Reverse Image Search - AI Project - Spring 2022

Aiswarya Sriram[1], Anshul Agrawal[1], and Sourabh Kumar Bhattacharjee[1]

New York University, NY {as14988, ava8249, skb5275}@nyu.edu

Abstract. This project performs Reverse Visual Search on images using the LFW dataset. Given a query image(person of interest), we generate 20 similar images. This paper addresses 2 methods of Visual Search. The baseline model uses a ResNet-50 architecture followed by k-Nearest Neighbors. The improved model uses MT-CNN + FaceNet model and the similarity search is done with k-Nearest Neighbors(KNN). We empirically observe that the improved model which generates feature vectors using just the faces detected from the images performs much better than the baseline.

Keywords: Reverse Visual Search, FaceNet, KNN, KMeans, ResNet, MT-CNN

1 Introduction

Face recognition is a technique of identification or verification of a person using their faces through an image or a video. It captures, analyzes, and compares patterns based on the person's facial features. The face detection process is an essential step as it detects and locates human faces in images and videos. This project focuses on images. Given a query image, the aim of our project is to return 20 similar images of which a good number should be images of the same person given in the query.
The reverse Visual Search System would have the following steps:

1. Face detection
2. Feature Extraction - to get embeddings
3. Similarity Search - to find similar faces to a given query image

In this project, we will be using the lfw dataset. [3] The project is divided into 2 parts

1. **Baseline**: In this part we try to build a basic model that uses a resnet50 architecture to extract features from every face. These feature embeddings are then sent to a similarity search algorithm. Here, we use Kmeans clustering following by KNN. Then, with the generated vectors, we input a query image and return 20 similar images. We do this for 10 query images. Upon looking at the results , we see that while some of the faces are similar or yield the same person, the overall accuracy is low. We need to improve upon this model so we introduce a different architecture outlined below.

2. **Improvement**: For the improved architecture, we first preprocess the dataset. Instead of using the entire image provided which often has background details (e.g. A tennis star Albert Costa's face also has a tennis racquet and a green background), we extract only the face from it. This face extraction is done using the MTCNN method. After this the transformed dataset is passed to the facenet Keras model. It is a deep learning approach and is one of the Industrial Giant Nets. Once we obtain the feature embedding vector, we again use Kmeans and KNN for the similarity search. We then pass query images and predict similar faces from the query image. It is seen that this model performs vastly better than the Baseline.

The details are outlined in the below sections.

2 Related Work

Some of the work we referred to while researching the best ways to perform visual search were Amazon's guide to building a visual search application with Amazon SageMaker and Amazon ElasticSearch [5]. This however required very heavy compute power which was not available to us.
We referred to medium articles and blogs to help us understand the image similarity model and the most optimal similarity search techniques.

For the image similarity search we used KNN and Kmeans with reference from Analytics Vidhya Community [6]. We referred to [2] and [8] for face detection and extraction.

3 Data

The labeled faces in the Wild dataset, known as the LFW dataset [3] is a public benchmark for face verification. The dataset we used contained 13430 images with 5749 classes. Each class is a folder labelled with a person's name. Within the folder, there are 1 to many different photos of the same person. Thus, each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.

4 Methods: Overview of Image Search and Face Detection

4.1 Feature Extraction in Baseline model

The input query given by a user will be an image. The image can be represented as a matrix of pixels. Running similarity search on the entire image is not a good idea for 2 important reasons.

Firstly, all the pixels in the image are not very useful in identifying the facial features which help in distinguishing a face from another or finding similarity.

For example: The forehead of person's face doesn't give a lot of information about the features of his/her face to identify similar faces.

Secondly, the input image can have varied size. In the cases where it is a high resolution image we will have to do a lot of processing. For example: an image with 512X512 resolution will have 262144 input features! Also, blindly resizing them to a smaller size can be dangerous because, it might lead to losing of important features of the face.

