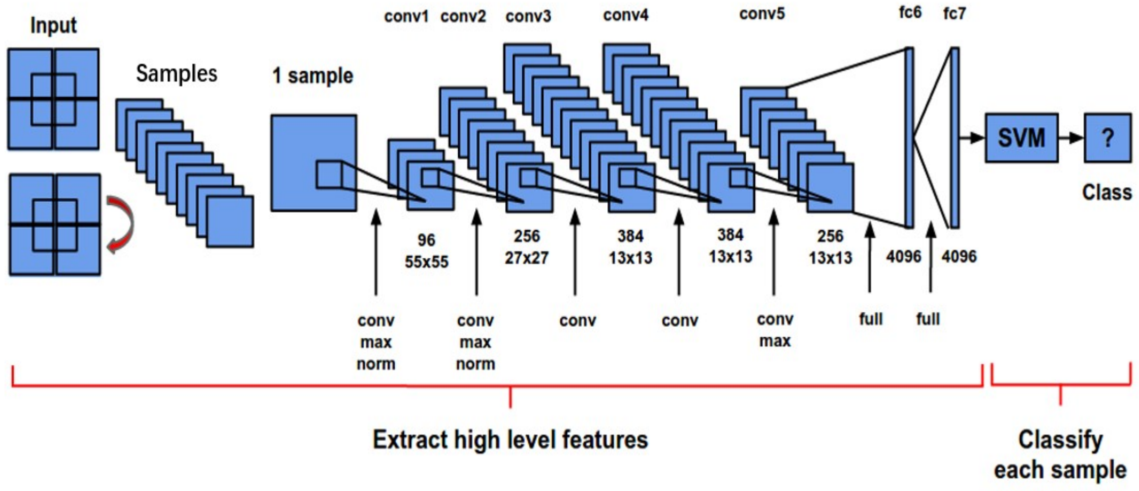


Figure 5: Structure of AlexNet model

4. Visual Geometry Group19(VGG19)

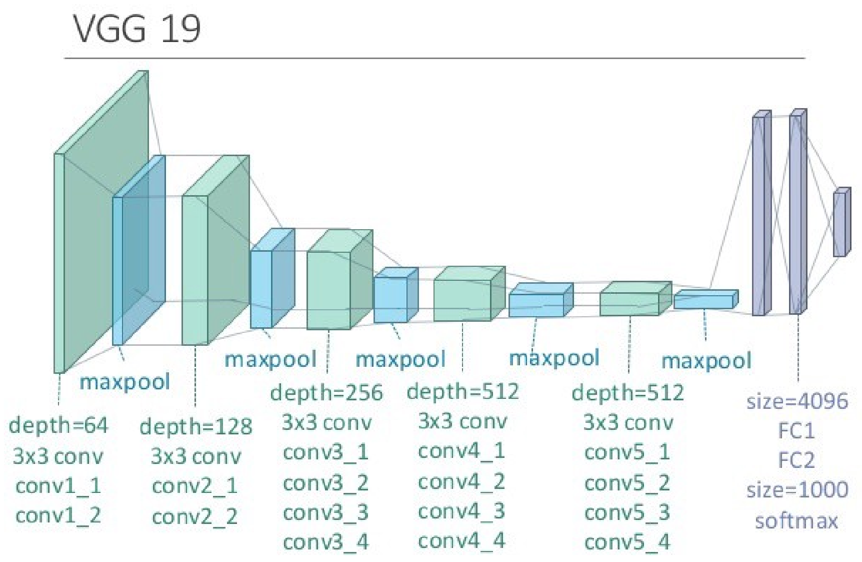


Figure 6: Structure of VGG-19 model

VGG-19 is 19 layers deep.

5. Visual Geometry Group19 with BatchNormalization(VGG19_bn)

   Compared to VGG19, VGG19_bin involves batchNormalization into network. Others parts are same.

We also utilize the above five models to directly extract features of whole images, and will deal with them in later section.

2 Feature encoding methods

In this part we try five different feature encoding methods to encode the SIFT descriptors we get in the first part.

2.1 Bag-of-word

In computer vision, the bag-of-words model (BoW model) can be applied to image classification, by treating image features as words. A bag of visual words is a vector of occurrence counts of a vocabulary of local image features.

Since descriptors of the whole training set is too large, we sample 20% of them and build the visual vocabulary by running k-means on them. Dimension of outputted features is just the number of clusters.

