New York University Tandon School of Engineering

CS-GY-6613

Prof. Pantelis Monogioudis

PROJECT REPORT

Reverse Visual Search

Group 9

SUBMITTED BY:

NAME	NET ID	N NUMBER
Eashan Kaushik	ek3575	N19320245
Rishab Singh	rs7623	N18032325
Rohan Patel	rp3617	N13328358

TABLE OF CONTENT

PROBLEM STATEMENT:

We have all played the game of "spot the difference" in which we need to find differences between two similar images. To build upon the context, can you find images that are similar to a given image? The google reverse image search is an apt description of what we are trying to building in this project. Our problem statement is to find N similar images, given an input image.

1. INTRODUCTION:

As humans, we are fairly good at comparing an image and finding similar images just by looking at an image. But what if we need to find similar images from a pool of thousands of images? This is where state of the art Deep Learning techniques can help us. To be a bit more specific, we can utilize the concepts from computer vision, and deep learning to build a model that can implement this task for us and can provide results within minutes.

Unlike the other machine learning project that is based on metadata such as keywords, tags, or description, we are going to build our project on the 'content based' which means that we are going to extract meaningful features from the images to achieve our goal of finding similar images. This is a common in a real-life-based scenario in which we are only given an image and we do not have the corresponding meta-data associated with it. Fortunately, the recent advances in the field of computer vision and deep learning will help us despite this shortcoming.

2. DATASET

In this project, we are going to utilize the LFW Dataset (Labelled Faces in the Wild), a compilation of face pairs of positively and negatively weighted images samples collected by the researchers of the University of Massachusetts. The dataset comprises 13233 images of 5749 people with 1680 people having two or more images. LFW Dataset is the benchmark for pair matching, also known as face verification. For our project, we are going to build our model on this dataset and later utilize this model for a reverse visual search on a given face image.

Figure 1 Sample Data

3. ARCHITECTURE

In this section we will discuss the architecture that we have used to build this model.

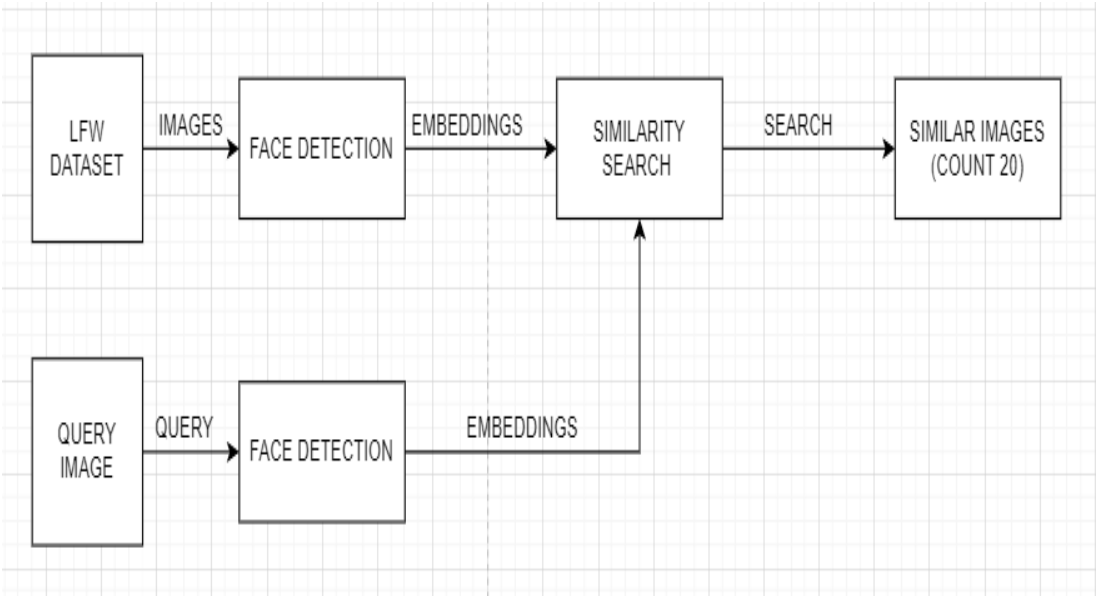

Figure 2 General Architecture

Steps for reverse visual search:

1. Generating the embeddings for entire dataset.
2. Storing these embeddings in a vector database.
3. Generating the embedding for query image.
4. Searching the 20 closest neighbors in the vector database.
5. Giving the results.

For the purpose of developing a model from ground up we have developed three different architectures and tried to get better results in each iteration. Different models are discussed below:

1. Developing a **Baseline Model**: In this model we have used ResNet50 for generating the embeddings of an image and providing it as an input to searching ML model, in this case K-Nearest Neighbor.
2. **Initial Improvement** on Baseline Performance: Utilizing MTCNN Face detection to extract target images (faces) and utilizing ResNet50 along with K- Nearest Neighbor (similar to the baseline model) for visual search.
3. **Final Improvement** on Baseline Performance: Utilize MTCNN Face detection to extract target faces from the image, generating embeddings using FaceNet and finally using Milvus to search for the similar images.

