NYU CS-GY 6613: Artificial Intelligence Spring 2022

Final Project

Name: Richard Fu, Akshat Sharma, Neha Galla

1 Introduction - Reverse Visual Search Overview

Reverse visual search is a popular technique used in computer vision to compare the similarity between two facial images, with the purpose of identification. We seek to train a model on a set of faces, then use the model to identify people based on query image inputs.

1.1 Dataset

We use Labelled Faces in the Wild (LFW), one of the more popular datasets used in facial recognition. This dataset is comprised of 13,233 centered images of 5,749 people, with 1,680 people that have more than two distinct images. It is important to note that we train the model on this database from the Keras module import, where the images are further zoomed than the downloadable content from the LFW website. This may affect results in the testing phase.

1.2 Architecture

The model we use will have this architecture, taken from the AWS ”Building a Visual Search Application” Machine Learning blog post,

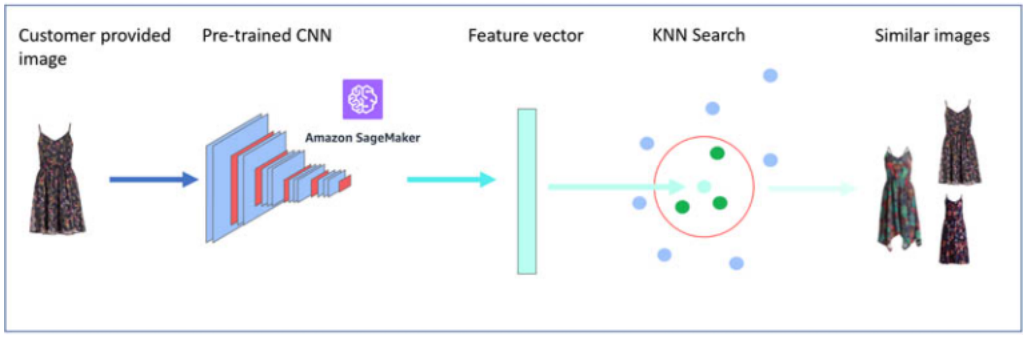

Figure 1: AWS Visual Search Architecture

Initially, we will experiment with the CNN and KNN combination described in the article and test the accuracy. Then, we will attempt to augment the model to hopefully improve on this baseline.

2 Implementation - Baseline

The baseline model we use comprises of the ResNet50 pretrained CNN, and a k-nearest neighbors similarity search.

2.1 ResNet50 Classifier

The ResNet 50 model comprises 5 stages each with a residual block, as seen in Figure 2. Each residual layer has 3 layers with 1*1 and 3*3 convolutions. In traditional neural networks, each layer feeds into the next layer but in a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away, called identity connections.

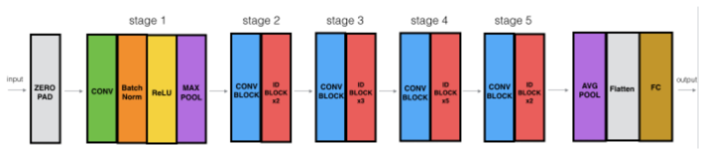

Figure 2: ResNet Model Architecture

It is trained on the imageNet dataset, which consists of over a million images of daily objects (keyboards, mice, animals, etc.) including faces classified into 1000 labels. To extract the feature vector, we remove the bottom classification layers, and the resulting vector has a size of 2048.

2.2 k-Nearest Neighbor Algorithm

The labelled feature vectors are then fed into a k-Nearest Neighbor similarity search algorithm, which detects the k-closest vectors to a query vector based on Euclidean distance.

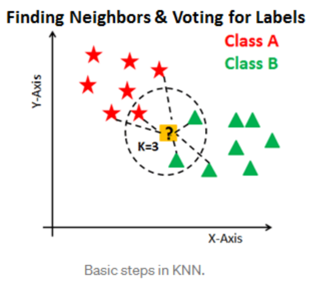

Figure 3: K-Nearest Neighbor Example

The optimal parameter k in this application would be 8 neighbors, since considering the fact that there are 1,680 people out of 5,749 in the dataset with more than two images and 13,233 total images, each person has on average 8 neighbors. However, we will use a parameter of 20 since our project objective is to find 20 closest images.