We therefore use a pre-processing step, to identify the important features in the given image using the below method and then

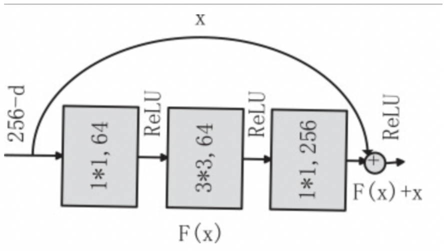


Fig. 1: Illustration of Skip Connection in Resnet [4]

ResNet-50 We used a deep residual network ResNet-50 [4] to extract the important features of the face and store it in a vector of size 8192. This essentially means that we are extracting 8192 important features from the face and then using this to predict similar faces.

The problem with a deep fully connected neural network is exploding gradient because of a large number of layers in it. In order to overcome this issue, we use Resnet-50 which provides options to skip layers in the middle as shown in the Figure 1. These are called **skip connections**. This helps us use a deep residual network of 50 layers too. We then do an ensemble learning based on the values received from all the connections and then decide the output.

There is a fully connected layer at the end of the resnet model, to summarize the features at the end. We then flatten the output to a single dimensional vector. The activation function used is Relu. More nodes are added at the end of the network using the dense method, to highlight the important features and get the desired vector size of 8192.

We first train the described model on the LFW dataset [3], to be able to extract the important features from a given image. Then, on receiving an input image, we feed into the trained model to retrieve the important features from it. This is passed on to the similarity search method in the pipeline.

4.2 Feature Extraction in Improved model

Now in order to improve upon the features obtained in our baseline model above, we experimented with the *FaceNet* [7] model to extract features.

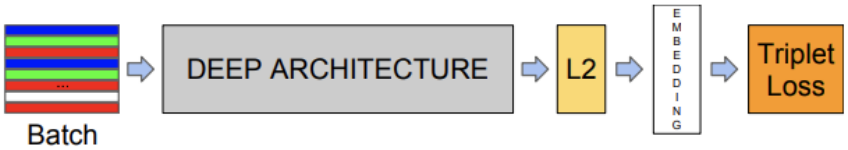


Fig. 2: Overall FaceNet Architecture [7]

As can be seen in Figure 2, *FaceNet* uses a Deep Convolutional Neural Network, along with L-2 Normalization to obtain a d-dimensional embedding vector, which we end up storing in our vector database for finding images with similar embedding vectors. The primary reason why *FaceNet* performs much better than ResNet-50 is due to the underlying loss function that is used during pre-training, i.e. the *Triplet Loss*.

The concept of *Triplet Loss* [1] is derived from a class of neural network architectures known as Siamese Networks. These networks use the same weights in tandem for a pair of images to compute similar image vectors. A 'triplet' is defined on an 'anchor' image (i.e. the image whose vector we want to obtain), a positive image (i.e. an image similar to the 'anchor'), and a negative image (i.e. an image which is dissimilar to the 'anchor'). The *Triplet Loss'* objective for an 'anchor' image is to maximize the euclidean distance between the vectors of the 'anchor' and the 'negative' images, and minimize the euclidean distance between the vectors of the 'anchor' and the 'positive' images.

Formally, suppose we have an embedding vector-function $f(x) \in R^d$, where x is an input image and d is the dimension of the vector in euclidean space, and we constrain it such that $||f(x)|| = 1$. We define an 'anchor' image x_i^a, a 'positive' similar image x_i^p and a 'negative' dissimilar image x_i^n. The objective is thus defined as:

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2 \tag{1}$$

$\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in T$, where T is the set of all Triplets and α is a margin enforced between the positive and the negative pairs. Therefore the *Triplet Loss* which is to be minimized is written as:

$$\sum_i \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha \right] \tag{2}$$

We further attempt to improve the quality of the *FaceNet* features by generating features on just the faces of the images. In order to do so we made use of the MTCNN [9] pre-trained network. This is a network inspired by the independent strides made in the area of *Face Detection* and *Alignment* and this network was the seminal attempt to exploit the correlation of these two tasks. In this network, an initial P-Net (or Proposal Network) is trained to 'propose' all candidate bounding boxes, and they are fed into an R-Net (or Refine Network) which essentially performs *Non-Maximal Suppression* on these windows to obtain a candidate (by merging highly overlapped candidates) and five facial landmark points.