Table one shows result with different cluster numbers. We use PCA to do dimension reduction and feed them into SVM to see performance.

2.1.1 Performance with SVM

Table 1: accuracy of SIFT features based on Traditional BoW model

clusters k	accuracy	run_time
30	0.2002	433.652
60	0.2321	453.652
120	0.2474	496.798
240	0.2498	606.798
400	0.2523	700.453

From the result, we can see, when k increases, accuracy increases, and its increasing rate is going down. Besides, bag-of-word model's performance is not good, with no more than 30 percent. This is because it only utilizes 0-order information of the image, simply calculating the frequencies of each word(feature)'s occurrence. Position's information is largely lost in this model. Therefore, we have researched on how we can improve this model.

2.1.2 Improvement of BoW Model

As we say above, main limitations of BoW model is that it is an orderless representation of encoded keypoints found on an image. The geometric and spatial information in an image can reveal a lot about the scene, while Bag-of-Word model simpy ignores them. Therefore, one method for improving proposed by Lazebnik et al.[2] is to partition the image into increasing fine subregions and computing a histogram per region. We apply pyramid match kernel method(PMK) to perform this. The PMK is a function that can compare histograms of image descriptors at increasingly coarser grids, as seen in Figure 9.

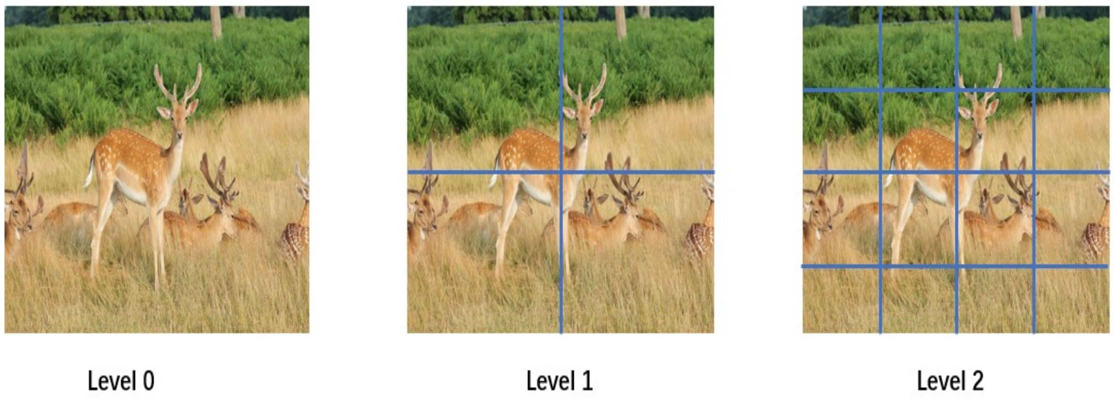


Figure 7: Spatial Pyramid construction on image

Each successive level(from level 0,1,2) in the pyramid has an increased weight applied to the histogram I^l defined as:

$$I^l = \frac{1}{2^{L-1}} forl = 0, 1, 2, ..., L-1$$

Multipling the weight with corrsponding counts of each level, combine and sum these counts, we get another version of feature vector based on improved BoW model. We also feed this group of vectors into support vector machine and get the resulting accuracy.

Table 2: accuracy of features based on Improved BoW

clusters k	accuracy	run_time
30	0.2202	473.652
60	0.2521	498.652
120	0.2773	586.798
240	0.28984	806.798
400	0.2923	990.453

From table 2, we can see that imporved BoW model's performance is better than traditional one. Accuracy and time consumption's changing trends are same. The reason for accuracy improvement is that traditional Bag-of-Word model only counts the number of local descriptors assigned to each category, whereas improved BoW model involves part of image's position information. However, since position information is not included much in this model, its accuracy is still worse than VLAD and fisher vector model, which we will discuss in the following part .

2.2 Vector of Locally Aggregated Descriptors

The "Vector of Locally Aggregated Descriptors", a.k.a., VLAD captures the mean of local descriptors. It concatenates the aggregated vectors of each category, and encodes the descriptor into a single compact vector. Compared to Bag of word model, it involves first-order information.

The main process of program is:

1. Clustering descriptors into k categories, with centroids:$\mu_1, \mu_2, ..., \mu_k$
2. Each descritpors x_i has an assignment to these k clusters.
3. For each catrgory, calculate $v_i = x - \mu_i$, and sum them together. Combine them together. Encoded feature vector for an image can be denoted by $[v_1, v_2, ..., v_k]$

The dimension of outputted feature vector is $128 \times K$(K is number of clusters). Since it's too large, we use PCA to reduce it to $8 \times K$, with preserving 0.98 variation.

