CS5187 - Vision and Image Assignment 1

Zhanghan Ke (55460880)

zhanghake2-c@my.cityu.edu.hk

Note: please scan the **README.md** in project folder to get more information about how to run the code.

All files also available from https://github.com/ZHKKKe/vi_assignment1.git

1 PART-1: PROOF

1.1 PROOF A

Let $\sigma = \sigma_x = \sigma_y$, we have:

$$G(x,y) = G(x)\,G(y) = \frac{1}{\sigma\sqrt{2\pi}}\,e^{-\frac{x^2}{2}}\ \frac{1}{\sigma\sqrt{2\pi}}\,e^{-\frac{y^2}{2}}\ = \frac{1}{2\pi\sigma^2}\,e^{-\frac{x^2+y^2}{2}} \tag{1}$$

1.2 PROOF B

From Eq. (1) we get $G(x,y)$. The 1st derivative of 2D Gaussian is:

$$\begin{aligned}\frac{\partial G(x,y)}{\partial x} &= -\frac{x}{2\pi\sigma^4}\,e^{-\frac{x^2+y^2}{2\sigma^2}} \\ \frac{\partial G(x,y)}{\partial y} &= -\frac{y}{2\pi\sigma^4}\,e^{-\frac{x^2+y^2}{2\sigma^2}}\end{aligned} \tag{2}$$

1.3 PROOF C

From Eq. (2) we get $\frac{\partial G(x,y)}{\partial x}$ and $\frac{\partial G(x,y)}{\partial y}$. The 2nd derivative of 2D Gaussian is:

$$\begin{aligned}\frac{\partial^2 G(x,y)}{\partial x^2} &= \frac{\partial(\frac{\partial G(x,y)}{\partial x})}{\partial x} = (-\frac{1}{2\pi\sigma^4})\,(1-\frac{x^2}{\sigma^2})\,e^{-\frac{x^2+y^2}{2\sigma^2}} \\ \frac{\partial^2 G(x,y)}{\partial y^2} &= \frac{\partial(\frac{\partial G(x,y)}{\partial y})}{\partial y} = (-\frac{1}{2\pi\sigma^4})\,(1-\frac{y^2}{\sigma^2})\,e^{-\frac{x^2+y^2}{2\sigma^2}}\end{aligned} \tag{3}$$

1.4 PROOF D

The Laplacian of Gaussian (LoG) filter is generated by the sum of $\frac{\partial^2 G(x,y)}{\partial x^2}$ and $\frac{\partial^2 G(x,y)}{\partial y^2}$, formally, we have:

$$LoG = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} = -\frac{1}{\pi\sigma^4}\,(1-\frac{x^2+y^2}{2\sigma^2})\,e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4}$$

2 PART-2: CONVOLUTION

2.1 Display the 48 image filters

Running **"python -m scripts.p2a"** for the results. I implement the 48 image filters myself, the code is saved in **"src\kerneler.py"**. Fig. 1 shows the results.

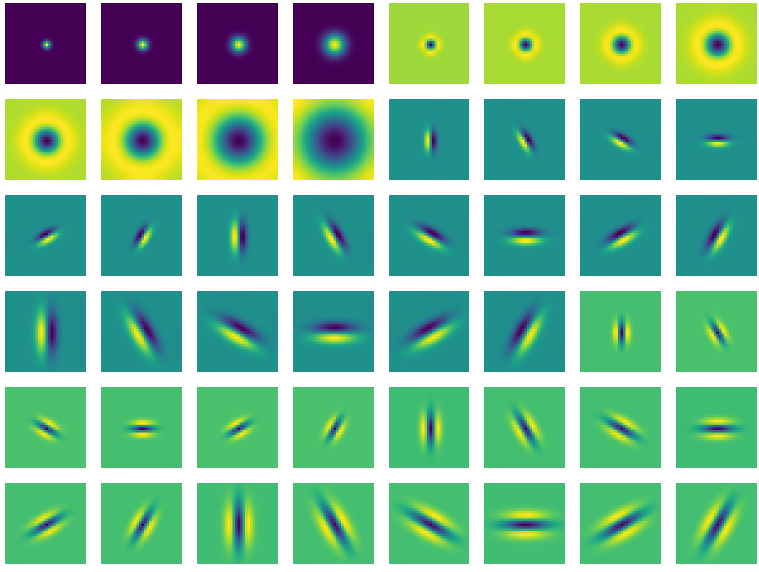


Fig. 1. 48 image filters

2.2 Display the responses of two images

Running **"python -m scripts.p2b"** for the results, which are shown in Fig. 3. Please zoom in the pictures for more details.

