Software Challenge

Web Application For Image Retrieval

Vinod Rajendran

1 Introduction

Based on the requirement of cape analytics software challenge, a web application for image retrieval has been developed. In the web application, the user can browse and upload the image. Once the image is uploaded, the top three similar images from the given dataset are displayed (see Figure 1). In addition, the application stores the date and time, features, filename, and top three results of uploaded image in the database.

2 Web Server Application

As suggested, **Docker** technology used in this challenge for packaging everything. This, in turn, can be used to run applications across systems and machines [1].

Nginx is used to serve this "dockerised" application. It provides a very high performance for web server / (reverse)-proxy). It is capable of handling a lot of requests at a time. It is also useful for serving static files such as images, scripts or style-sheets [2].

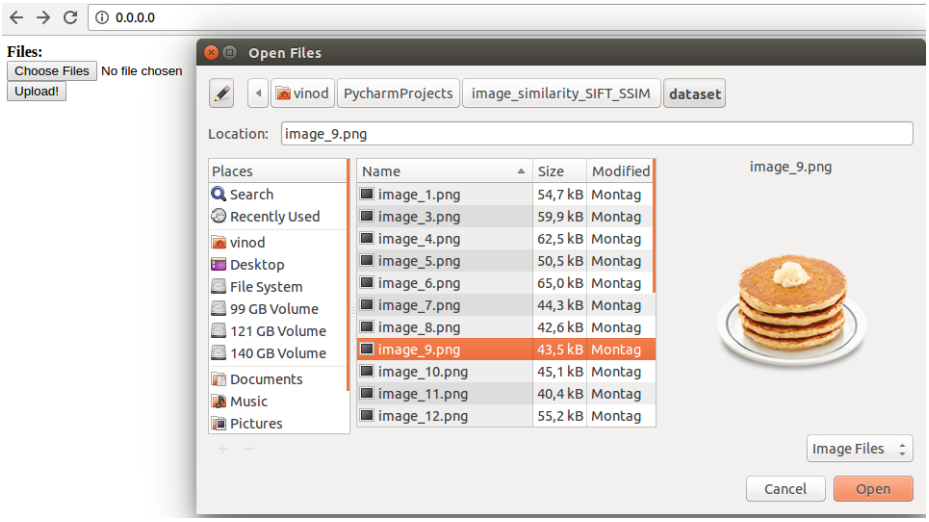
Linux (ubuntu 14.10) has been chosen has the os environment. The whole applications was implemented using python web frameworks : **Django** and **Flask**. These frameworks were considered to be the obvious choice for beginners due to the lack of roadblocks to getting a simple app up and running [3].

MongoDB is used as a database to store the details of uploaded image. It stores data in rich structures like maps of maps of lists, which contains integers and floating point data. MongoDb supports easy sharding, much easier than SQL.

3 Image similarity algorithm

In this challenge, some of the techniques exist in computer vision and machine learning was implemented.

(a)



(b)

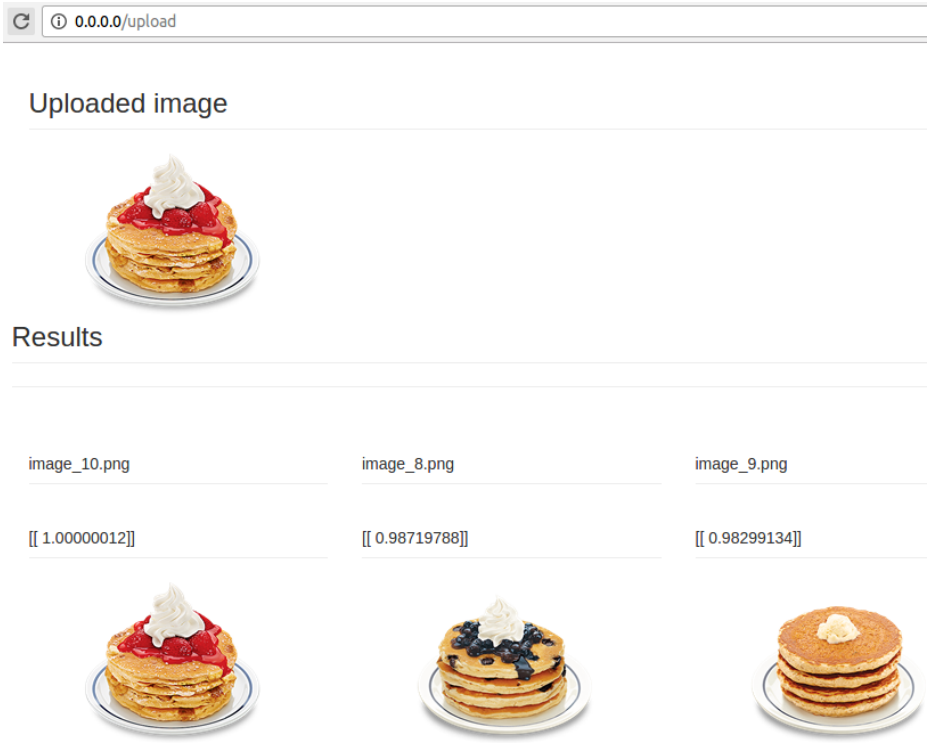


Fig. 1: (a) A page to upload the image. (b) The uploaded and the top 3 similar images are displayed.