2.3 Results

We first attempt to measure the accuracy of the model. Using the Keras backend, we extract the images and labels of every person in the LFW dataset that contains 5 or more images. This is done to speed up computation, while still maintaining a significant part of the dataset (approximately 5000 examples). We then split and shuffle the images into a training and testing set. The training set is passed through the baseline CNN, and a set of feature vectors is extracted, which is then used to fit a k-nearest neighbor cluster using the Scikit-Learn library implementation. Then, the test set is passed through the baseline CNN, and we attempt to predict, from the k-nearest neighbor cluster, the correct label from which it came from. The Scikit-Learn kNeighborsClassifier predicts the class labels by taking the set of 20 neighbors for each query input and chooses the label that appears the most within those neighbors. This is then passed through an accuracy metric which judges the proportion of correct and incorrect predictions.

The accuracy metric for the baseline model was approximately **33.2%**. This accuracy is fairly low for a variety of reasons - the main one being that the ResNet50 model is trained on many labels that do not include faces. Thus, the network architecture and the weights are not designed specifically for faces and the feature vector will be far too generalized. The similarity search algorithm may also be upgraded. K-Nearest Neighbors tends to be less useful as the dimensionality of the feature space increases due to computational complexity. Additionally, k-Nearest Neighbors is heavily affected by noise and outliers in the feature space.

3 Implementation - Upgrading the model

3.1 FaceNet Classifier

Instead of the ResNet50 CNN model, we seek a model that is specifically designed for facial recognition. The model itself, which is described in the paper, FaceNet: A Unified Embedding for Face Recognition and Clustering uses a Siamese architecture, which is two identical neural networks running in parallel. Each network takes in two images and the features vector outputs are compared with a distance function d. This similarity metric judges whether or not the two images are of the same class or not.

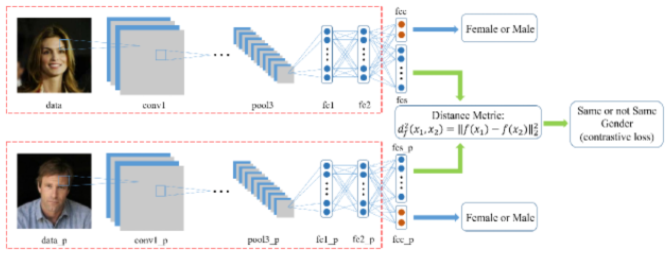

Figure 4: Basic Siamese Network

The way FaceNet trains can be described in the following example. Consider three images I1, I2, I3, where I1, I2 are in the same class (positive) and I1, I3 are not in the same class (negative). FaceNet then measures the distance using a Siamese network between I1, I2, denoted $d_{1,2}$, and between I1, I3, denoted $d_{1,3}$. We'd expect $d_{1,3} > d_{1,2}$, and so the loss function used would attempt to maximize

$d_{1,3}$ whilst maintaining a low $d_{1,2}$ when training. The effect would be a feature vector separation that splits the classes significantly. This loss training is known as the 'triplet loss', and is shown in 5. Note that in the image, I1 would be the 'anchor', I2, the 'positive', and I3, the 'negative.

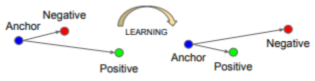

Figure 5: Triplet Loss Learning (as described in the FaceNet paper)

The FaceNet consists of 22 layers, 6 convolutional layers with two kernels each and normalization, four pooling layers, along with a flattening layer and four fully connected layers for output.

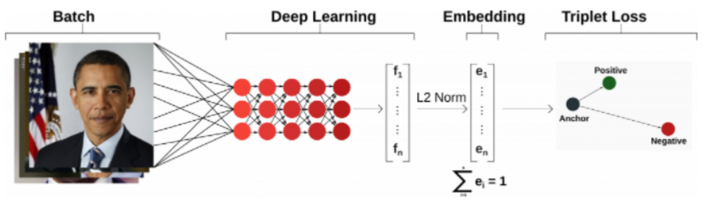

Figure 6: FaceNet Model Overview

We use the TensorFlow implementation of this model with a pretrained FaceNet model file and weights file developed by Hiroki Taniai received from Kaggle.

The output layer of the model is a 128 dimensional feature vector after dropout and batch normalization. We remove the last three layers of the model, so we can extract a feature vector right after the pooling layer similar to the ResNet process. This feature vector has a dimension of roughly 1700.

3.2 Similarity Search - SVM

A similarity search algorithm that works well in high dimensional space is support vector machines. Support vector machines work by optimizing a n-dimensional hyperplane, where n is the number of classes. It does this by maximizing the distance between the hyperplane and the vectors that are closest to the hyperplane by a margin. These points are the 'support vectors'. In doing so, it, in its vanilla implementation can find a binary separation between vectors in high dimensions, using only a few support vectors. To expand to the multiclass example, we simply break the problem down into multiple binary classifications, known as One vs One if we are separating between each pair of class, or One vs Rest if we are separating one class from the rest of the classes. Figure 11 illustrates the One vs Rest hyperplane separation.

