Reverse Image Search

Introduction to Artificial Intelligence, New York University

Prof. Pantelis Monogioudis

Spring 2022

Abstract

In this project, we have developed a system that can find images which are similar to the query image through Deep Learning models. Our system retrieves top 20 most relevant images from the dataset, given a particular query image. We investigate and extract various features from images through the dataset given to us. We then pass our features to our CNN model. And we can then try finding out the nearest images to our query image. We then try improving our performance with the help of Milvus (which is an open-source vector database built to power embedding similarity search and AI applications.) We compare the performance of our model between our baseline and then through our reverse image search improvement method. At the end, we have proposed some future works that can be explored to improve our reverse visual search system.

Introduction

The problem statement given to us is where we are given an image as a query, then we need to provide images that are similar and have correlation with the query image. The dataset that we have used is LFW(Labeled Faces in the Wild). This task comes very handy when we have a very huge database, and our users want to get some items that look similar with the one that they see. On the other hand, the task is also challenging as our hardware and software should be able to distinguish and find correlation between one image and the others. To be more clear, we have a large dataset given to us, and we have to computer similar images to the given image and for the second step we need to work on the improvement of the accuracy of the model. To start with the development of such a system, we need to understand how are we going to learn about images. If our algorithm understands how images look like, it can find out similar images. So, we have described each step we have taken for the making of such a system and what all software did we use for the same.

FIRST CHALLENGE – Reverse Image Search Baseline

This step uses a pretrained CNN model to extract features from the LFW dataset of images. A k-nearest-neighbours algorithm is used to determine which images in the data are most similar to an input image.

Libraries used –

Os , keras , tensorflow , random , numpy , matplotlib and scipy

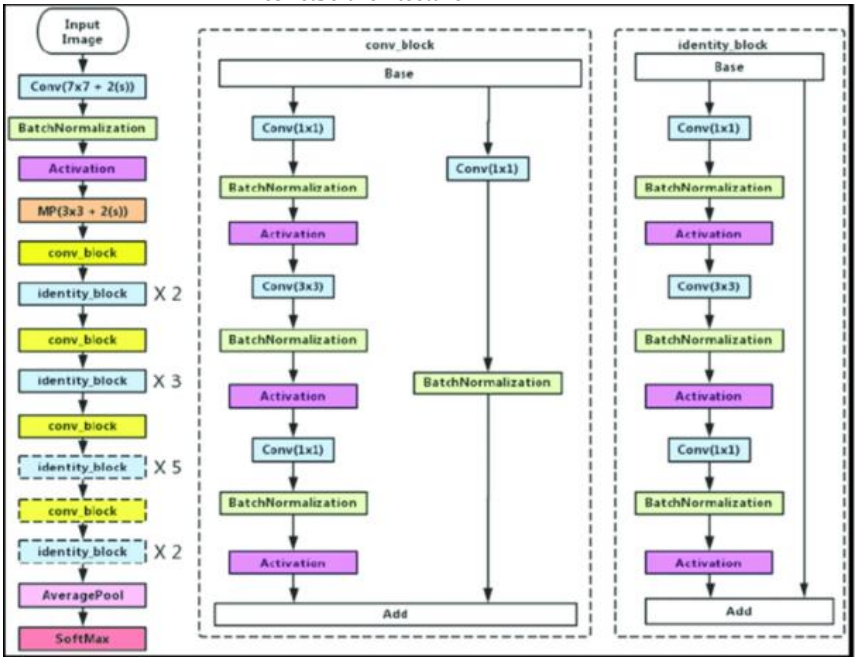
Methodology –

1) We firstly converted our images into numpy array. After this, it was given a label using the function model.predict(). The array was further converted into a tensorflow object, and then it was again passed through numpy library.
2) CNN Model - We are using a ResNet50 CNN from Tensorflow Keras pretrained on images from the ImageNet database. The ResNet50 model is a residual convolutional neural network, meaning it uses 'residual blocks' to improve accuracy and prevent degredation (due to vanishing/exploding gradients or

model performance) by utilizing skip connections. Skip connections enable the model to learn an identity function, ensuring accuracy across layers (regardless of how far down the gradient goes) and sets up shortcuts for the gradient to pass through (preventing its degredation). This means the model can include a large number of layers (in this case 50). The output of the CNN for each image is a one-dimensional array which is considered to be the embedded features of the image and is saved in an array. To use Resnet50 in the model, we used the following code –

```
model = tensorflow.keras.applications.ResNet50(weights='imagenet', include_top=True)
```

ResNet50 architecture



Credits - https://www.researchgate.net/figure/REsNet50-CNN-model-architecture_fig3_354517417

3) KNN The K-nearest-neighbours algorithm uses the euclidean distance between datapoints to determine the similarity between different data objects. The k 'nearest' objects are typically used to determine the 'class' of the input. In this case, the euclidean distance for the extracted features of images are calculated and the 'closest' images to the input are returned.