2.2.1 Performance with SVM

Feeding feature vectors into support vector machines, we get resultshown below:

Table 3: accuracy of features based on VLAD

clusters k	accuracy	run_time
10	0.2731	493.725
30	0.2921	502.342
60	0.3033	686.598
100	0.3058	811.746
160	0.3096	920.235

Compared to bag of word model, VLAD achieves a better performance. Recalling BoW model, it involved simply counting the number of descriptors associated with each cluster in a codebook(vocabulary). VLAD is an extension of it. It accumulates the residual of each descriptor with respect to its assigned cluster. This first order statistic adds more information in VLAD feature vector and hence gives a better performance.

Though better than BoW model, VLAD's accuracy is still not high. We try to improve it.

2.2.2 Improvements of VLAD

According to our research, there are several improving extension possible for VLAD, primarily various normalization options. We implement the one using power normalization. Table below shows accuracy.

Table 4: accuracy of features based on improved VLAD model

clusters k	accuracy	run_time
10	0.2851	443.262
30	0.3071	456.354
60	0.3143	654.342
100	0.3198	763.548
160	0.3205	904.732

2.3 Fisher Vector

Fisher Vector is an image representation obtained by pooling local image features. FV encoding assumes that descriptors are generated by a GMM model with diagonal covariance matrices. Similarly to bag of word and VLAD, there needs an estimation of K(number of categories in GMM model) before running this encoding method. Gaussians is first learned on the training set. Once the model (μ_k, σ_k) is learned, the fisher vector can be represented as:$[\mu_1, \sigma_1, \mu_2, \sigma_2, ..., \mu_k, \sigma_k]$ Terms in the vector can be calculated as:

$$\mu_k = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ki} (\frac{\mathbf{x}_i - \mu_k}{\sigma_k})$$

$$\sigma_k = \frac{1}{N\sqrt{2\pi_k}} \sum_{x=1}^{N} q_{ki} \left[\left(\frac{\mathbf{x}_i - \mu_k}{\sigma_k} \right)^2 - \mathbf{1} \right]$$

We implement this method by ourselves, and algorithm of calculating fisher vector is described below.

Algorithm 1 Compute Fisher vector from local descriptors

```
Input: Local image descriptors X=$x_t \in R^D$, t=1,2,...,T
        Gaussian mixture model parameters $\lambda = w_k, \mu_k, \sigma_k, k = 1, 2, ..., K$
Output: normalized Fisher Vector representation $F^X \in R^{2KD}$
        1. Compute statistics
  for k = 1,...,K do initialize accumulators $s_k^0, ..., s_k^2 = 0$
  end for
  for t = 1...T do Compute posterior probability or responsibility($\epsilon(k)$) with
  $\epsilon(k) = \frac{w_k u_k(x_t)}{\sum_{j=1}^{K} w_j u_j(x_t)}$
    for k=1,...,K: do update $s_k^0, s_k^1, s_k^2$
    end for
  end for2. Computer the Fisher vector signature
  for k=1,2,...,K do
      $\phi_{\alpha_k}^X = (s_k^0 - Tw_k)/\sqrt{w_k}$
      $\phi_{\mu_k}^X = (s_k^1 - \mu_k S_k^0)/(\sqrt{w_k}\sigma_k)$
      $\phi_{\sigma_k}^X = (s_k^2 - 2\mu_k S_k^1 + (\mu_k^2 - \sigma_k^2)s_k^0)/(\sqrt{2w_k}\sigma_k^2)$
  Concatenate all Fisher vector components into one vector, and get the result.
      $\phi_\lambda^X = (\phi_{\alpha_1}^X, ..., \phi_{\alpha_K}^X, \phi_{\mu_1}^X, ..., \phi_{\mu_K}^X, \phi_{\sigma_1}^X, ..., \phi_{\sigma_K}^X$
  end for
```

The dimension of outputted feature is $2 \times K \times 128$. It's very large. So after reducing dimensions, we try some small values of k and find accuracy doesn't increase much when k is up to 30.