3.1 3D color histogram in the HSV color space

This technique was replicated from [4]. In short, images are represented using HSV color space. Then histograms are used to give a (rough) sense of the density of pixel intensities in an image.

A 3D HSV color descriptor will ask a given image how many pixels have a Hue value that falls into bin 1 and how many pixels have a Saturation value that falls into bin 1 and how many pixels have a Value intensity that falls into bin 1. The number of pixels that meet these requirements is then tabulated. This process is repeated for each combination of bins; however, it is done in an extremely computationally efficient manner.

In my experiment I'll be utilizing the same parameter settings like in [4]. A 3D color histogram in the HSV color space with 8 bins for the Hue channel, 12 bins for the saturation channel and 3 bins for the value channel, yielding a total feature vector of dimension 8 x 12 x 3 = 288.

Instead of computing a 3D HSV color histogram for the entire image, a 3D HSV color histogram for different regions of the image is computed. A separate file with image id's as index and their corresponding features are created.

For each row, the color histograms associated with the indexed image are extracted and then compare it to the query image features using the chi-squared distance.

Images that have a chi-squared similarity of 0 will be deemed to be identical to each other. As the chi-squared similarity value increases, the images are considered to be less similar to each other.

3.2 SIFT and SSIM

SIFT Scale-invariant feature transform (SIFT) is an algorithm in computer vision to detect and describe local features in images. SIFT is a method to detect distinct, invariant image feature points, which easily can be matched between images to perform tasks such as object detection and recognition, or to compute geometrical transformation between images [5].

In my experiment, SIFT functionalities available in OpenCV are used to detect and compute the key-points from the image [6]. Then the Brute-Force matcher with knn matches is applied to key-points to return a set of k matches. With few constant definitions and custom normalizations, similarity scores are sorted.

SSIM The Structural Similarity (SSIM) index is a method for measuring the similarity between two images. The SSIM index can be viewed as a quality measure of one of the images being compared, provided the other image is regarded as of perfect quality. The SSIM Index quality assessment is based on the computation of three terms, namely the luminance term, the contrast term and the structural term [7].

In my experiment, the default SSIM function available from OpenCV without any modification is used to determine the scores.

In both SIFT and SSIM techniques, images that have a similarity score of 1 will be deemed to be identical to each other.

3.3 KNN with SURF

The idea behind this experiment is to apply machine learning on top of the computer vision technique. Speeded Up Robust Features (SURF) is a local feature detector and descriptor partly inspired from SIFT. It is reported that SURF is faster than SIFT. Its feature descriptor is based on the sum of the Haar wavelet response around the point of interest [8].

In my experiment, SURF functionality available in OpenCV are used to determine the key-points and descriptors. For the given image, knn trains the data based on the extracted key-points and descriptors. This trained model is then used to find the nearest neighbors from the descriptors of all the images present in the dataset. Based on the corresponding distances, the closest ones are grouped and sorted. The images that have a distance of 1 will be deemed to be identical to each other.

3.4 Deep Learning

Deep learning models have achieved great success on image classification tasks [9]. However, similar image ranking is different from image classification.

The given dataset is very small and it's not feasible to perform deep learning on it. Therefore, web scraping was performed to download more images based on the keyword "pancakes" and "tesla models". These downloaded images were then converted into a jpg format. It is important to note that only 100 images were downloaded for each keyword.

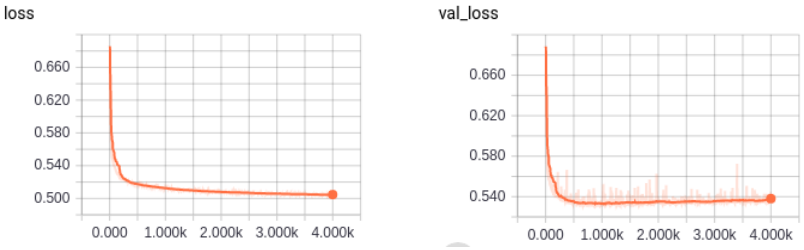
Two types of autoencoders namely convolution autoencoders (CAE) and variational autoencoders (VAE) are employed in this challenge. The main idea is

to obtain a model (i.e. encoded representation) from the downloaded images. The generated model is then used to predict the features of the uploaded image and the images present in the given dataset. Once the features are obtained, cosine-similarity distance is used to measure the difference between the images.