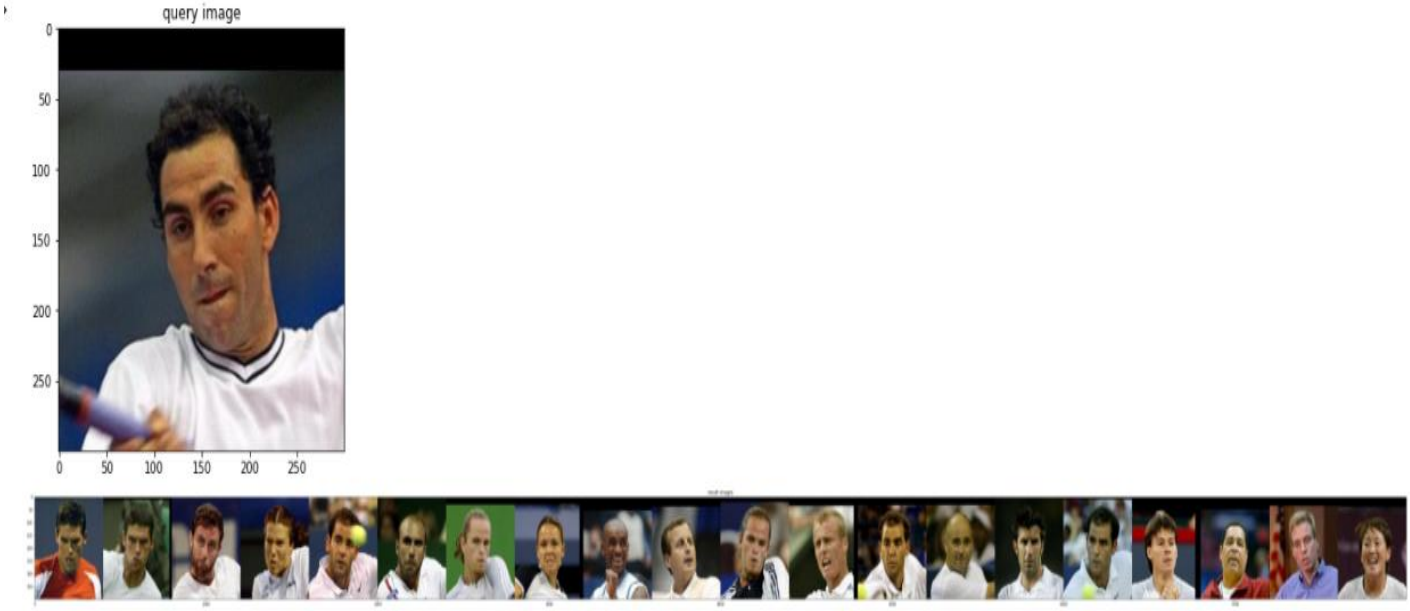
The equation works as follows:

$\sqrt{}\ d(p,qi)=\sum_{nj=1}(qij-pj)2$

For all image embeddings q0...qn in features where n is the number of features returned by the CNN from the feature extraction step.

The results are mapped to the index of the image embedding in features and sorted from smallest to largest value for d(p,qi). The first *output_size* indices in the sorted array are returned as the results.

Result -



We got a descent level of accuracy in the first step, where out of 20 result images, we have first few images similar to query images.

SECOND CHALLENGE – Reverse Image Search Improvement

For the second step, we used MTCNN algorithm along with Milvus Search for the betterment of our results. Here, we use MTCNN in order to extract the facial features from a photo before feeding it into the CNN. MTCNN is a CNN-based classifier which is able to detect the presence of a face in an image and provide the location of that face. We use the location MTCNN returns to crop the image around the detected fase. Using a cropped image allows the ResNet50 CNN to extract only those features which make up the person's face, meaning our results will be based on only n the features of a face and not the environment in the image around the face. At the end, we are using Milvus database which is built to power embedding similarity search. It optimizes the ability to query an embedding and retrieve similar results.

Libraries used - PIL , MTCNN , torch , torchvision , torch.utils.data , os, sys , time

Background Information –

1) MTCNN - Multi-task Convolutional Neural Network.
2) CNN Model – We are using pre-trained CNN model, ResNet -50. ResNet-50 is a convolutional network that is 50 layers deep (48 Convolutional Layers along with 1 maxPool and 1 Average Pool layer). A residual neural network (ResNet) is an artificial neural network of a kind tht stacks residual blocks on top of each other to form a network. We are loading a pre-trained version of the network on more than a million images from ImageNet database.
3) Milvus – Milvus is an open source vector database. It helps us in storing, indexing and managing massive embedding vectors generated by deep neural networks and other machine learning models. It is basically a database, that helps to design queries over input vectors, and it is capabale of indexing

vectors on a large scale.