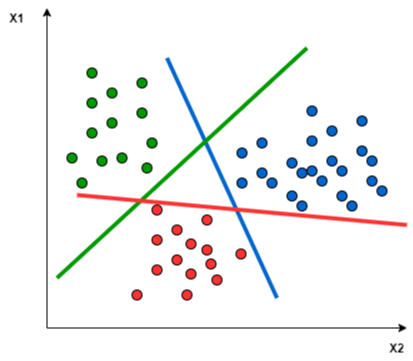

Figure 7: Multiclass SVM

We use the One vs Rest method using Scikit-Learn's SVM implementations with two separate SVC (support vector classifier) models, NuSVC and LinearSVC. It is also important to note that we use soft margin SVM, which allows for some classification errors from the hyperplane, but makes the model in general more flexible, as opposed to hard margin, which enforces a strict decision split.

3.3 Results

We apply the same training and accuracy measuring method as in the baseline model above.

We first measure the accuracy while using a k-nearest neighbors classifier to judge the effectiveness of using a different CNN. Since FaceNet is trained on specifically faces and the model is designed for facial recognition, we would expect the accuracy to be significantly superior to ResNet. We achieve an accuracy of **82.1%**, which is drastically higher than our previous result.

We then attempt to upgrade the similarity search algorithm by using SVM.

As mentioned above, the first SVC model from Scikit-Learn that we use is NuSVC, in which we are able to modify the margin ν bounds. This allows us to manipulate the number of support vectors that are used in the calculation of the optimal hyperplane. The ν value we use is chosen somewhat arbitrarily, as any higher than this would produce an infeasible model. The ν parameter enforces an upper bound on margin error and a lower bound on fraction of support vectors, so it seems that increasing this parameter, we force the model to increase support vectors in calculation, which may exceed the upper bound on margin error, causing infeasibility. We achieve a **99.4%** accuracy with $\nu = 0.025$, which is significantly higher than using k-nearest neighbors.

The second SVC model used is LinearSVC, which allows us to manipulate the loss function used when optimizing the SVC. We choose a hinge loss function, which is the common loss function for SVM classifiers. The default for SVC in Scikit-Learn's implementation is squared hinge loss. The hinge loss punishes vectors that are too close to the hyperplane, or are on the wrong side of the hyperplane. By using the regular hinge loss instead of the squared hinge loss, we reduce the total loss for misclassification, and overall allow for more errors (softer margin). This has the effect of generalizing the model. We achieve a **98.3%** accuracy with this method, which is inferior to our previous NuSVC model. This indicates that the effect of changing margin bounds has a stronger effect then changing the loss function.

4 Final Thoughts and Sample Query Outputs

The similar faces chosen are documented in the folders 'baseline', 'facenetknn' and 'facenetsvm', where the names denote the architecture used to find these faces. The names of the files within the folders correspond to the names of the query images.

To test our model against real world examples, we use the first image on the first ten people in the dataset provided at the link `http://vis-www.cs.umass.edu/lfw/number_6.html`. This accounts for images of Albert Costa to Carmen Electra.

There is a noticeable similarity increase when upgrading from the baseline model to FaceNet, even when only using k-nearest neighbors algorithm. When using the SVM model, the results are very inconsistent and often incorrect. SVM is typically used for classification, so it seems that clustering methods may be more accurate in finding and outputting similar images, but this result is somewhat surprising.

It is important to note that SVM produces a label output, so the result is all the images that correspond with that particular label. The difference between this method and k-nearest neighbors is that if there is an error with SVM, we will receive faces that are all completely wrong and may not be similar (since we will have chosen the label incorrectly). With KNN, even if there is an error, we may receive similar faces, since we are retrieving images solely based on feature vector distances.

This fact causes us problems when using the SVM model, even though the testing described above has superior results in terms of accuracy, when running predictions from external sources, it fails to classify the query image often. A fix may be to provide clearer images that are more centered around the target face, or to use the images from the Keras dataset and not jpg images scraped from the download site.

The following is an example of all three methods on an image of Calista Flockhart.

