Content Based Image Retrieval

I. Introduction

Image retrieval has become one of the important features of search engines nowadays. Efficient ways of image searching are required from the users on a daily basis, for fashion, medicine, architecture etc. Therefore, various techniques were presented and are being used in many search engines (Google, Yandex). There are two types of image retrieval techniques: text-based and content-based. In text-based systems images are manually annotated and then image retrieval is performed by database management system. However, there are two main disadvantages with this approach. Firstly, considerable amount of human labor is needed to annotate the images. Secondly, the annotation accuracy depends on human perception [1]. Therefore, in this paper we decided to implement content-based image retrieval technique. In this type of approach images are indexed by the content, e.g., color, shapes in it, textures. Given an input our model has to find similar images from the dataset. The main algorithm of content-based image retrieval (CBIR) can be seen from the Figure 1. It can be seen that first we have to extract the features of the training dataset images and then compare these features with the test image features. To compute the similarity we used euclidean distance metric measure.

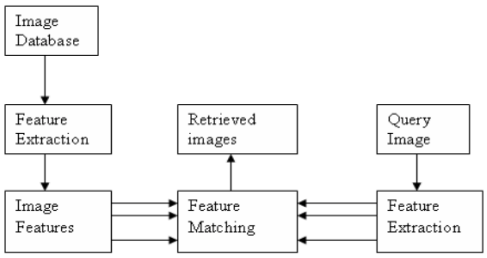

Fig. 1: CBIR Algorithm [2]

In this paper we implemented different CBIR algorithms to compare the accuracy of each technique. In furhter sections we will describe the working principles of autoencoders, different kNN methods and CNN, and explain how they can be used to retrieve similar images.

II. Dataset Description

As our dataset we chose Caltech 101 dataset. It contains 101 object categories, varying 40 to 800 images per category. Most categories have about 50 images. However, we deleted "Google Backgrounds" folder, because these images do not belong to any category they are very random images. All the images will be submitted with other codes and video presentations.

III. Autoencoders

Autoencoders are a part of machine learning ecocystem, which is a part of unsupervised learning. They try to copy input to the output, but with some losses. Autoencoders usually compress the input image to obtain some latent-space representation and through this representation they reconstruct the input image. We will extract features of an input image from latent-space representation. To complete this model we used PyTorch Library.

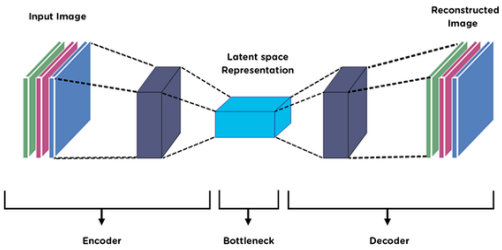

Fig. 2: Autoencoder architecture [3]

A. Data Pre-processing

First step in building an autoencoder was a data preprocessing. We converted our dataset path, containing images, to a pandas dataframe. To manipulate data easily. Then we splitted the dataset into training and validation sets, test size was chosen as 0.2. After this transforms.compose was implemented. It consists of transforming our PIL image to tensor, resizing each image to the size 128x128 and normalising the image. Normalization means setting the mean and standard deviation to 0.5 for each channel (RGB). It will work like this: **image = (image - mean) / std**
Also, we converted checked each incoming image and if it was in Grayscale we converted it to RGB.

B. Neural Network Layers

The second step was to build autoencoder model. Our neural network consists of 7 encoders and 7 decoders. At each step as activation function rectifying linear unit (ReLU) was chosen. In order not to get confused with the layers, parameter numbers we consulted GitHub repository on convolutional arithmetic [4]. In total our net has 3,313,731 parameters and all of them are trainable parameters.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 512, 512]	1,792
ReLU-2	[-1, 64, 512, 512]	0
Conv2d-3	[-1, 64, 512, 512]	36,928
ReLU-4	[-1, 64, 512, 512]	0
MaxPool2d-5	[-1, 64, 256, 256]	0
Conv2d-6	[-1, 128, 128, 128]	73,856
ReLU-7	[-1, 128, 128, 128]	0
Conv2d-8	[-1, 128, 126, 126]	147,584
ReLU-9	[-1, 128, 126, 126]	0
MaxPool2d-10	[-1, 128, 63, 63]	0
Conv2d-11	[-1, 256, 32, 32]	295,168
ReLU-12	[-1, 256, 32, 32]	0
Conv2d-13	[-1, 256, 32, 32]	590,080
ReLU-14	[-1, 256, 32, 32]	0
Conv2d-15	[-1, 256, 32, 32]	590,080
ReLU-16	[-1, 256, 32, 32]	0
MaxPool2d-17	[-1, 256, 16, 16]	0
ConvTranspose2d-18	[-1, 256, 16, 16]	590,080
ConvTranspose2d-19	[-1, 256, 16, 16]	590,080
ReLU-20	[-1, 256, 16, 16]	0
ConvTranspose2d-21	[-1, 128, 33, 33]	295,040
ConvTranspose2d-22	[-1, 64, 65, 65]	73,792
ReLU-23	[-1, 64, 65, 65]	0
ConvTranspose2d-24	[-1, 32, 129, 129]	18,464
ConvTranspose2d-25	[-1, 32, 257, 257]	9,248
ReLU-26	[-1, 32, 257, 257]	0
ConvTranspose2d-27	[-1, 3, 512, 512]	1,539
Tanh-28	[-1, 3, 512, 512]	0