3.1. Baseline Model

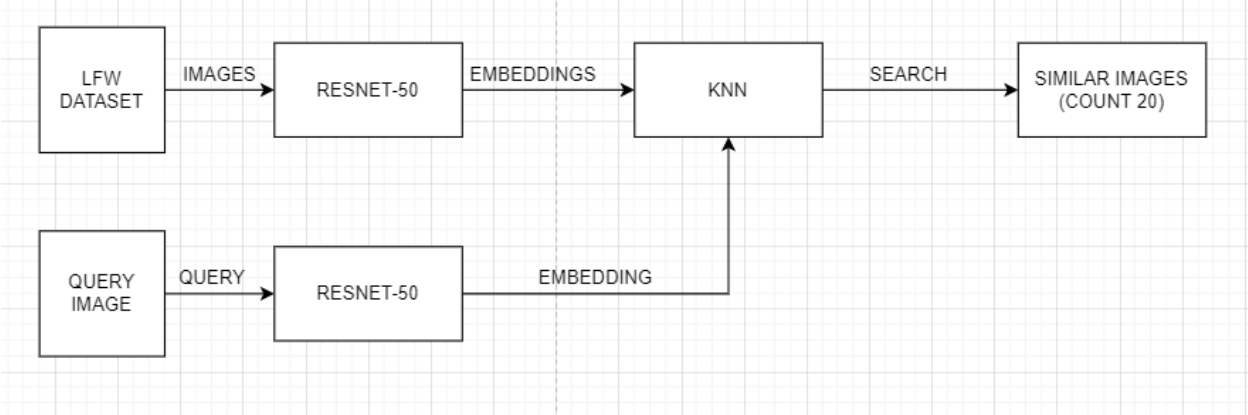

Figure 3 Baseline Architecture

Before we dive deeper into finding the embeddings on an image, let's understand what embeddings are and how it helps us. When we are dealing with images, the machine interprets an image in the form N x N matrix with each cell containing the pixel value of the image.

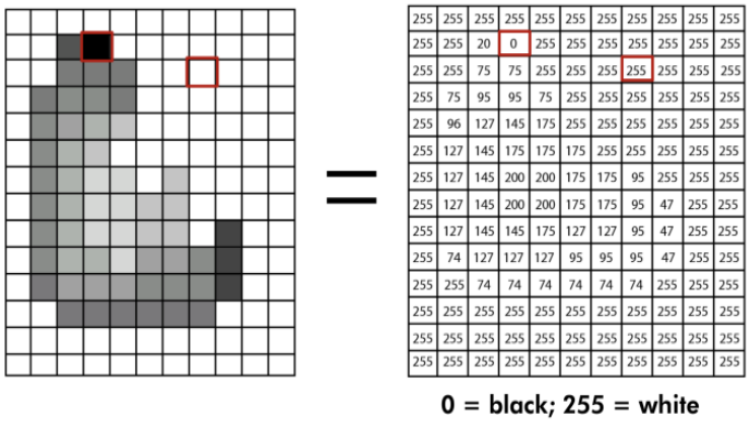

Figure 4 Image Gradient Display

To find the similarity between two images, one of the ways to go about it is by using a distance measure. Although this method works pretty well to find similarities in the images, it fails to account for the different variations such as size, rotation, etc. of the same image. To account for this, we have various classical descriptors (SIFT, SURF) that help us to extract the salient features of a given image. This method has been widely used in the past decade for hand-crafted content-based image retrieval. However, with the recent developments in deep neural networks, especially convolutional neural networks, we can achieve the state-of-the-art results in classification problems. These same CNN layers can be utilized to generate a feature vector that extracts features from an image, invariant to its geometrical transformation and instance. These feature vectors

contain the key points of the image from our embedding. Embeddings are vectors that can be painted on the cartesian plane representing an image. Hence, in our project, we are going to utilize ResNet50 to generate these key point feature vectors that will be later utilized by our machine learning model to generate similar images.

ResNet50 (Residual Network): ResNet50 was an improvement over the AlexNet which helped in training an extremely deep neural network by solving the problem of vanishing gradient. The deep neural network here means that we can achieve great results even if our model has hundreds of fully connected layers in the network. But we will not dive deeper into this as we are concerned with the feature vector that is generated by the convolutional layers of this model. In short, the ResNet50 model is used to find the embeddings of an image in our project. Let's see a bit more into the ResNet50 Architecture to understand what is happening behind the scenes and how we are going to get the embeddings.

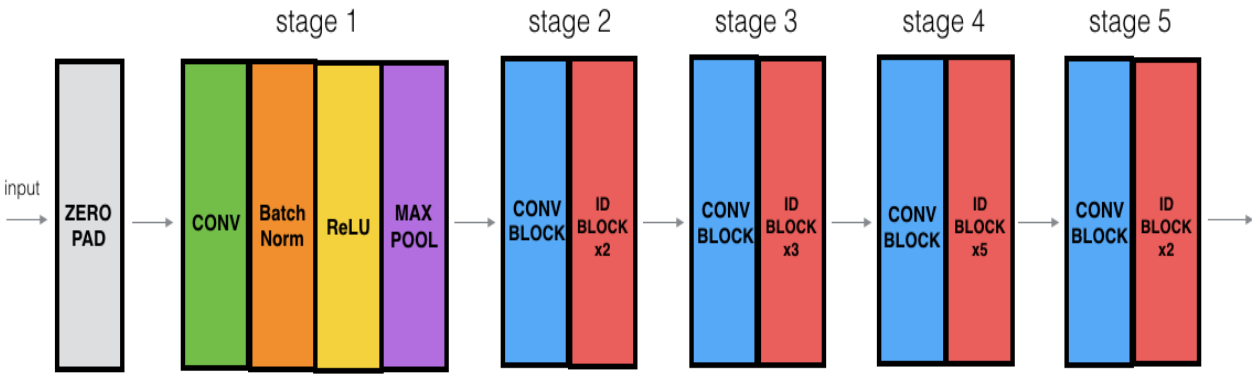

Figure 5 ResNet- 50 Convolutional Layer

The model consists of 5 stages with each convolutional block consisting of three convolutional layers (convolution + Batch Normalization + ReLU) and each identity block containing 3 convolutional layers (Convolution + Batch Normalization). The identity block here is what was introduced by the authors of ResNet50. This block helps us to perform 'skip connections. Skip Connection here refers to adding the original input to the output of the convolutional block. This is what helps ResNet50 help to deal with the problem of vanishing/exploding gradient problems.

