Reverse Visual Search in Images and Videos

Namratha Upadhya nf2207@nyu.edu
Yongtai Chen ytchen@nyu.edu
Keerthana Ravikumar - kr2695@nyu.edu

Abstract

In this paper we propose to perform reverse visual search to find similar images as query images. We are aiming to improve the accuracy of the system that has been developed, and we successfully improved the accuracy of reverse visual search by extracting the ROI first. After extracting the ROI, we extract the embeddings from the ROI only, so we can have better representation vectors, this way we successfully improved the accuracy while maintaining the calculation speed.

In the last part, we propose a fast method to do reverse visual search in videos by filtering the frames, we drop unnecessary frames: empty frames and “duplicate” frames. We only handle the key frames, so we can improve search performance by reducing data. This way we also get a high accuracy because key frames are kept.

Keywords: reverse visual search; embedding; ROI; filter

Introduction

Back in the 1990s, when the Internet was still a young baby, the speed of the Internet was slow, and the capacity of the Internet was small. The main content on the Internet is text, so the search engines invented that time can search texts, and those search engines work really well on text. But the internet has developed really fast in the last 10 years, new technology makes it faster and stronger, people are surrounded by images and videos, some search engines like Google can search images, and we are interested in how they can search using images, so we did some research in this field.

Compared with text search, reverse visual search is much more difficult. When doing text search, we can query text, find sentences with query keywords, but how do we search images? Or, what should we find when querying an image? Well, that depends on what kind of image we are querying.

To do reverse visual search, we should have a set of images to search, after all, we are searching images, not creating images. In this project, we will do reverse visual search on the LFW dataset, which is a dataset of human faces. We take an image file as an input query and return results related to the image, that is, identify people's faces and find similar faces in the dataset.

Related work

Visual search is a relatively new field of study in Artificial Intelligence, hence there isn't much research and related work. Most of these algorithms include two steps:

- Feature extraction - extracts the features of an image.
- Hashing - hashes the features so the complicated visual information can be represented in a one dimensional array.

Visual Search has the following applications:

1. Google image search (https://images.google.com/)
2. Bing image search (https://www.bing.com/images/feed)
3. TinEye (https://tineye.com/)
4. Pixsy (https://www.pixsy.com/)

Dataset

Labeled Faces in the Wild (http://vis-www.cs.umass.edu/lfw/) is a benchmark dataset for facial verification. LFW contains 13,233 images for a total of 5749 people. Among these there are 1680 people with two or more images.

Doing reverse visual search on a face dataset is like face recognition, we are given a bunch of images, of different people's faces, as candidates.

we need to find the similar faces for a given target face. If the person in the target image has other images in the dataset, we need to find them, if not, we can find similar faces.

We handle the dataset using following methods:

Storing image paths

- Download the dataset and store it in your google drive > MyDrive.
- Traverse through the dataset and find any files with image file extensions such as '.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG'.
- Store these filenames and directory names in an array.

Baseline

Method Overview

In this approach, we replicate a simple but easy understanding method, which has two parts, the first part is building a KNN index, and the second part is searching using an image. To create the baseline system we are:

- Using a pre-trained deep learning model (ResNet-50)
- Extracting the features(embeddings) for the faces in our dataset.
- Then we will use these embedding vectors to find similar images by using sklearn's nearest neighbor algorithm.

1. Create KNN reference index

- Use a pre-trained neural network, remove the last layer.
- Push the images into the neural network, since the last layer is removed, we will get image features(embeddings) from the endpoint of the neural network.
- Build a KNN using the embeddings extracted above.

We have tested several neural networks on embedding extraction, including VGG16, mobilenet, inceptionv3, ResNet50, xception, and find **ResNet50** is better than others on efficiency and accuracy. ResNet50 extracts 2048 features for each image.

2. **Query image**

- Push the image into same neural network, extract the embeddings
- Compare the target image embeddings with all the embeddings in the dataset, find the top similar images using KNN

Model

We are using the ResNet-50 model. It is 50 layers deep and uses weights from ImageNet by google. It uses max pooling to condense the feature maps.

Extracting Features

1. Define input image shape as 244, 244, 3 (input size for ResNet-50)
2. Resize and normalize the images from LFW to the same format.
3. Predict the features and flatten the numpy array.
4. Normalize the flattened 1-dimensional feature array.
5. Return the extracted features.
6. Save the features in file.

Building KNN Index

We are using KNN function from python sklearn package (https://scikit-learn.org/stable/modules/neighbors.html), we choose brute force algorithm with Euclidean distance metric, set k = 5, build the KNN, fit the embeddings extracted above.

Query the Target Images

Extract features using exactly the same neural network with same weights, using the KNN built above to predict the results, despite the requirement of top 20 images, we select 5 most similar images because in the LFW dataset, there aren't many image for same person, a small number of similar images can show our model better.

