Siamese Neural Networks for Face Recognition

Raafi Rahman

City University of New York Hunter College, New York City, New York, 10065

1. Introduction

1.1. History

Face identification is a field that has been studied for decades. Originally, when face identification was first being developed, Woody Bledsoe, Helen Chan Wolf, and Charles Bisson used various landmarks on faces such as eye location, mouth location, and so on. By comparing distances between landmarks, they were able to determine the identity of the subject. Later on, researchers added 21 landmarks such as hair color to increase model accuracy. In the ’80s, Lawrence Sirovich and Michael Kirby developed the Eigenfaces method of characterizing images of faces. In 1991, Turk and Pentland created an automatic system that can automatically detect faces in an image, allowing full automation of face identification.

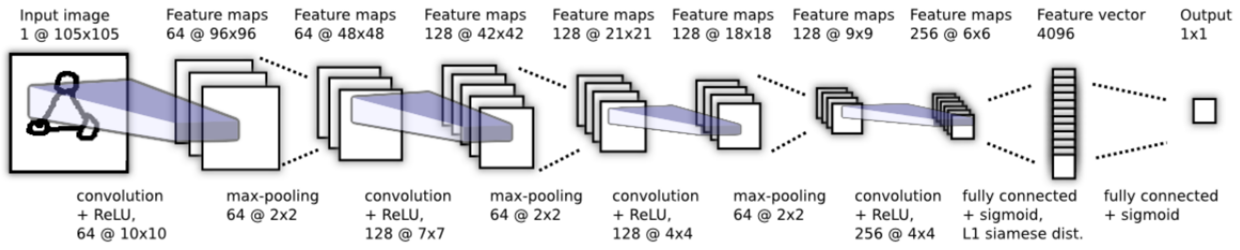
The foundation and original interest posed by these mathematicians and computer scientists created the groundwork for many applications today. The United States Department of Defense’s Defense Advanced Research Projects Agency (DARPA) had a keen interest in the technology, realizing its potential. They promoted the idea by organizing and supplying datasets to inspire those interested. Companies in the private sector also took interest. Meta, known as Facebook at the time, started developing their own models using user data. Apple also applied the technology to develop their FaceID, which comes standard in all new iPhones today.

This technology has many applications and uses. They range from security to convenience. Identify people in a large crowd to ensure safety, only allow permitted people access to a restricted area such as a workplace, track accounts by the face of the owner rather than an arbitrary identification number, or allow you to have access to your mobile devices.

1.2. Siamese Neural Networks

It is believed that Apple's FaceID uses a Siamese Neural Network model. Exact information is hard to uncover due to trade secrets, but this is our best inference. Siamese Neural Networks get their name from the fact that two similar, in fact exactly the same, models run in parallel. The outputs from the final layers of both of these models are then fed into a single layer whether the model connects. Here you can apply some sort of transformation or calculation using both of these outputs and continue on with your layers. Below is an image taken from "Siamese Neural Networks for One-shot Image Recognition".

Neural Network Architecture



This is the embedding model. This piece of the network embeds the image into a vector in representation space. We then make another model. This new model we will call the siamese model. The siamese model takes two inputs, both vectors of length 4096. These vectors are then used to find the difference vector between them using some metric. Finally, we take this vector and connect it to a single node dense layer with a sigmoid activation function. This output node tells us whether the two images are a match or not. Match if the value is above some threshold, and not a match otherwise.

2. Implementation

2.1. Data

We use a combination of different datasets for our training, validation, and testing sets. We use the images from Labeled Faces in the Wild along with a Bollywood celebrity dataset. We make pairs from them and label them with a 1 or a 0 for a match and non-match, respectively. We will call a pair that is a match a positive, and a pair that is not a match a negative. The way we make the pairs is by looping over every folder that contains more than 1 image (Each identity has a unique folder) and every image in that folder. We take a list with every image in that folder, make a duplicate list and reverse it. Now we have a set of positive pairs with a length equal to the number of images contained in that folder. Then we choose images from the complement of the dataset for each image in our list to create an equal number of negative pairs. In total, we end up with 10124 image pairs, half positive pairs and half negative pairs.