In our project, we obtain the embeddings/feature vector of size **2048** at the end of stage 5 by adding an **Average Pooling layer.** Once we obtained this feature vector, we now pass it through our machine learning model to train it. The model that we used for baseline performance is K- Nearest Neighbors.

K- Nearest Neighbors (KNN): This algorithm is an easy-to-implement supervised machine learning algorithm that will help us to find images that most closely resemble our given search query based on the distance. We utilize this algorithm from the sklearn library.

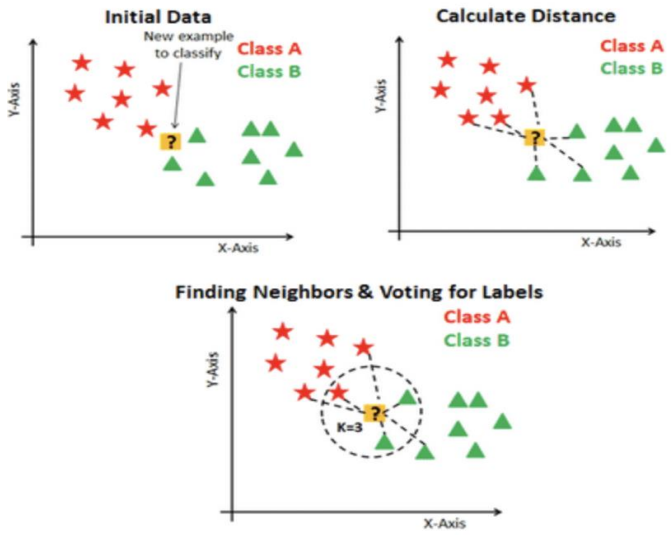

Figure 6 KNN Working

Results:

The result here shows the similarity search done by our model for Carmen Electra. Just for comparison purposes between models we have used accuracy to understand the performance of the model. In the case of Baseline, we have received an accuracy of 9%.

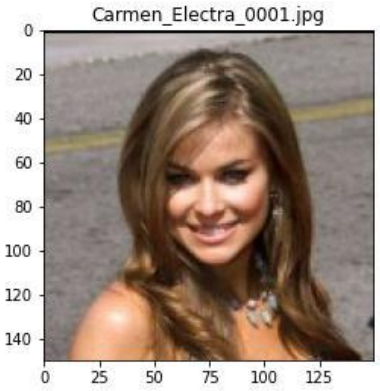

Figure 7 Carmen Electra

Figure 8 Baseline model result 20 similar faces for Carmen Electra

Other output results for baseline can be seen here: [Drive]

As we can see our model couldn't classify properly and we could only find 1 out of the 6 Albert Costa images. Hence, we move forward to our initial improvement.

3.2. Initial Improvement on Baseline Performance

When we check the LFW Dataset we see that the images in this dataset have a lot of noise in their background, in the form of unknown faces, objects, etc. So, the embeddings that we found earlier take into account the noise associated with a given image. To improve our image search query, one of the ways we adopted is using MTCNN to help us extract the relevant information in the form of face extraction and provide this through our base model.

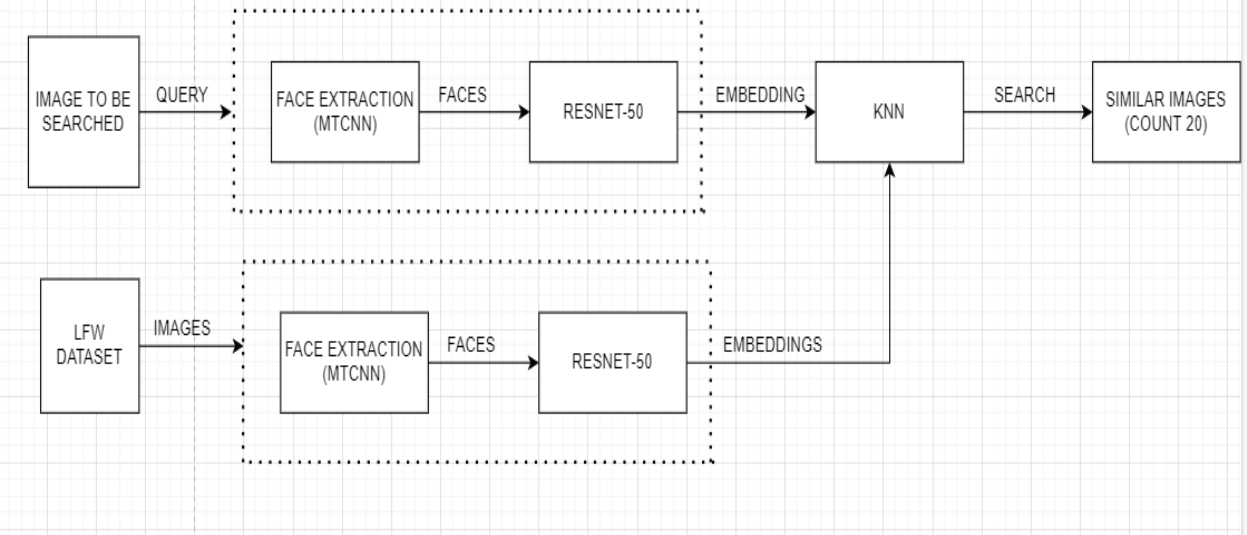

Figure 9: Architecture Design for Initial Improvement

MTCNN (Multi Tasked Cascaded Convolutional Neural Network): This is a face detection model that was based on the paper by Kaipeng Zhang. The paper describes a deep cascaded multi-task framework that takes features from different sub-models, namely P- Net, R-Net, O-Net), to boost their correlating strengths. This model works by generating a pyramid of images for a single (to account for different sizes of a face in an image in the form of multiple scaled copies). These pyramids of images are then fed to the P-Net which then gives us the bounding boxes over the faces. To refine this, the model removes the bounding boxes of low confidence level and performs non-maximum to give us the final output. This output is then again fed to R- Net and O-net with the same aforementioned procedures. The output of this model is an image with a bounding box over the face.