We hypothesized that if we used the MTCNN generated face images and obtained the resultant feature vectors using *FaceNet* then the quality of feature vectors might be more suited towards obtaining images with *similar* faces, rather than get bogged down by the irrelevant noise present in the rest of the query image. Further we also hypothesize that these feature vectors would be able to better discriminate between dissimilar images.

MTCNN+FaceNet To generate feature vectors, we used a pre-trained *MTCNN* model to get 160 X 160 dimension cropped face images of the entire dataset, and then we used a pre-trained *FaceNet* model to generate 128-dimension feature vectors for each image, and stored them in a vector database (Pandas DataFrame in our case). We then subsequently used a similarity search algorithm (as described below 4.3), to obtain *similar* images or, more formally, images with *similar* feature vectors.

4.3 Finding similar faces

Now, that we have the features extracted from the input query, we need to find faces with similar features from our LFW dataset [3]

We used **k-nearest neighbours** method to achieve this. However, kNN requires labeled data, but we have unlabeled data. This is because the problem is "similarity" finding and not "exact" finding of faces.

To solve this problem, we first do a **k-means clustering** on the feature vectors of all the images to group them into 6 clusters. The choice of number of clusters is through trial and error, to get the best accuracy and performance time for kNN method.

This provides us with labeled data, and we then run the knn method on this cluster to get the 20 nearest neighbours on the query image's feature vector.

We then display the images associated with the 20 feature vectors retrieved in the above step.

One, thing to note, is that the similarity finding methodology is the same in both baseline and improved model. This is because the face extraction with facenet worked out so well to give an exhaustive list of images of the "same person" from the dataset on an input query.

Had we encountered accuracy issues, we could have used methods like Expectation-Maximization, or Principal Component Analysis in the preprocessing part.

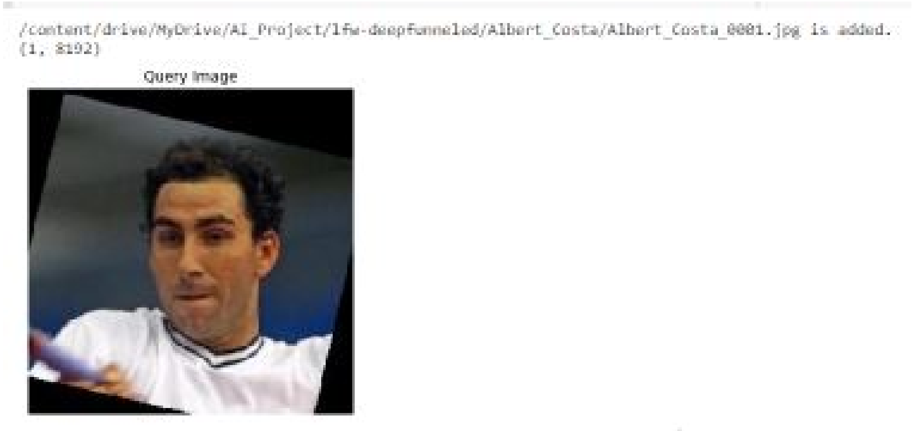
Similarly, in case of performance issues, we could have have used elastic kNN or Milvus to search for similar embeddings in the dataset.

5 Environment Setup

Our entire project was done on 2 Google Colab notebooks, one for the baseline model and one for the improved model. Google colab runtime type was set to GPU. Google Colab Pro was used to run the improved model since the Facenet model requires higher compute power and longer runtime. We used Google drive to store the dataset, processed dataset, model weights (feature embeddings), kmeans model and the knn model. To re execute our experiments, follow the instructions mentioned in the README of the project GitHub repository. The code will create the appropriate folder directories and sub-directories in Google Drive that are necessary for execution.