4) To run Milvus locally, we used the Docker-compose software.

Methodology –

1) We firstly made a Utlities directory, which contains all the important functions that we will be using for this step.
   a) The first utility module made is utilities.image_to_vec. This python file will help us to get embdeddings through ResNet50 model for the entire directory. It even searches for images in the subdirectories. It takes two parameters- first ; path like string which points towards the directory in the file system, second; extension in the form of string. Default extension for images has been set to "JPEG". It returns a numpy array with all the generated embeddings and a list with respect to file paths for the processed images.
   b) The second utility module helps us in getting similar images from the milvus server. It takes in input in the form of an image through a given file path name and plots top 20 similar images according to query results.
   c) The third utility module, which is utilities.milvus_utilities, which creates the milvus collection, transfers all the embeddings to milvus collection along with ids and downloads the top nearest neighbours.

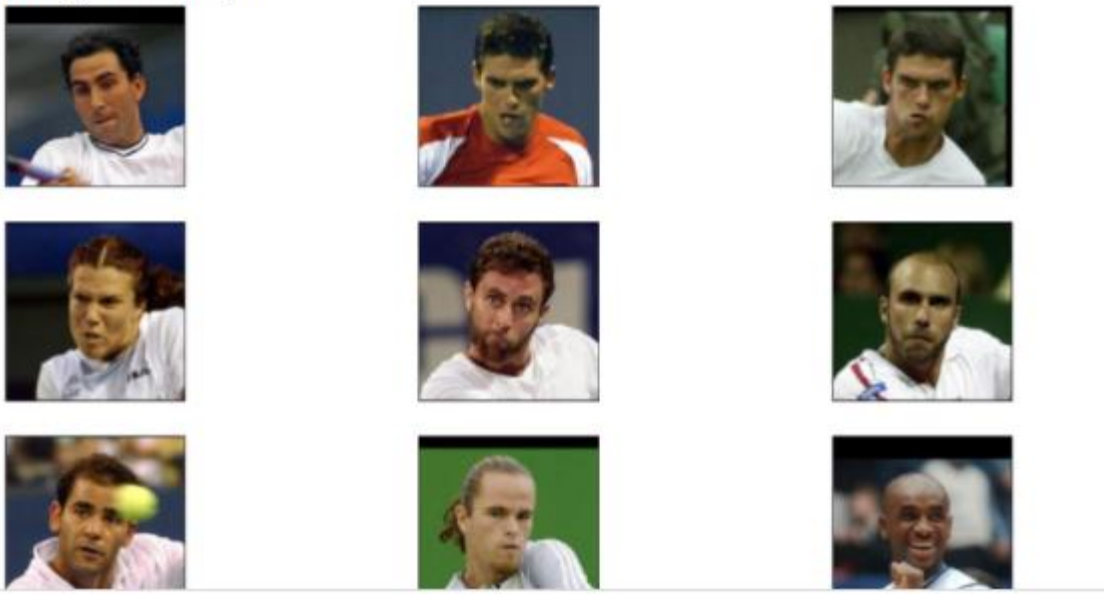
2) In our final notebook, we used all of the above functions to put in a query image and extract top 20 similar images.
   a) We first created an embedding generation model based on the ResNet-50 pre-trained model.
   b) We then generated embeddings for all the train set of ImageNet images.
   c) We created a milvus collection and uploaded the generated embeddings.
   d) We completed the process of extraction of similar images.