Figure 10: MTCNN input and output

Other output for MTCNN can be seen here: Drive

We use these bounding boxes to crop our LFW Dataset. The K-NN model is then again trained on the embeddings based on the faces of the LFW Dataset. To search a query, we pass the input image through face extraction and embedding layer as shown in figure3.

Results:

In this case we received an accuracy of 12%. The increase in accuracy does not look like much but here we can infer that, extracting target faces from the images gives us better results in the end. Moving forward we will keep this in mind, and include MTCNN in our preprocessing stage. Hence, we move forward to our second and final improvement.

3.3. Final Improvement on Baseline Performance

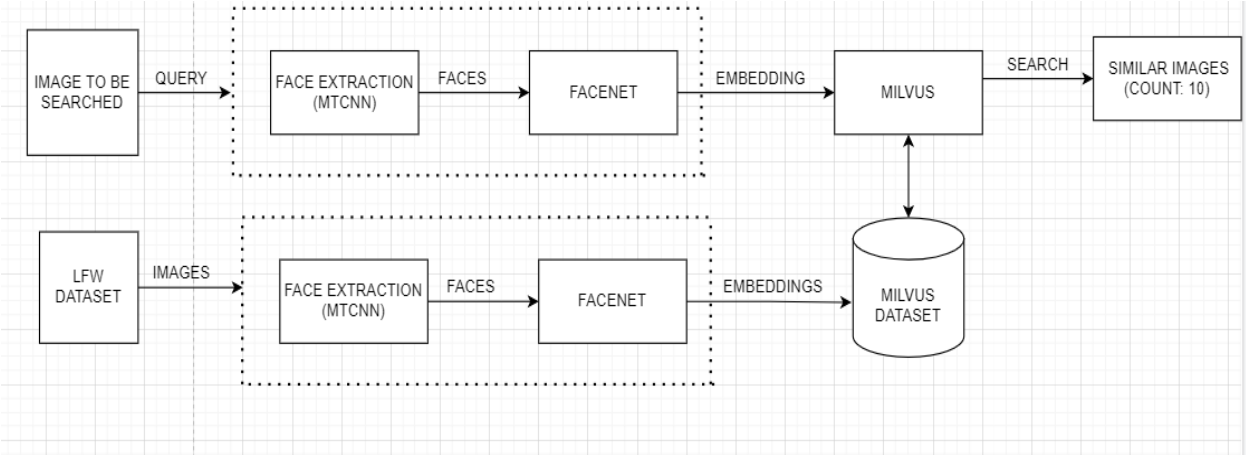

Figure 11 Architecture Design for Final Improvement

The results from the previous method didn't give us satisfactory accuracy. Therefore, for our final improvisation, we utilize MTCNN, FaceNet, and Milvus. Now in the previous improvement, we utilized MTCNN to extract faces from the image and find the embedding based on these new images. However, in this improvisation, we are going to pass these extracted face images to FaceNet.

Before we begin talking about FaceNet, let's understand problem in hand. The data set has a very few samples for a particular class. These causes data scarcity and an imbalances class. Along with this, adding new classes of data gets difficult as the neural network has already learned features for certain number of classes. Now adding another class will require us to re-train the network and while dealing with real time data, it is computationally really expensive to retrain the model frequently. Thus, instead of learning the new parameters again, we have something called Siamese network that learns based on similarity function. Not only can we train it to check if two images are similar but also enables us to classify new classes of data without training the network.

Siamese Network- Siamese Network is Neural Network Architecture in which two neural networks, called sister network, are identical to each other. Siamese networks work by finding the distance between the two images to be compared. With this intuition in mind, if two images are of same label, they should be closer to each other. Hence, the neural networks should learn in such a way that the distance between the image embeddings is less for similar images, and

large for unsimilar images. The distance between these two embeddings is called similarity score. There are two approaches – direct and indirect, to validate the model. FaceNet uses the direct approach.

FaceNet: This is a deep neural network that is used to extract salient features from an image of a face. FaceNet has adopted two types of CNN in its architecture namely, Zeiler & Fergus architecture and GoogLeNet Style Inception Model. It works with the help of a special loss function, called the triplet loss. To measure this loss, we use three images, namely anchor, positive and negative. The purpose of the triplet loss is to: -

1. minimize the distance (L-2 Distance) between the anchor (target image) and negative image
2. maximize the distance (L-2 Distance) between the anchor (target image) and positive image

The above intuition to finding embeddings in such a way that reduces the triplet loss. The final embedding is of size 1792 which is formed by flattening the feature vector using Average Pooling Layer, in our project. .

Milvus: Another change that we will have from our previous approach is that we will use Milvus instead of KNN. Milvus is an open-source vector database that is used to build a reverse image search system, for our project. For our project, we have used 'IVF_FLAT' as our index type and Euclidean Distance as the similarity metrics. Our embeddings (of size 256) for the LFW Dataset are sent to the Milvus Database (cluster). For the query search, we pass the image through MTCNN and FaceNet again to fetch the embedding layer. This layer is then sent to Milvus which is compared against the data in the cluster to find a similar image.