6 Experiments and Results

6.1 Result from Baseline



```
/content/drive/MyDrive/AI_Project/lfw-deepfunneled/Albert_Costa/Albert_Costa_0001.jpg is added.
(1, 8192)
```

Fig. 3: Albert Costa- Query image for baseline

Inference We can see that certain features from the face and background of the input query were given more importance in finding similar images. For example, we can see images of people holding rackets, or similar facial expressions etc. For the baseline however, we did get one image of the same person in the results.

This helps us infer that the background noise is causing reduction in search accuracy and also more importance is being given to certain features and certain features are being ignored completely while predicting.

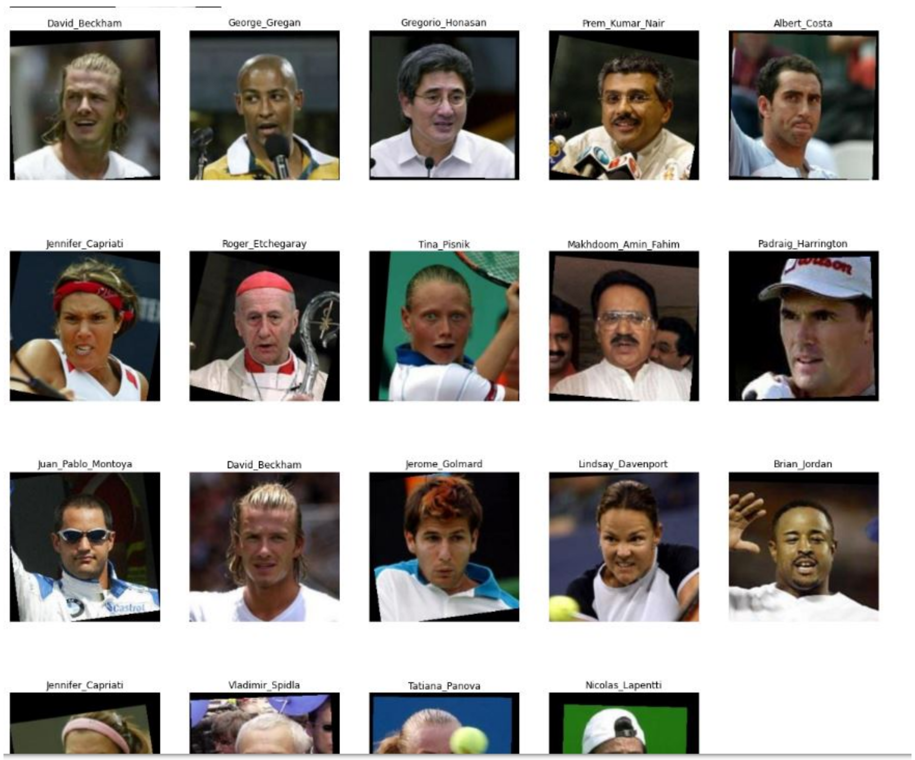


Fig. 4: Albert Costa- Result of the baseline model

```
WARNING:tensorflow:No training configurat
```