2.2. One-Shot Learning

Originally, when testing the concept of Siamese Neural Networks, I used the MNIST hand-written digits dataset. The data was also organized differently. Instead of using pairs, the data was organized into three lists: an anchor list, a positive list, and a negative list. This implementation did not use a one-shot learning approach. Here, I arbitrarily determined my model would learn to differentiate between a 4 and a non-4 (An integer element of [0,9] - 4). I would populate the anchor list with *n* pictures of 4, the positive list with another *n* distinct 4s, and *n* non-4s. Then my model would take the *i*th image of each list and make pairs using the *i*th anchor and positive, and the *i*th anchor and negative.

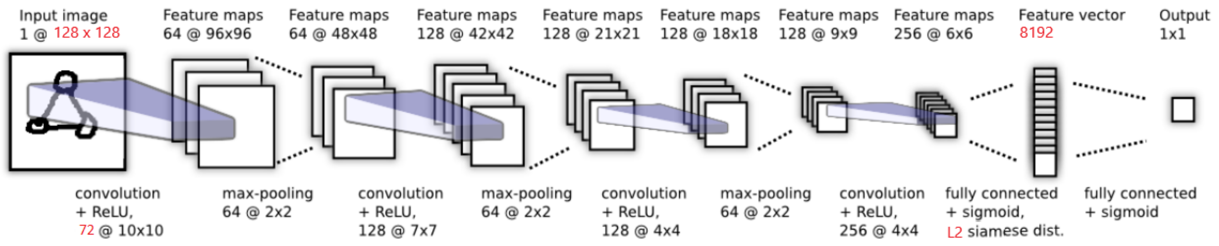
The weakness in this approach is that the model only learns what a 4 is and only a 4. It will not learn to differentiate all hand-written digits from each other, but rather 4s from other integers. Suppose I had this “4 recognizing” model and now I want to apply it for 3 instead. Analogously, it may be more intuitive to think about a phone that recognizes my face in order to unlock it. But let’s say I give away my phone to my brother and need FaceID to recognize him instead of me. The whole model will need to be trained again. In a one-shot learning approach, we avoid this situation. We create our model in a general form so that it can be used dynamically rather than statically.

Our improved approach would take pairs of images and the model will learn to classify 2 images as a positive pair or as a negative pair regardless of whose face is in the image. The model will do this by comparing features in the images. This is more robust and allows us to use this model for more general purposes. Continuing our FaceID example, now the model will not need to retrain when we need to switch who the verified user is. Instead, the model will be given a new set of images of the verified user, forget the old user’s images, and continue working how it has been. This is also useful in situations where you have more than one verified user. For example, an employee sign-in turnstile in the lobby of an office building. You can have hundreds or thousands of employees that use this system. People leave the company and people join the company all the time. Retraining the model every time there is a change in your workforce is inefficient. With this one-shot approach, you simply need to control the database of verified users, updating the images stored in the database as the workforce updates.

2.3. Architecture

We have already reviewed the general architecture of the Siamese Model. We will be deviating slightly from the architecture presented in 1.2. For example, our input images will be of shape (128, 128, 3). Each image is 128 pixels by 128 pixels with three color channels, red, green, and blue. Another change is instead of a feature vector of length 4096, we will be using a feature vector of length 8192. After the embedding layer, we are opting to use the L2 distance rather than L1 (see 2.4.1). Everything else largely remains the same.

Modified Neural Network Architecture



Embedding Network Summary

Layer (type)	Output Shape	Param #
input_image (InputLayer)	[(None, 128, 128, 3)]	0
conv2d (Conv2D)	(None, 128, 128, 72)	21672
Max_pooling2d (MaxPooling2D)	(None, 64, 64, 72)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	451712
Max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	262272
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 256)	524544
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 8192)	536879704

Siamese Network Summary

Layer (type)	Output Shape	Param #	Connected to
image_1 (InputLayer)	[(None, 128, 128, 3)]	0	[]
image_2 (InputLayer)	[(None, 128, 128, 3)]	0	[]
embedding (Functional)	(None, 8192)	538139304	[‘image_1[0][0]’, image_2[0][0]’]
lambda (Lambda)	(None, 1)	0	[‘embedding[0][0]’, embedding[1][0]’]
batch_normalization (BatchNormalization)	(None, 1)	4	[‘lambda[0][0]’]
dense_1 (Dense)	(None, 1)	2	[‘batch_normalization[0][0]’]

The ‘embedding’ layer in the Siamese Network Summary is the Embedding Network. Our final, complete network is the Siamese Network. The model will take two input images and classify them on whether they are a positive or a negative.