2.3.1 Performance with SVM

Table 5: accuracy of features based on Fisher Vector

clusters k	accuracy	run_time
5	0.29332	533.652
8	0.33543	653.652
15	0.347376	656.328
30	0.35032	786.748
35	0.35232	773.883

From the result, we can see, Fisher vector performs better than Bag of Word and VLAD model. We think this is because Fisher vector encodes a vector with richer image information(1-order + 2-order information). Besides, although including more information, Fisher Vector's time consumption doesn't increase too much. This is because fisher vector can be computed from much smaller vocabularies(k). According to the table, FV's accuracy increasing rate slows down with much smaller k, compared to BoW and VLAD. Besides, computing mean and variance is quite fast since the covariance matrices $\sum_k$ are diagonal.

2.4 Triangulation Embedding Method

In Triangulation Embedding method, anchor graph proposed for the purpose of binary encoding is considered. Feature vector representation is defined by triangulation, and is achieved by considering the set of normalized residual vectors. (C is set of centroids obtained by k-means)

$$\mathrm{r}_j(x) = \left\{ \frac{x - c_j}{\|x - c_j\|} \right\} \text{ for } j = 1 \ldots |\mathcal{C}|$$

This preserves the angular information between x and c_j while discarding the absolute magnitude. This is kind of similar to the principal of cosine similarity. Dimension of feature encoded through this method is $128 \times K$.

2.4.1 Performance with SVM

Feeding feature vectors into support vector machines, we get result table shown below:

Table 6: accuracy of features based on Triangulation Embedding method

clusters k	accuracy	run_time
10	0.2731	473.652
30	0.2921	498.652
60	0.3033	586.798
100	0.3058	806.798
160	0.3096	900.675

2.5 Super Vector

Super Vector adopts 0-order and 1-order information. There are two variants of this encoding, based on hard assignment to the nearest codeword or soft assignment to several near neighbours.(We adopted k-means(hard assignment) here.) Let $q_{ik} = 1$ if x_i is assigned to cluster k by k-means and 0 otherwise. σ is twice the mean distance between points and means within the k-means algorithm and I is the identity matrix. Also define

$$p_k = \frac{1}{N}\sum_{i=1}^{N} q_{ik}$$

$$s_k = s\sqrt{p_k}$$

$$u_k = \frac{1}{\sqrt{p_k}}\sum_{i=1}^{N} q_{ik}(x_t - \mu_k)$$

s is a constant choosen according to dataset, which is used for balancing s_k with u_k numerically. Encoded super vector is given by

$$f_{super} = [s_1, u_1^T, s_2, u_2^T, ..., s_K, u_K^T]^T$$

- Performance with SVM

Table 7: accuracy of features based on Super Vector

clusters k	accuracy	run_time
6	0.270177	433.652
12	0.29343	453.652
20	0.2987376	496.798
30	0.3012	606.798

2.6 Comparison of different encoding methods

2.6.1 Accuracy Comparison

Figure 9 shows different encoding methods results with k.