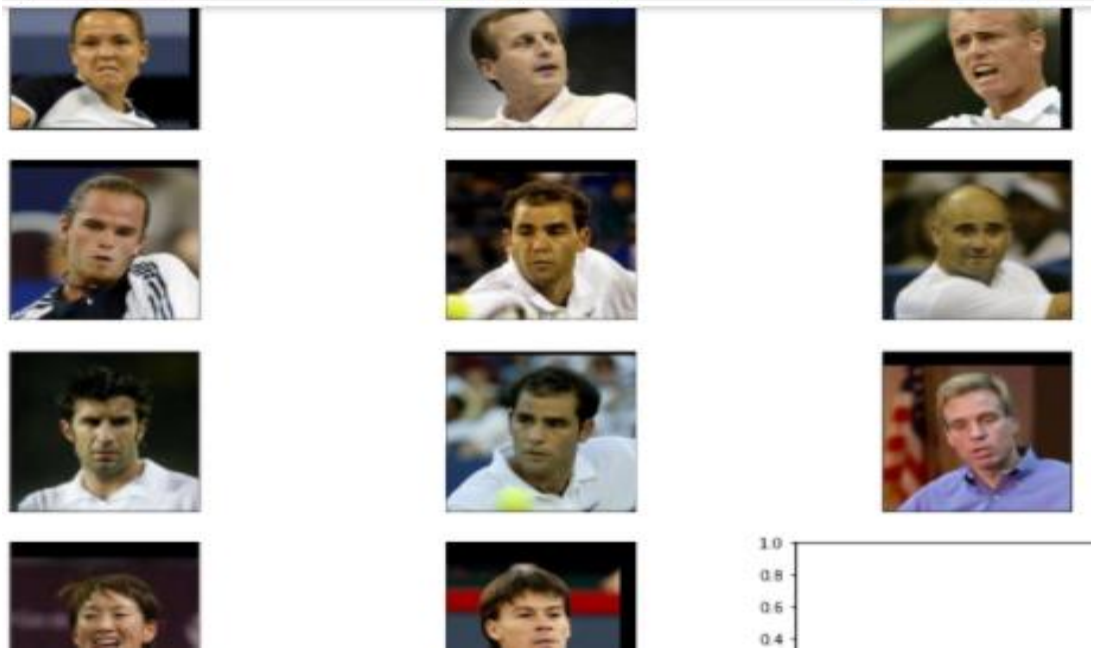
3) To sum up, the image_to_vec.py class loads the ResNet50 and removes the last layer, so that we can get last average pooling layer output. The output which we will get will be in the form of embedding and are equivalent to deep feature, along with the path of the image file. All this information has to be stored in the form of a database. We are storing it locally, in the form of CSV file. After this, we pulled Milvus docker container and connected our milvus server. Once our Milvus server is running, we can upload our ImageNet embeddings. The insert_embeddings function detects the size of the array and insert embeddings in chunks to avoid error raised by the milvus server. This gives us ids for the uploaded vector and then we can further attach these ids to our earlier generated CSV file. Now, our code is ready. We just need to plug-in the path of a particular image, whose nearest neighbours we are interested to find.

Result –

Plotting original image:

Plotting similar images:





THIRD CHALLENGE – Reverse Video Search

For the third challenge, which focuses towards making of reverse video search where we will be demonstrating that our designed code is able to produce videos containing a person of interest from a query video(which is obviously going to contain the person of interest).

Libraries used –

Os , keras , tensorflow , random , numpy , matplotlib and scipy

Approach Followed –

We would randomly select a set of frames from each video. It's not necessary to run the CNN over every frame since it should be able to identify a person, even with one frame. We would select around 5 frames for each video and run these through the CNN before averaging over the results of those frames. We got the features in the form of array, on which we then applied dimensionality reduction technique to reduce the dimension (as they are very big in size). These results could be stored into memory with Milvus. A query image could then be used to search the stored data. First we would take around 5 frames from the query image and run these through the CNN, just as we did with the input. The results form the CNN would be averaged over. Finally, we would use milvus, as outlined in the previous section, to look up the most similar feature vectors and return the videos associated with each.

In the actual notebook included here, we were not able to download, retrieve, store and work with the video files which came in a 5gb tar file. Instead we give present some code that uses a csv of the youtube video data and performs analysis on the attributes of the files. The similarity among all attributes is compared with those of the input in order ot determine the most similar results. Along with the explanation of how we would have liked to implement the video search program, the notebook is meant to give outline our thoughts and intentions for the extra credit.

Future Work

In our project, we mainly dealt with MTCNN, CNN model (ResNet50), algorithm like KNN and Milvus search. Besides the methods, we mentioned, there are still a lot of rooms for further experiments. Some qualified techniques, specifically for image reverse search task, are summarized below –

1) For first challenge, we could have experimented with CNN model ResNet 152, which has more layers than Resnet50
2) For the second challenge, we could have tried combination of facenet and elastic search, or a combination of deep face and milvus.

References

1) https://pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/
2) https://pro.europeana.eu/post/image-similarity-search-demo
3) https://medium.com/analytics-vidhya/image-similarity-model-6b89a22e2f1a
4) https://blog.milvus.io/frustrated-with-new-data-our-vector-database-can-help-e5c430b29be7
5) https://blog.milvus.io/the-journey-to-optimize-billion-scale-image-search-part-1-a270c519246d
6) https://zilliz.com/learn/supercharged-semantic-similarity-search-in-production
7) https://zilliz.com/blog/building-a-search-by-image-shopping-experience-with-vova-and-milvus
8) https://milvus.io/docs/overview.md