2.4. Distance Metrics

The next few functions I will talk about take the embeddings of both of our vectors as input. These embeddings are vectors in our *n*-dimensional representation space. The following functions are different metrics we can use to measure distance in this space.

2.4.1. Euclidean Distance

The L2 distance is the distance we are familiar with. It is represented as follows…

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2.4.2. Manhattan Distance

The L1 distance is similar to the L2 distance except that you don’t take squares or square roots. It is represented as follows…

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)$$

2.4.3. Cosine Similarity

Cosine similarity measures distance using the angle between two vectors

$$similarity(x, y) := cos(\theta) = \frac{x \cdot y}{||x||\,||y||}$$

where $x \cdot y$ is the dot product between *x* and *y* and ||•|| is the L2 norm.

2.5. Loss Functions

2.5.1. Triplet Loss

Triplet loss is one loss function often used with Siamese Neural Networks. This method looks at three images at a time. One will be the anchor which is the image we will be testing. Then we have two more images, the positive and the negative. The positive will be another image of the subject in the anchor while the negative image will be someone else. We can compare the similarity between the anchor and positive, and then the anchor and the negative. We want the distance between the anchor and positive to be much smaller than the distance between the anchor and the negative. During training, we try to make our parameters such that the difference between the two distances is as large as possible. Let *f* be our model, *A* be an anchor image, *P* be a positive image, *N* be a negative image, and α be some margin hyperparameter. We want to achieve the following relationship.

$$||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha \leq 0$$

Note that the output of *f* is our embedding vector of length 8192. Thus our loss function is…

$$L(A, P, N) = max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0)$$

And our cost can be defined as

$$C = \sum_{i=1}^{m} L(A_i, P_i, N_i).$$

We need to minimize this cost function in order to have an accurate model.

2.5.2. Contrastive Loss

For our purposes we are using Contrastive Loss. Contrastive loss doesn’t look at 3 input images at a time. Rather, it looks at two images at a time and determines whether they are similar or different. Or more precisely, how far two images are in the representation space based on the embedding function. Then based on that, parameters are adjusted so that images that should be far apart are actually far apart, and images that should be close together are actually close together.

$$L = (1 - y_{true})\, L_s(y_{pred}) + y_{true} L_d(y_{pred})$$

Contrastive loss takes the predicted output and the true output. Depending on whether the true output is one thing or the other, one term in the function activates and the other is multiplied by 0, canceling it out. In either case, if the prediction is close to the true value, the loss added is minimal. If the prediction is not close to the actual, then the loss added is larger. This is summed up and the final cost is returned.