CAE Since our inputs are images, it makes sense to use convolutional neural networks (convnets) as encoders and decoders. In practical settings, autoencoders applied to images are always CAE –they simply perform much better.

The encoder will consist in a stack of three Convolution2D and MaxPooling2D layers (max pooling being used for spatial down-sampling), while the decoder will consist in a stack of three Convolution2D and UpSampling2D layers. The activation function defined for each layer was 'relu' and for back-propagation 'adadelta' method was defined.

The network was trained with 160 images for 4000 epochs with a batch size of 10 and it is tested with 40 images. The minimum validation loss achieved was 0.532, after which it resulted in overfitting. The loss and validation loss plot are illustrated in Figure 2. And also the encoded and decoded representation of 10 images are shown in Figure 3.



((a)) Loss of CAE ((b)) Validation loss of CAE

Fig. 2: Loss and Validation loss of CAE

I tried to improve the model with the data augmentation function like random rotation, flip, etc. available in keras library. However, there was not much improvement observed.

VAE It is slightly more modern and interesting take on autoencoding. More precisely, it is an autoencoder that learns a latent variable model for its input data [10].

(a)

(b)

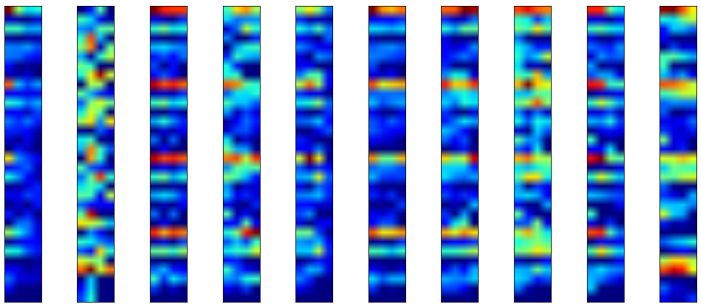
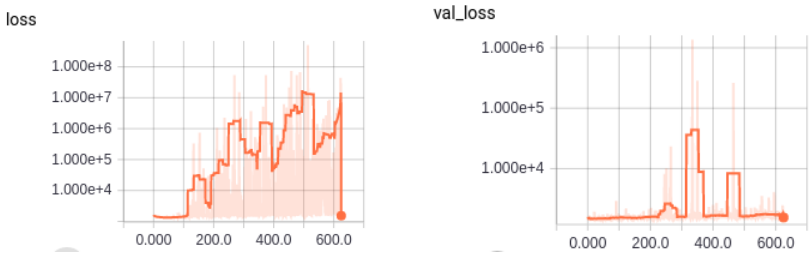


Fig. 3: (a) Decoded representation of 10 images obtained from CAE. (b) Encoded representation of 10 images obtained from CAE.

The model was trained using the end-to-end model, with a custom loss function: the sum of a reconstruction term, and the KL divergence regularization term. In my experiment, the latent space dimension was defined as 10.



((a)) Loss of CAE ((b)) Validation loss of CAE

Fig. 4: Loss and Validation loss of VAE

However, this experiment didn't workout well as one can observe in Figure 4,the losses didn't converge well. This demands a change in parameter settings.

4 Future Work

Future work can be categorized into two sections: web application and image similarity algorithm.

In the case of web application, a database can be used to store the feature descriptors of all the images. The design of UI should be improved. Finally, the server response time should be reduced to display the results quickly.

In terms of similarity algorithms, the deep learning methods can be facilitated by using more number of images for training. The above mentioned experiments should be repeated for a different set of network parameters.

5 Conclusion

A web application for determining the image similarity has been developed successfully by using docker technology along with the computer vision and machine learning techniques. However, as mentioned in the future work there is a lot of room for improvement. The methods used here might serve as a foundation for building intelligent image retrieval machines.

References

1. Digitalocean, "How to install and use docker: Getting started," 2013. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-getting-started
2. DigitalOcean, "How to containerize and use nginx as a proxy," 2013. [Online]. Available: https://www.digitalocean.com/community/tutorials/docker-explained-how-to-containerize-and-use-nginx-as-a-proxy
3. C. D. blog, "Choosing python web frameworks: Django and flask," 2016. [Online]. Available: http://www.codingdojo.com/blog/choosing-python-web-frameworks/
4. pyimagesearch, "The complete guide to building an image search engine with python and opencv," 2014. [Online]. Available: http://www.pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/
5. P. Android, "Sift based tracker," 2012. [Online]. Available: https://jayrambhia.wordpress.com/2012/09/24/sift-based-tracker/
6. OpenCV, "Introduction to sift (scale-invariant feature transform)," 2012. [Online]. Available: http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html
7. Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
8. OpenCV, "Introduction to sift (scale-invariant feature transform)," 2012. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html
9. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
10. C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.