2.3 Largest mean and variance of each image

Running **"python -m scripts.p2c"** for the results. For "leapord.jpg", the largest mean is 87.15 from filter "Gaussian with $\sigma = \sqrt{2}$" and the largest variance is 2747.13 from filter "Gaussian with $\sigma = 1$". For "panda.jpg", the largest mean is 98.64 from filter "Gaussian with $\sigma = 1$" and the largest variance is 4724.22 from filter "Gaussian with $\sigma = 1$".

3 PART-3: IMAGE RANKING

Running **"python -m scripts.p3"** for the results. In this section, I use the 96-dims feature vector $V_i = \{v_i^0, ..., v_i^{95}\}$ from **p2c** presents a image I, and retrieve images by these vectors. However, using V for comparing directly cannot get

good results since the elements in the vector are unbalance. So I apply some pre-handle processes for them. First I normalize each dimension d by the mean and variance calculated among all n samples in dataset D as Eq. (5).

$$
\begin{aligned}
V_i^{'} &= \{norm(v_i^0), ..., norm(v_i^{95})\} \\
&\textbf{where,} \\
norm(v_i^d) &= \frac{v_i^d - mean(v_{all}^d)}{var(v_{all}^d)}, \quad d \in [0, 96), \quad i \in [0, n)
\end{aligned}
\tag{5}
$$

$V^{'}$ is the normalized vector. Then I split the $V^{'}$ into four sub-vectors and assign different balance coefficients for them as Eq. (6).

$$
\begin{aligned}
V^{''} &= [\alpha\, V^{'}[0:12],\ \beta\, V^{'}[12:48],\ \gamma\, V^{'}[48:60],\ \delta\, V^{'}[60:96]] \\
&\textbf{where,} \\
[\alpha,\ \beta,\ \gamma,\ \delta] &= [0.02,\ 0.1,\ 0.03,\ 2]
\end{aligned}
\tag{6}
$$

Finally, I calculate the similarity between $V^{''}$ by the distance or similarity function. If we not consider the color of images, the results of first four queries are not bad. The file **"results\p3_imvec.pkl"** saves the extracted features of all images, reading the features from this file could reproduce the result quickly. Fig. 2 shows the results.

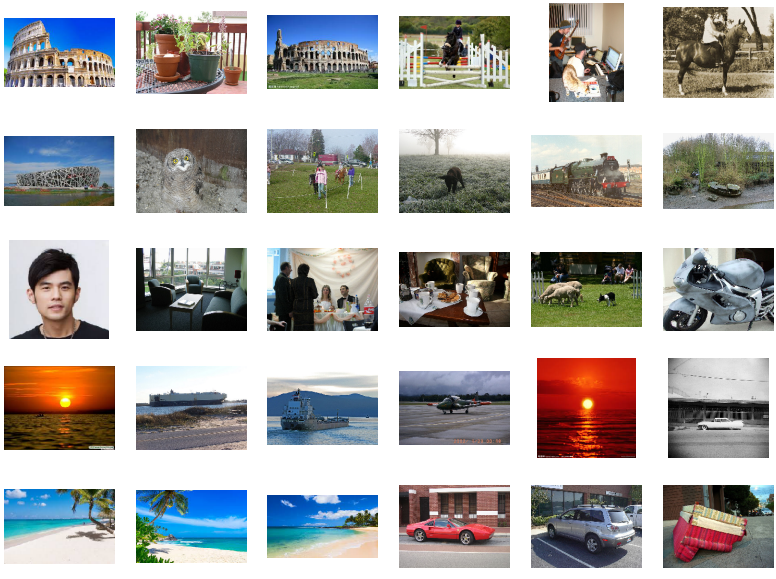
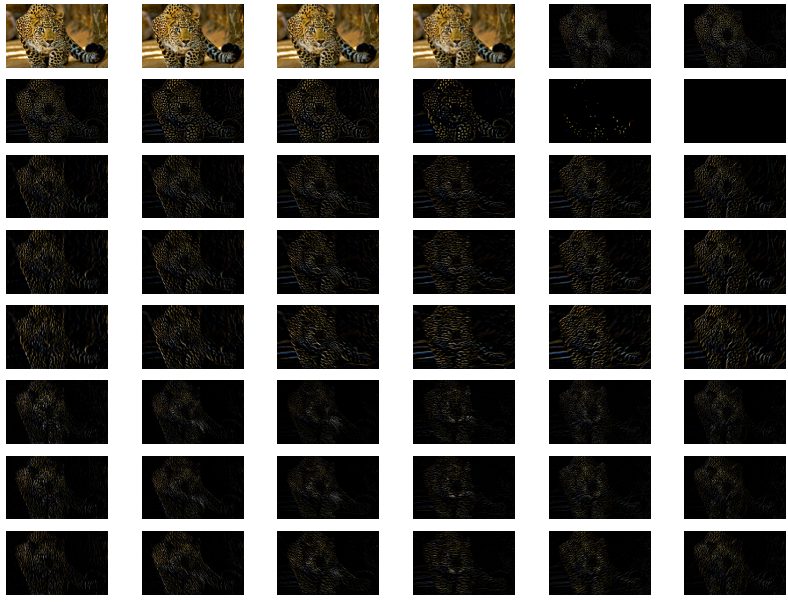