Results:

We see that all the images of Carmen Electra have been found by our model. Furthermore, all the images obtained are fairly similar to Carmen Electra. Thus, we can see the improvement over our previous approach towards Reverse Image Search. In this case we received an accuracy of 62%. This is a huge development, and the results are very good as well.

All the output results for final improvement can be seen here: Drive

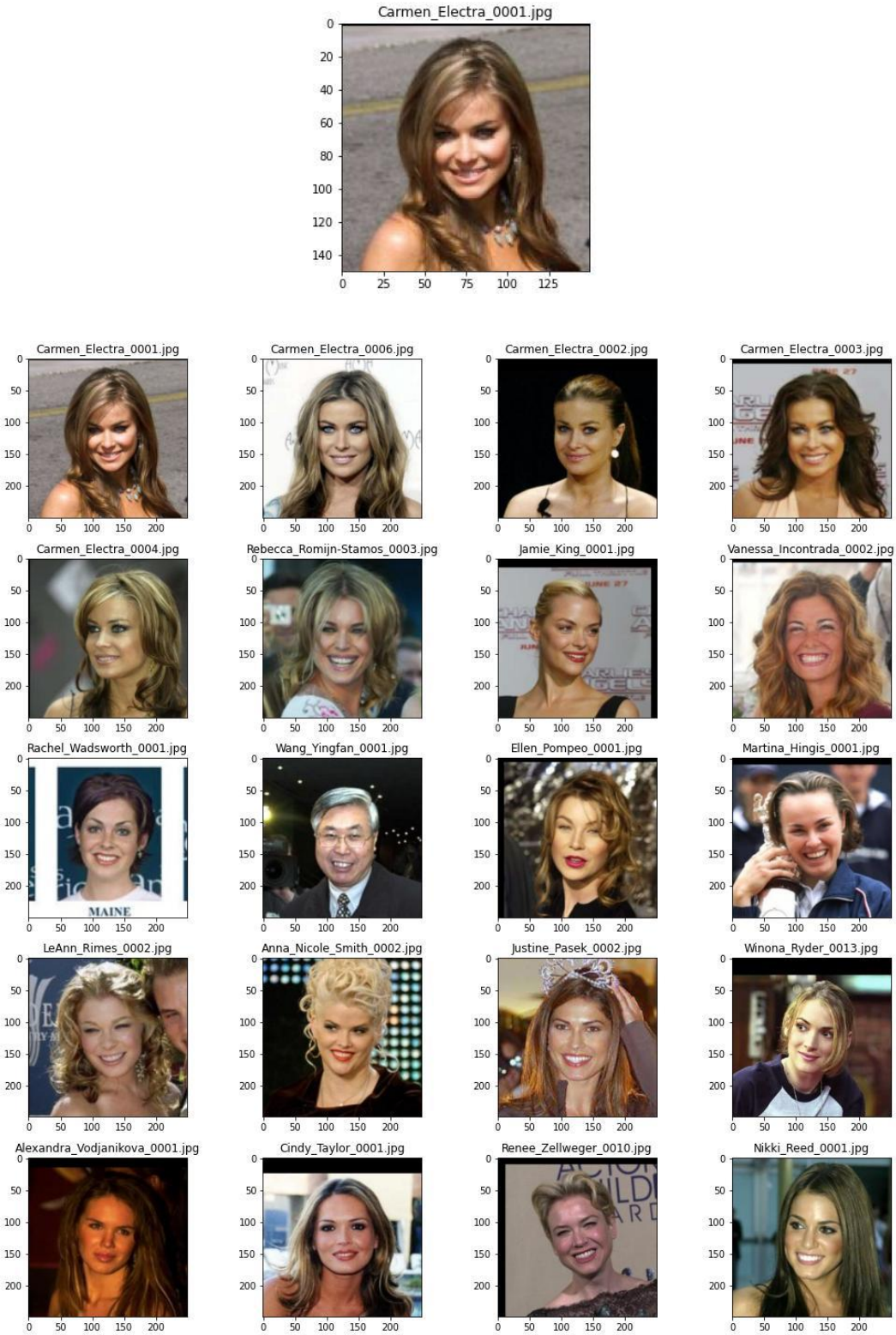

Figure 12 Carmen Electra

Figure 13 Albert Costa

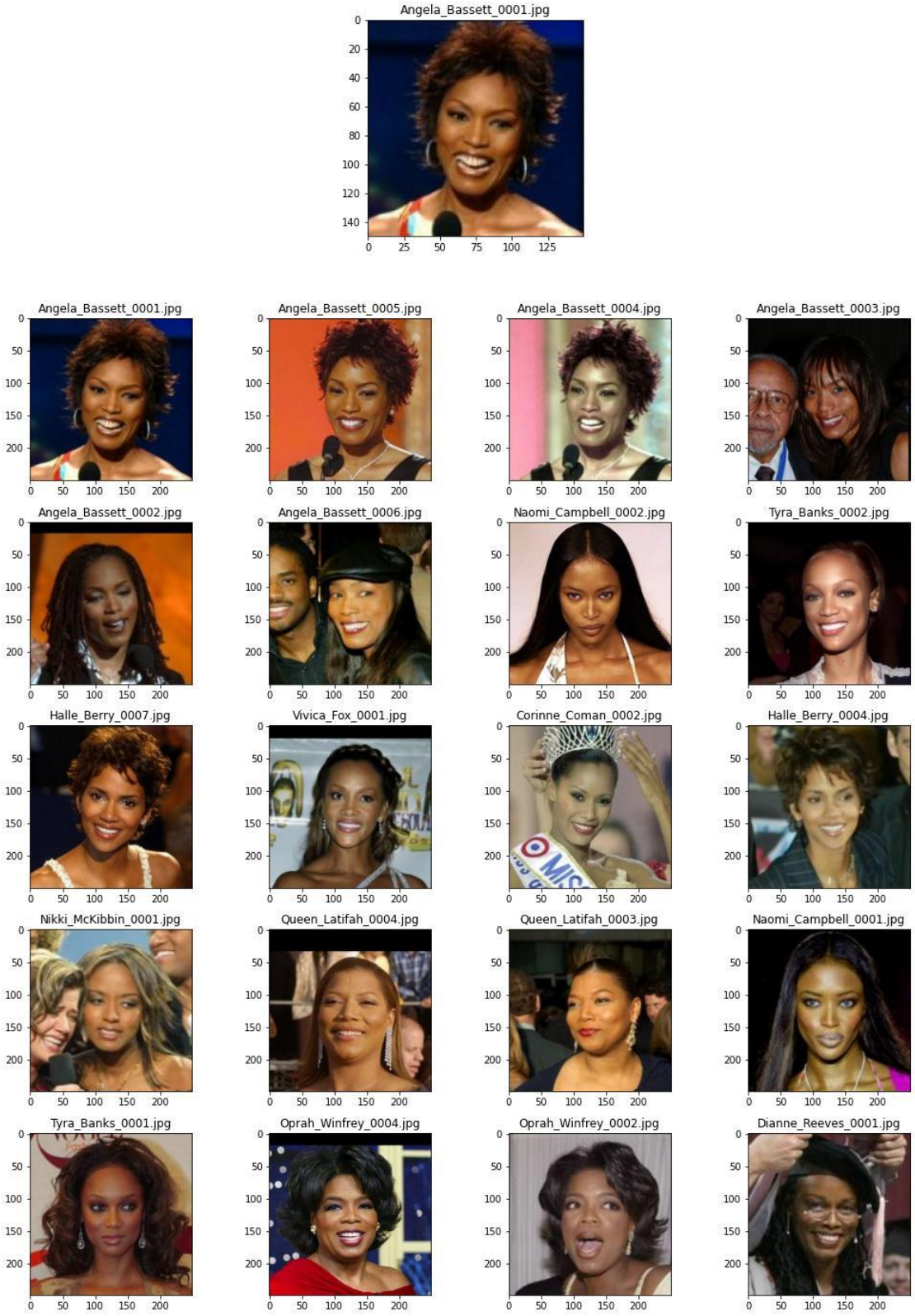

Figure 14 Angela Bassett

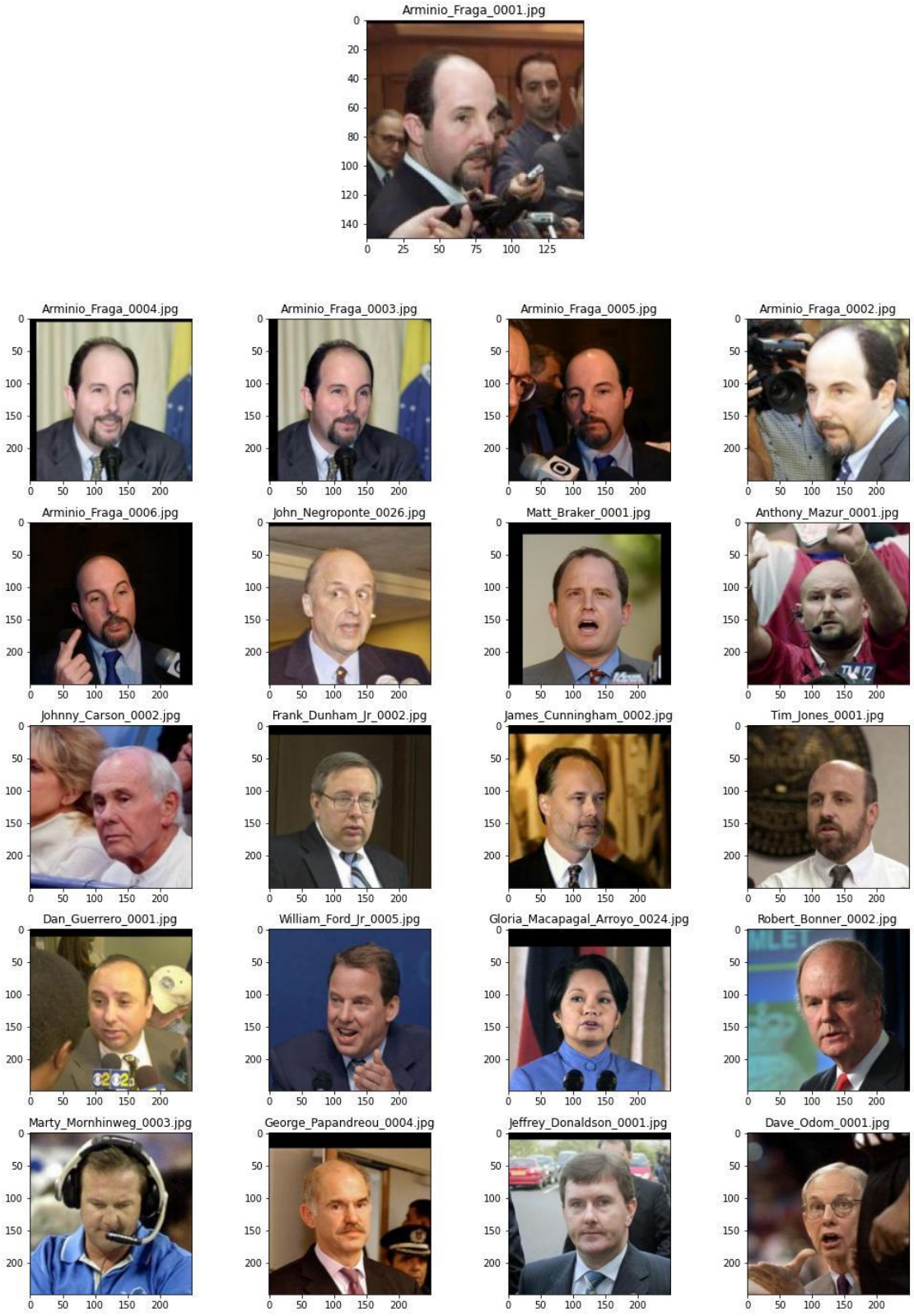

Figure 15 Arminio Fraga

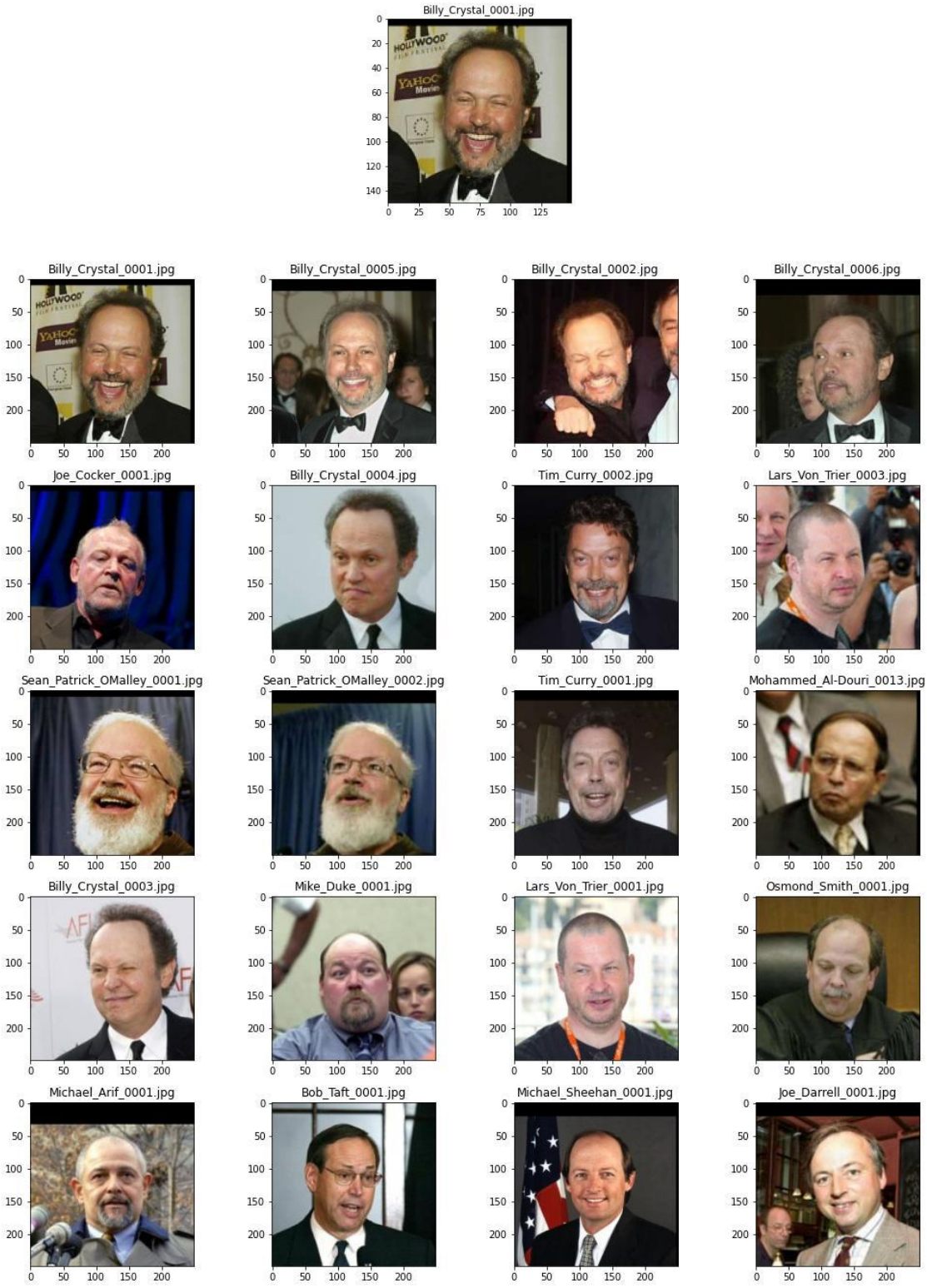

Figure 16 Billy Crystal

Figure 17 Bob Graham

Figure 18 Boris Becker

Figure 19 Bulent Ecevit

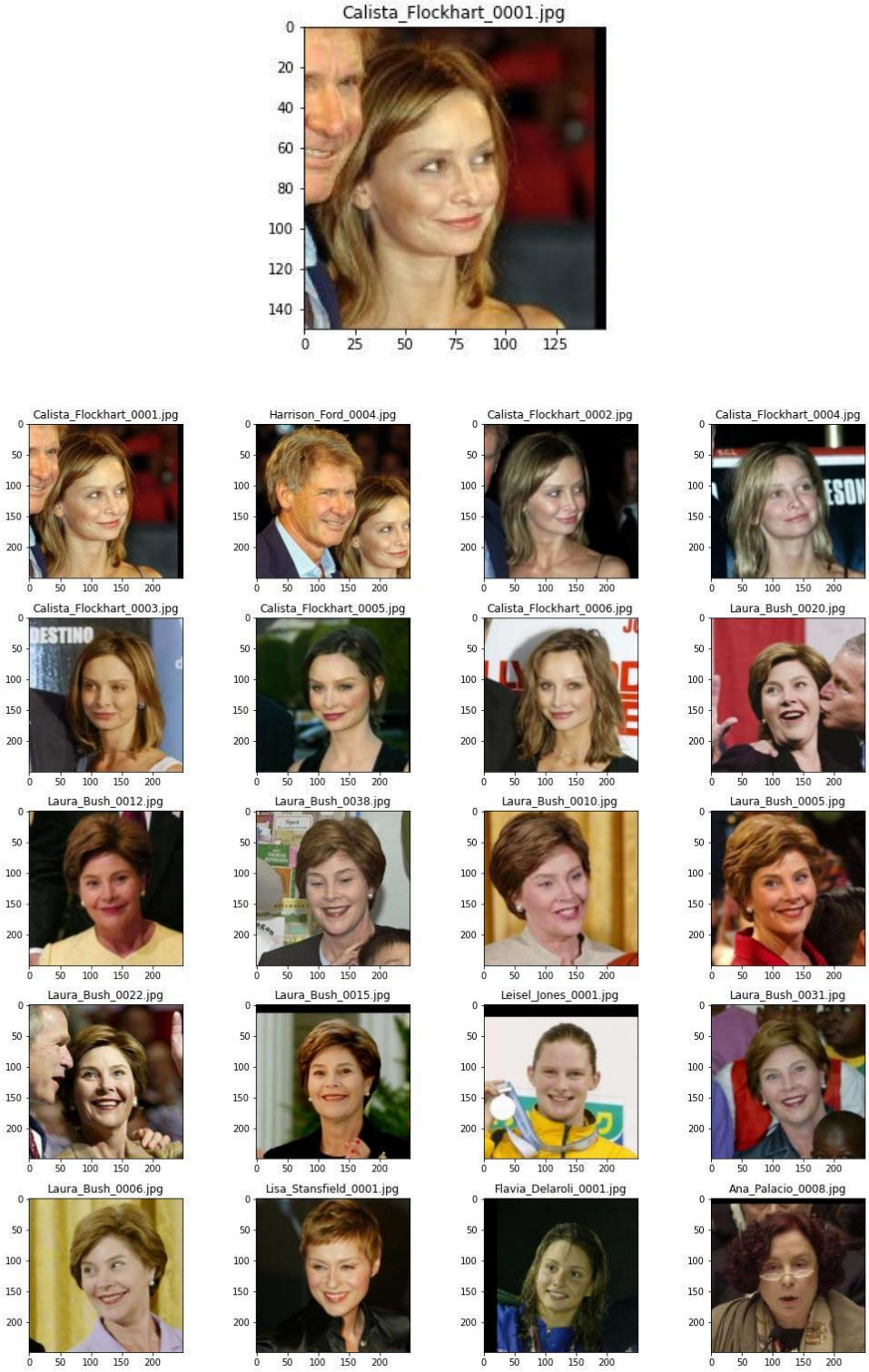

Figure 20 Calista Flockhart

Figure 21 Cameron Diaz

4. CONCLUSION