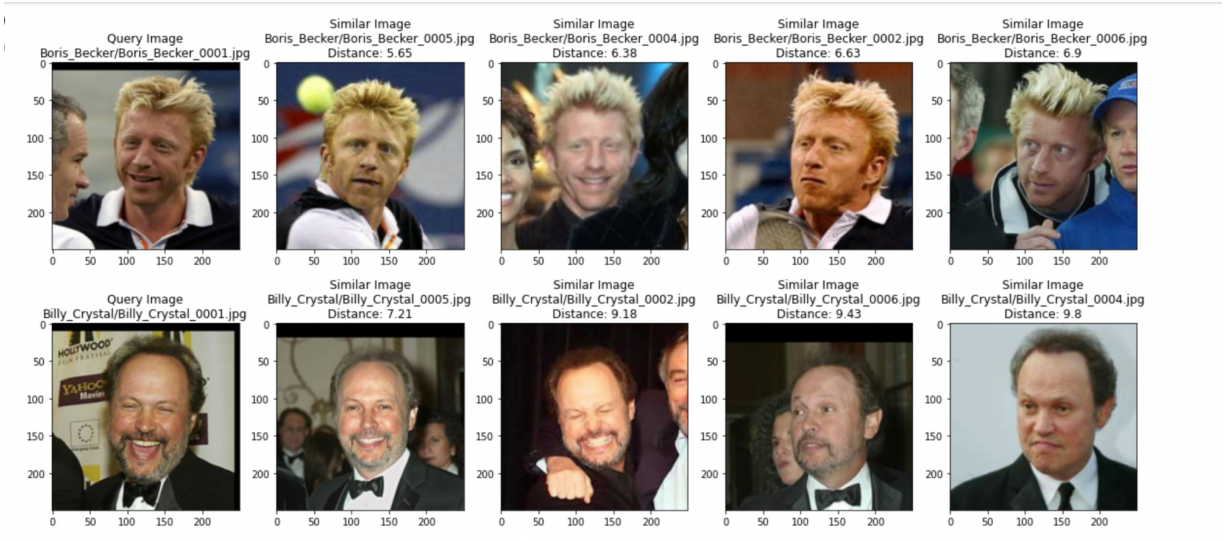
Improvements

The accuracy obtained by the above mentioned baseline model was significantly lower than expected. We experimented with different models to check the change in accuracy and compared to the baseline model.

Our Approach

We derived from our results of the baseline model that the potential reasons for error could be that the whole image is used to perform similarity search. To improve upon this error we cropped the part of the image that was of interest to us, i.e, the faces. We decided to find the **region of interest.**
On further research and study of various methods to achieve this we found **FaceNet** (https://arxiv.org/abs/1503.03832) to detect facial regions in the images and crop the region of interest. The steps to perform reverse visual search with this approach are:

1. Detect the Region of interest (ROI)
2. Crop the detected area.
3. Extract features from the cropped area.
4. Build the KNN using above features (embeddings).



Conclusion

We have successfully retrieved the similar images from LFW dataset, and we learned about many convolutional neural networks(ResNet, VGG, FaceNet, etc.), the difference between these networks and the thinking behind these networks, which would help us design our own neural network architectures in the future. We also learned about KNN, search, and most importantly, creative thinking to solve real world problems.

In future, we may improve our methods by hashing: KNN is slow and consumes high RAM / computing resources. It would have weaker performance if we use a larger dataset. We can improve this by hashing, for example, we can flatten all features and then hash the result array, so the features can be represented by a much smaller string, this way we can perform reverse visual search on a much bigger dataset, with better performance.

Possible application: since the method behind reverse visual search is similar, we can modify the above method, to make it suitable for more tasks, we can do reverse visual search on plants, buildings, and so on, by using a different neural network weights.

Similarity Search in Video

Like described in the requirement, we can do similarity search in video by examining per frame, but that is low efficiency, in this section, we designed a fast way to search in videos.

Overview

Filter the frame

We don’t want to calculate each frame, though calculating each frame is accurate and straight forward, it is slow, and contains much unnecessary work. We can filter the frame by:

1. Detect the ROI in each frame first, if the current frame doesn’t contain any ROI, we drop that frame.
2. Calculate the difference between the current frame and the last frame, if the difference is smaller than a pre-set threshold, drop the current frame.
3. The difference between two frames is defined as the Euclidean distance between corresponding pixel’s color in CIELAB color space.

Since we filter the frame first, we reduce the calculation, and because we keep the key frame, we keep the accuracy. Calculating the distance between the color of pixels in CIELAB color space is better than RGB color space based on experimental experience.

References

[1]: https://aws.amazon.com/blogs/machine-learning/building-a-visual-search-application-with-amazon-sagemaker-and-amazon-es/

[2]: https://scikit-learn.org/stable/modules/neighbors.html

[3]: https://hackernoon.com/6-best-open-source-projects-for-real-time-face-recognition-vr1w34x5

[4]: Florian Schroff, Dmitry Kalenichenko, James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering https://arxiv.org/abs/1503.03832