Fig. 2. The results of retrieve images by 96-dims feature vectors in **PART-3**. The first column is the query while the columns 2-6 show the top1-top5 matched images.



(a) leapord.jpg

(b) panda.jpg

Fig. 3. The 48 responses of the images “leapord.jpg” and “panda.jpg” after performing convolution with the filter bank.

4 PART-4: METHOD COMPARISON

4.1 Color Histogram and Deep Learning

In this part, I implement two feature extraction methods, deep learning and color histogram.

For deep learning, I use PyTorch [3] to implement an Auto-Encoder model as the feature extractor since the labels of images are unknown, and I treat the outputs of encoder as the feature of image. Running **"python -m scripts.p4a2 –train"** to train the model. However, the result is not pretty good since the training set contains only 2000 images. A better result could be gained by running **"python -m scripts.p4a2 –pretrained"**, which uses a ResNet50 [2] model pre-trained by ImageNet [1] as the feature extractor. And the image's feature is a 1000-dims vector outputted by the network. Fig. 4 shows the results of these two settings.

For color histogram, running **"python -m scripts.p4a1"** for the result. I calculate the color histogram on RGB channels independence with 16 bins, then normalize each channel by image size. Finally I concatenate them as a 48-dims feature for each image. Fig. 5 shows the retrieve results.

4.2 Methods comparison

As the results shown in Fig. 2, Fig. 4 and Fig. 5, the best result is generated by deep learning method. Here I use the top1 and top5 error as the metrics (Table 1). Deep learning gets a 0% error rate, filter bank gets the second place, and color histogram is the worst.

The results are very make sense since the mean-var vectors from filter bank only consider the texture information of the image while the color histograms only use the color information of the image. However, the deep learning combines both color and texture information. In addition, the deep learning also does much more complex calculation than other two methods.

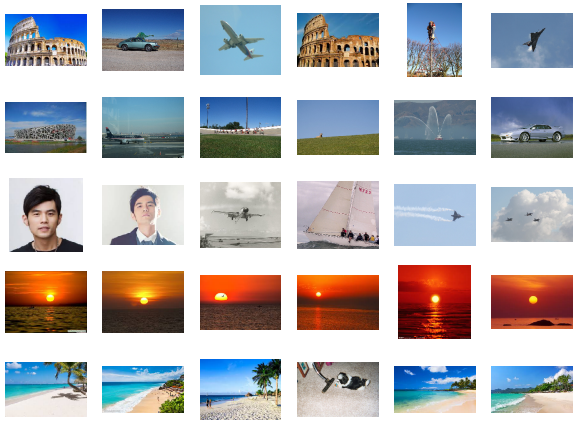
Table 1. top1 and top5 error rate comparison

Method	Top1 Err	Top5 Err
Deep Learning	0.0%	0.0%
Filter Bank	80.0%	76.0%
Color Histogram	100.0%	88.0%