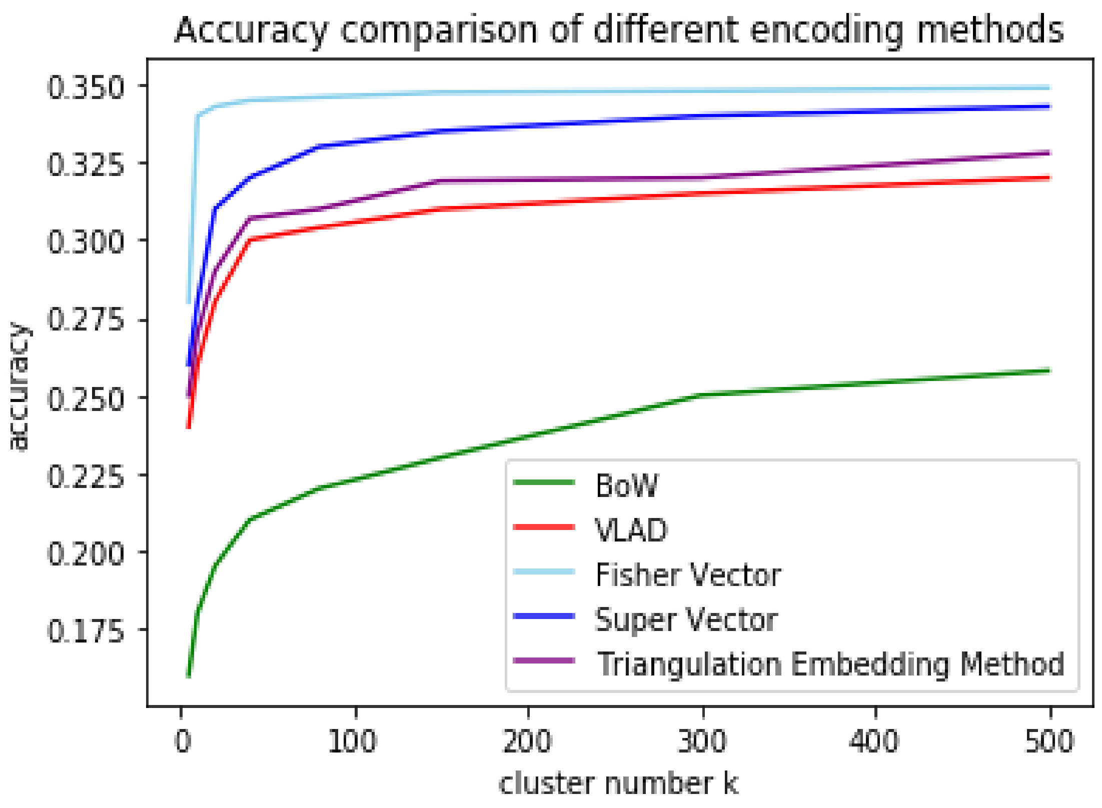


Figure 8: Encoding method's accuracy

From the result, we can see, Fisher vector's accuracy is highest, and is the fastest to be stable. Bag-of-word model has a not small accuracy difference with other methods. This is because compared to other methods, Bag-of-word method only utilizes 0-order information. All the rest models add more discriminative property in the feature vector by either adding the difference of each descriptor from the mean in its voronoi cell or the variance or both.

2.6.2 Memory and time consumption

In these five methods, super vector and fisher vector method consumes most memory, with largest dimensions($2 \times K \times D$).Bag-of-word model consumes least memory, with smallest dimensions(K). In terms of time consumption, these five methods don't have much difference, and all will increase when number of cluster increases. However, we also see from figure 9 that when k increases, each method's accuracy also increases.

It's always the case that a high accuracy will be accomponied by high memory and time consumptions. This is a trade-off. When using them, we must be clear which is more important.

3 SVM performance of deep learning features

In the first part, we have extracted two set of deep learning features using five neural networks. One is deep learning features of proposals. The other is deep learning features of whole images. In this part, we will feed them into SVM and check their performance.

3.1 Deep learning features got from whole images

We use the features got from whole images as the benchmark. Then we feed these features into SVM and get the accuracy shown in the table below.

Table 8: SVM results using deep learning features got from whole images

Pretrained neural network	Property	Accuracy
ResNet152	a residual learning framework to ease the training of networks	0.874714
Inception	a framework using inception module to reduce the number of parameters	0.865032
Alexnet	a new framework using LocalResponse Normalization	0.813405
VGG19	a framework with continuous convolutional layers and large number of parameters	0.833720
VGG19bn	VGG19 with batch normalization	0.840281

From the results above, we could see that features extracted by Resnet152 get the best accuracy. Resnet152 is preferred in the deep learning features extracted by the author of the data set so it is no surprise that it get the best accuracy on our task.

3.2 Deep learning features got from proposals

Then we try to feed encoded features from the proposals into SVM. Here are the results.

Table 9: SVM results using deep learning features got from proposals

Pretrained neural networks	Accuracy
Resnet152	0.883023
Inception	0.876832
Alexnet	0.804325
VGG19	0.849201
VGG19bn	0.856641

We could see that almost accuracy of all the methods rise. The reason might be that the pictures might have some useless information. Besides, we do not extract all the proposals and it might help the network to maintain the important information. As a result, the accuracy gets a slight improvement.

4 Bibliography

[1] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385 (2015)

[2] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, volume 2, pages 2169–2178. IEEE, 2006.

[3] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, volume 2, pages 1458–1465. IEEE, 2005

[4] Weinberger, K. Q., Blitzer, J., & Saul, L. K. (2005). Distance metric learning for large margin nearest neighbor classification. Proc. of NIPS.

[5] https://metric-learn.github.io/metric-learn/

[6] https://github.com/jlsuarezdiaz/pyDML/blob/master/dml/knn.py

[7] https://towardsdatascience.com/building-improving-a-k-nearest-neighbors-algorithm-in-python-3b6b5320d2f8