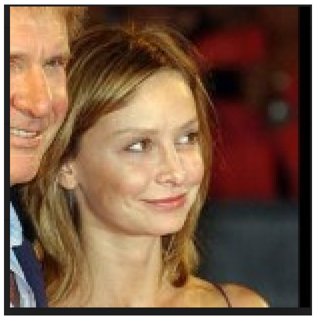

Figure 8: Original Image

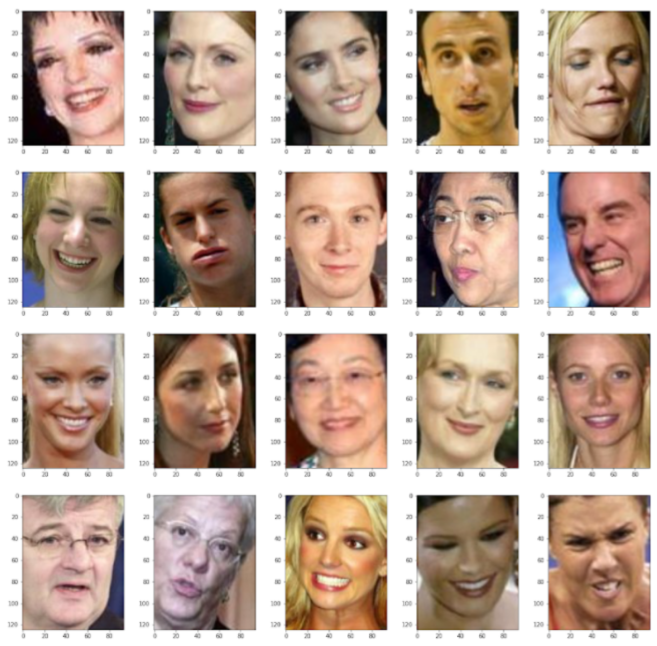

Figure 9: Example of ResNet + KNN baseline architecture similarity comparisons

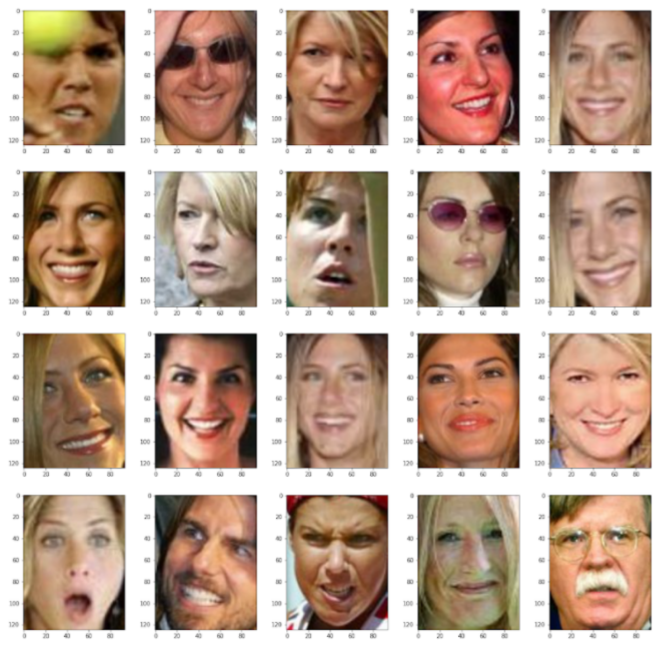

Figure 10: Example of FaceNet + KNN architecture similarity comparisons, successful

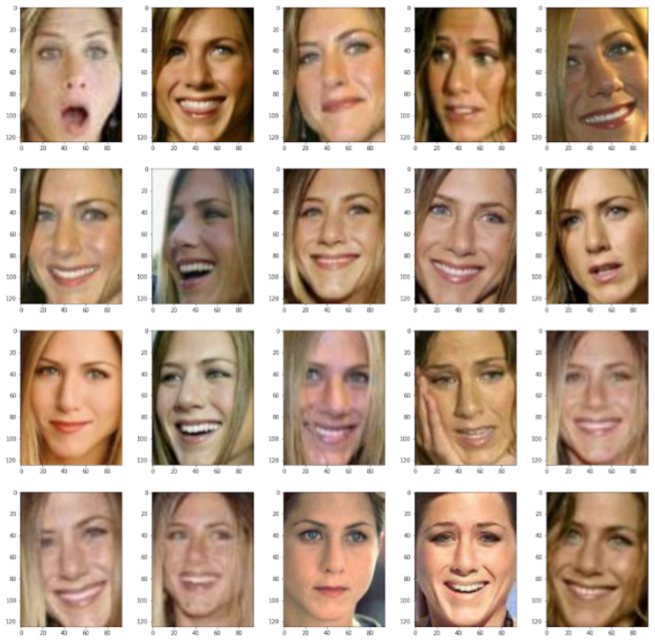

Figure 11: Example of FaceNet + SVM architecture similarity comparisons