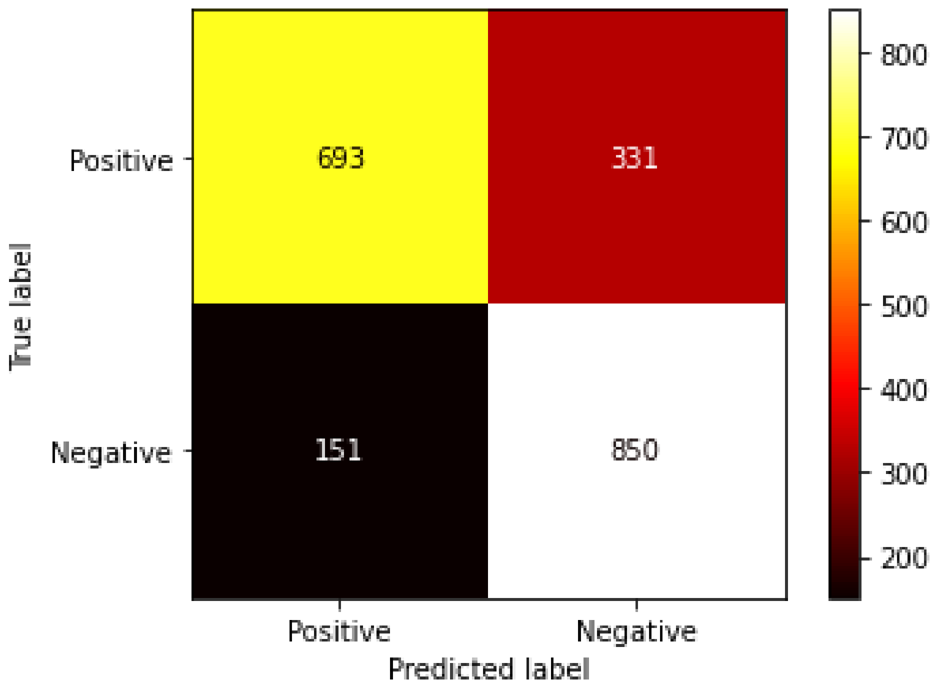
2.6. Training

We train our Siamese Network model using our training data for 100 epochs. As stated in the previous section, we will be using contrastive loss. Our optimizer is Stochastic Gradient Descent and we are optimizing for precision. The reason for this is that a false positive is more costly to us than a false negative. It is preferable for the model to predict negative when a verified user tries to verify their face, rather than for the model to predict positive when a non-verified user tries to verify their face.

3. Results

Below is the confusion matrix from my final model. My final model had an accuracy of 76% and a precision of 82%. A problem I encountered while making this model is creating "challenging" training pairs. Many of our negatives were easily identifiable as negatives. The model training off this data will not be sufficiently prepared to handle pairs of different people that look similar. While checking what the false positive pairs were, I noticed many were of people of the same minority race. As a way to counter this, I added more data on people of color. For example, I added a Bollywood celebrity database and made pairs from that. Another cause for false positives was the pose of the images. If the people in both images were in a similar pose, the model had a higher chance of labeling the pair as positive.

Confusion Matrix



What are some ways we can improve this model? More data and lots of it. Large companies with large amounts of resources and large pools of existing customer data are able to use and process very large amounts of quality data for their models. Even more helpful would be training the model off challenging pairs. An example of a challenging pair could be comparing images of siblings. Since siblings usually look similar but are in fact different people, we can train our model to spot small differences that differentiate similar-looking people. Companies like Apple go a step further. Rather than using 2-dimensional images (3-dimensional if we include the RGB channels), Apple uses a depth perceiving sensor in order to make a 3-dimensional scan of the face, further increasing accuracy.

A special thank you to Professor Bora Ferlengez for consistently encouraging me to do more.

Datasets

Labeled Faces in the Wild with cropped faces

https://www.kaggle.com/datasets/jonathanloscalzo/lfw-cropped-faces

Face Dataset all at one place \ celebs

https://www.kaggle.com/datasets/shanmukh05/vggface-using-tripletloss

Originally taken from Bollywood Celebrity Faces

https://www.kaggle.com/datasets/havingfun/100-bollywood-celebrity-faces

Technologies

Libraries: Scikit-learn, TensorFlow, NumPy, Matplotlib, OpenCV

All code was written in Google Colaboratory.

The data organization script was written in Visual Studio Code.

References

https://www.nec.co.nz/market-leadership/publications-media/a-brief-history-of-facial-recognition/#:~:text=The%20earliest%20pioneers%20of%20facial,to%20recognise%20the%20human%20face.

https://machinelearningmastery.com/how-to-create-a-random-split-cross-validation-and-bagging-ensemble-for-deep-learning-in-keras/

https://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision/Readings/1987-Sirovich_and_Kirby-Low_Dimensional_Procedure_for_the_Characterization_of_Human_Faces.pdf

https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

https://www.youtube.com/watch?v=-FfMVnwXrZ0&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=32&t=3s&ab_channel=DeepLearningAI

https://www.youtube.com/watch?v=96b_weTZb2w&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=33&ab_channel=DeepLearningAI

https://www.youtube.com/watch?v=6jfw8MuKwpI&ab_channel=DeepLearningAI

https://www.youtube.com/watch?v=d2XB5-tuCWU&t=595s&ab_channel=DeepLearningAI

https://www.youtube.com/watch?v=0NSLgoEtdnw&ab_channel=DeepLearningAI