In this project we have developed a model for reverse visual search, which gives us fairly good results for query images from LFW dataset. Firstly, we extracted faces from the images in LFW using MTCNN Face Detector. These faces are fed into FaceNet in which we generating embeddings of size 1792 for each image. These embeddings are stored into a Milvus Dataset. Milvus Dataset returns 20 similar faces for each query image.

5. REPLICATE RESULTS

GitHub Repo: https://github.com/EashanKaushik/Reverse-Visual-Search

This project was completed on multiple machines:

1. Colab Pro (High Ram and GPU) for running notebooks for training.

a. model-training-baseline.ipynb
Input LFW Dataset: Link
Output: Link

b. model-training-facenet.ipynb
Input: Link
Output: Link

2. EC2 Instance for Milvus.

a. model-training-milvus.ipynb
Input:
- Link
- Link
Output: Link

b. query/Improvement-final-Milvus.ipynb
Input:
- Link
- Link
- Link
Output Final Model: Link

3. Local Machine for running notebooks for generating outputs.

a. query/Preprocessing-Queries.ipynb
Input: Link
Output MTCNN: Link

b. query/Baseline_Model-Output.ipynb
Input:
- Link
- Link
- Link

Output Baseline Images: Link

c. query/Improvement-final-FaceNet.ipynb
Input:
- Link
- Link
- Link

Output: Link

Step 1
Download the LFW Dataset: http://vis-www.cs.umass.edu/lfw/lfw.tgz

Step 2
In this step we will generate embeddings for our training dataset which is essentially the entire LFW dataset. Also, for comparison purposes we will get accuracy for our Baseline Model and Improved Model. Please Note that accuracy is not the correct term for this problem statement however just to compare models we have incorporated this into our experiments. We will split the LFW dataset into train and test images (however when we generate results for query images, we will use the entire LFW dataset). Notebooks that are used in this step are as follows:
(1) model-training-baseline.ipynb (Colab)
(2) model-training-facenet.ipynb (Colab)
(3) model-training-milvus.ipynb (EC2)

Step 3
In this step we will get 20 similar faces for 10 query images. Query images are the first 10 images in http://vis-www.cs.umass.edu/lfw/number_6.html. The notebooks used in this section are as follows:
(1) PreProcessing Step: query/Preprocessing-Queries.ipynb (First notebook to run)
(2) Baseline Model: query/Baseline_Model-Output.ipynb
(3) Final Improvement: query/Improvement-final-FaceNet.ipynb, query/Improvement-final-Milvus.ipynb

REFERENCES:

[1] https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

[2] https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff

[3] https://milvus.io/docs/image_similarity_search.md

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

[5] http://vis-www.cs.umass.edu/lfw/