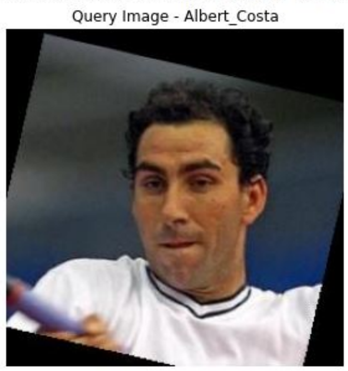


Fig. 5: Albert Costa- Query image of improved model

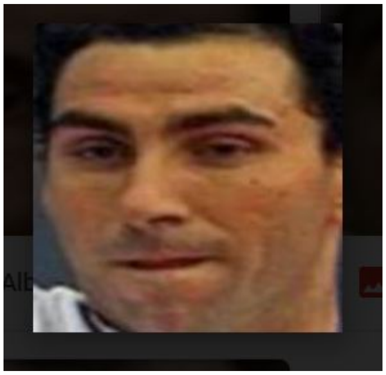


Fig. 6: Albert Costa- Face extraction

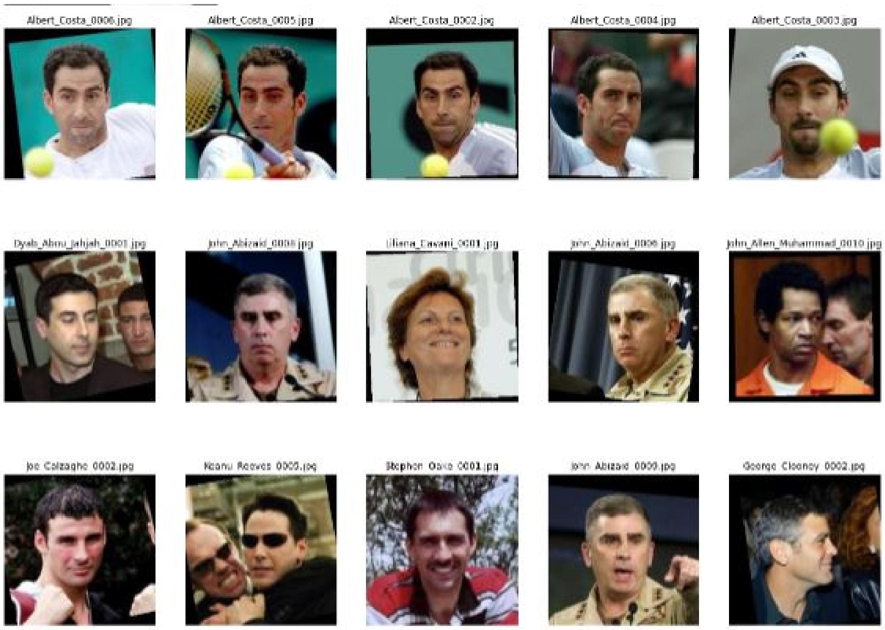


Fig. 7: Albert Costa- Result of improved model

6.2 Result from Improved model

Inference Now, we see that we got all the images of the queried person from our dataset, and also got images of people with reasonably similar face as the query image. These results hold for all of the 10 query images and can be seen in the last cell of the Colab notebook.

7 Conclusions

We see that the improved model using the Facenet architecture works much better than the baseline model due to deep learning techniques and preprocessing of the dataset to purely extract the faces. Our hypothesis is also empirically validated that feature vectors constructed from purely faces are less noisy, better suited for similarity tasks, and have better discriminant behavior towards dissimilar images.

References

[1] Xingping Dong and Jianbing Shen. “Triplet Loss in Siamese Network for Object Tracking”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[2] Saina Ghosh. *Face Detection*. 2020. URL: `https://medium.com/clique-org/how-to-create-a-face-recognition-model-using-facenet-keras-fd65c0b092f1`.

[3] *Labelled Faces in the Wild (LFW) Dataset*. UMass. DOI: `http://vis-www.cs.umass.edu/lfw/`.

[4] Bin Li and Dimas Lima. *Facial expression recognition via ResNet-50*. 2021. URL: `https://www.sciencedirect.com/science/article/pii/S2666307421000073`.

[5] Amit Mukherjee and Laith Al-Saadoon. *Amazon Sagemaker*. 2020. URL: `https://aws.amazon.com/blogs/machine-learning/building-a-visual-search-application-with-amazon-sagemaker-and-amazon-es/`.

[6] Chaitanya Narva. *Image Similarity Model-KNN*. 2020. URL: `https://medium.com/analytics-vidhya/image-similarity-model-6b89a22e2f1a`.

[7] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823. DOI: `10.1109/CVPR.2015.7298682`.

[8] vishal462. *FaceNet Face Extraction*. 2021. URL: `https://www.analyticsvidhya.com/blog/2021/06/face-detection-and-recognition-capable-of-beating-humans-using-facenet/`.

[9] Kaipeng Zhang et al. “Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks”. In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503. DOI: `10.1109/LSP.2016.2603342`.