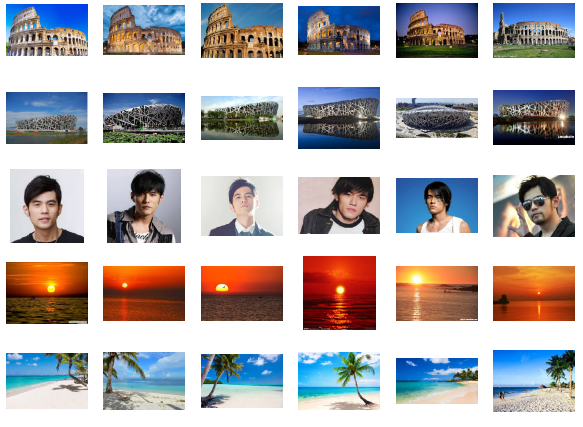
4.3 Methods fusion

Since all features extracted in my work are 1-dim vector. So I fuse them by the similar way as Eq. (6). The final feature contains three sub-vectors from different methods. However, the balanced result is not as good as the result which use deep

learning only. The reason is that the deep learning is a really powerful method and it extracts very high level features. The feature from other two methods not good enough and will decline the performance, so the result depends on the balance coefficients. But the best result of fusion is not better than using deep learning only.



(a) AutoEncoder trained by 2000 Images



(b) ResNet50 trained by ImageNet

Fig. 4. The retrieve results by two deep learning feature extractors. (a) The Auto-Encoder trained by 2000 images cannot gain a great result since the limitation of data. (b) The ResNet50 pre-trained by ImageNet could extracts great features from images.

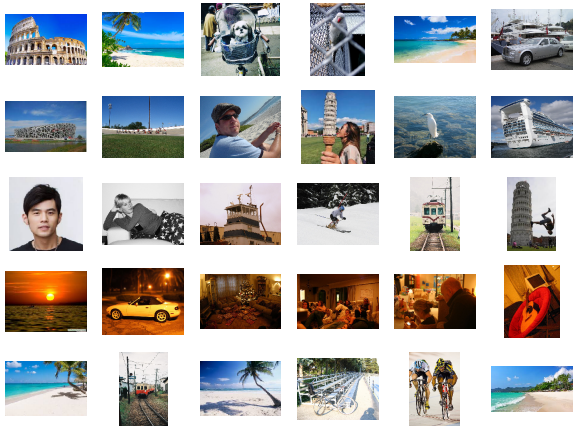


Fig. 5. The retrieve results by color histogram features.

5 PART-5: COMPETITION

I select the deep learning (pre-trained ResNet50) as my final method, and the retrieval result is saved in "rankList.txt". The Fig. 6 also shows the top10 similar results of this method.

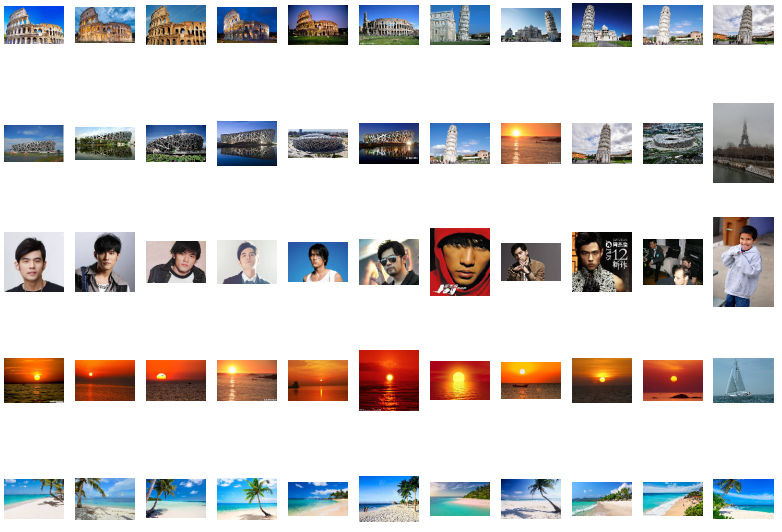


Fig. 6. Top10 retrieve results by deep learning with pre-trained ResNet50 model. The first column is the query while the columns 2-11 show the top1-top10 matched images.

References

1. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
2. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
3. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.