Total params: 3,313,731
Trainable params: 3,313,731
Non-trainable params: 0

Fig. 3: Autoencoder model summary

C. Training the Model

The next step is to build the training function. It is important to save our model at each epoch. A statedict is simply a Python dictionary object that maps each layer to its parameter tensor.We will initialize statedict from checkpoint to the model, to the optimizer and the losses. We have to optimize and apply backward to our model only when it is in training phase. As an optimizer we chose Adam optimizer and as an loss metric Mean Squared Error. Next, we wrote training function and implemented all the above mentioned parameters. Number of batches was chosen as 40 and epochs as 30.

D. Extracting latent features

One of the most important steps is extracting the latent features. Using these features we will be able to retrieve similar images. We unsqueeze the tensor to obtain row matrices and use model.encoder function of PyTorch to obtain latent features. Next, we have to map indexes of latent features with latent features.

E. Similarity Search

To perform image retrieval we used euclidean search metrics [2]. First we have to input an image to our model, then latent features of this images are obtained. Then using euclidean search latent features of our test image is compared with the latent features of our trained images. Closest 10 indexes of images were returned.

Fig. 4: Training and validation errors

Since autoencoder is unsupervised learning system, its accuracy was not as good as we expected. There were test images that our model has identified and retrieved 100 percent exactly the same images. Like this with airplane and faces:

Fig. 5: Airplane image retrieval

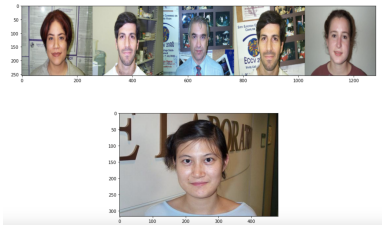

Fig. 6: Faces image retrieval

Also, there were cases where our model's accuracy was not perfect:

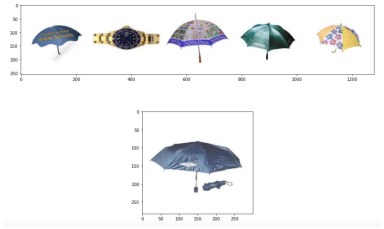

Fig. 7: Umbrella image retrieval

However, we also had situations when our model couldn't find any relevant image:

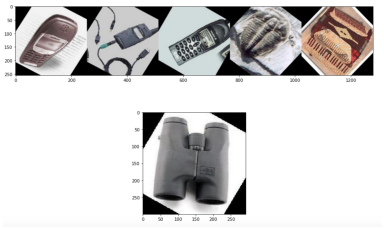

Fig. 8: Accuracy 0

It can be explained in a simple way. Autoencoder learns automatically the features that are in the image. For example, orientation of an image, background and foreground colors, how many items are in an image. We can see this situation here in our examples, our model had done very well when test and train images were of the same orientation, had almost the same colors and orientation. That's how our model searches for the same images. But, we haven't implemented image segmentation here, so we could obtain information about what is actually there in an image. This way we could get qualitative information about image. Let's discuss the case where our model failed to find similar images. We can see that all of them are turned 45 degrees to the right and background color is black, also there's only one element in an image and its color is black too. We think that our model extracted only these features from our training set and compared them with test image.

REFERENCES

[1] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, "A survey of content-based image retrieval with high-level semantics," Pattern Recognition, vol. 40, no. 1, pp. 262–282, 2007.

[2] S. Patil and S. Talbar, "Content Based Image Retrieval Using Various Distance Metrics," Lecture Notes in Computer Science, pp. 154–161, 2012.

[3] D. Birla, "Basics of Autoencoders," Medium. [Online]. Available: https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f

[4] Francesco, "Convolution Arithmetics", Github. [Online]. Available: https://github.com/vdumoulin/convarithmetic