Face Matching: Replication of Google Facenet

Saipranav Janyavula
University of Michigan
Ann Arbor, MI
sjanyavu@umich.edu

1. Introduction

The goal of the project is to create a face matching system that can recognize a person's face and find other pictures of the same person by utilizing the Google Facenet model. This type of system is quite common and is today part of the default functionality of common photo storage and photo sharing applications and services including Apple Photos, Google Photos, and Amazon Photos. These applications automatically scan all photos and create albums for each unique person (or even animal) found. Given that this functionality is commercially available (with generally good efficacy), there is no practical application to this project. It primarily serves as an educational experience and a tool to demonstrate my learning in taking EECS 442. Combined with a simple user interface, it can serve as a simple and quite visual demonstration that can be shown to others.

2. Related Work

Facenet (2014) was part of the first generation of models which applied Convolutional Neural Networks (CNN's) to the task of Face recognition. The earliest models (such as Eigenfaces, 1991) utilized various forms of principal component analysis and created vector embeddings representing the combination of base "eigen pictures" that would recreate the original face. Later methods would continue to build on this work without Neural Networks, with algorithms such as the Discriminant Face Descriptor achieving 80% accuracy on the Labeled Faces in the Wild (LFW) Dataset.

However by utilizing Neural Networks, algorithms such as Facebook's deepface and Google's Facenet would match or exceed human performance on popular datasets such as Labeled Faces in the Wild.

Although Facenet and related models represented a big advancement in 2014, facial recognition models have continued to advance since then. Apple's on-device face recognition model is built on the AirFace face recognition model that utilizes a linear version of the ArcFace angular loss function. This is then combined with a body embedding model to boost accuracy and reliability.

3. Method

The Facenet paper didn't create a new network architecture. Instead it modified two previous architectures, one from Zeiler&Fergus labeled NN1, and the other by C.Szegedy et.al. labeled NN2 which were originally created for image classification. The minor modifications led to a 128 dimension embedding vector as the output. These were then trained by utilizing a triplet loss function where each sample consists of 3 images (as embeddings): an anchor, a positive (representing the same identity as the anchor), and a negative(representing a different identity). The loss function attempts to minimize the difference between the l2 distance from the anchor to the positive, maximize the l2 distance from the anchor to the negative plus a constant margin as shown in the equation below.

$$\sum_{i}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2-\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]_+ \quad (1)$$

The big contribution of the Facenet paper was the utilization of an online triplet selection algorithm. Traditionally triplets would be pre-generated which could lead to issues as if the negative vector was already very far, the model would learn slowly. Alternatively if the negative vectors were already very close (and therefore "hard"), it could lead to a local minima in training. To combat this, the paper generated triplets during training (or "online") for each batch by generating all possible anchor-positive pairs and selecting a "hard" negative for each.

In their experiments, both NN1 and NN2 (while having different numbers of parameters) both had similar accuracy levels, so I chose to implement NN1 as shown in figure 1. To do this I replicated all the layers in pytorch except for the "concat" layer as I didn't fully understand what oper-

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Figure 1. NN1 by Zeiler&Fergus

ation was taking place, and added padding as necessary to achieve the same output sizes. For training data I utilized Microsoft's DigiFace-1M dataset which consists of 1 Million synthetically generated faces. However due to the comparatively low resolution of these images (112 by 112px), I chose to remove the maxout pooling operations from between the linear layers to prevent a dimensional reduction below 128. Additionally as I was unable to figure out how to perform effieient triplet selection, I utilized triplet selection code from Adam Bielski's siamese-triplet github repository

4. Experiments

The model was trained on google colab on the v100 gpu. Due to memory limitations and for speed of training, the batch size was limited to 10 identities with 20 photos per identity. Across 400 batches, this led to a total training set size of 80k images taking 3.57 hours to complete 50 epochs.
This is in contrast to the original paper which utilized between 100M-200M images with 1800 images per batch. The original paper trained the model on a cpu cluster which ran for between 1000-2000 hours. For learning parameters, the original paper utilized Adagrad with a triplet loss margin of 0.2 and a learning rate of 0.05. In my implementation I replicated these parameters except for the learning rate decay which I arbitrarily set at a constant 0.001 as the paper provided no further explanation beyond the statement that they lowered the learning rate to finalize the model.

For testing I utilized the LFW dataset by combining all 530 images of George W. Bush (the most of any person in the dataset) with XX images from all people whose names start with the letter A. As a baseline, I utilized a pretrained model from the facenet_pytorch library to match images of Bush given a single image of him. Both models were provided with resized and normalized face snippets from the MTCNN model in the facenet_pytorch library. The pretrained model unsurprisingly performed well with a 98.4% accuracy rate. Given the same task, my model matched 368 out of 529 images giving it a 69.4% accuracy rate. However it also matched 382 out of 1054 negatives giving it a 36.2% false positive rate as shown in figure 2.

As a smaller experiment, I attempted to utilize personal photos however the model performed poorly. Out of 10 personal photos, with 5 containing myself, the model matched 4 images without myself and one image with me where it matched on another person. This is likely due to the fact that the LFW dataset comprises public (likely professionally taken) images of public figures which would most likely be from various press-conferences or other public events where there is likely to be good lighting and the subject is most likely centered and in focus. Additionally the LFW images where most likely taken with DSLR cameras which have a depth of field effect not present in most smartphone images unless added digitally.

5. Conclusion

As stated earlier, this project has no practical application given the modern prevalence of highly accurate commercial solutions. As an academic exercise, one can consider many potential improvements to the training of the model including the usage of larger batch sizes with more identities which could enable the selection of better triplets leading to faster training. There is also likely a data quality issue. Although the Microsoft DigiFace dataset provided a large number of samples, being digitally generated they are likely to not be fully representative of real images and generally lack the level of detail present in real images. This can be seen visually in Figure 3 where the synthetic images appear blurry and distorted. Additionally the images were of a uniform and low resolution of 112x112px compared to Google's dataset which had images across a range of resolutions from 94-244px which would likely prevent the model from overfitting as the images are more varied. The Facenet paper also provided relatively little information on learning rate decay, experimenting with this will likely lead to improvements as I simply chose an arbitrary constant decay rate of 0.001.

However the bigger takeaways are in my opinion personal. This project revealed that I had core misunderstandings about CNN's and Machine Learning in general. One of the biggest issues was that I didn't understand how to switch

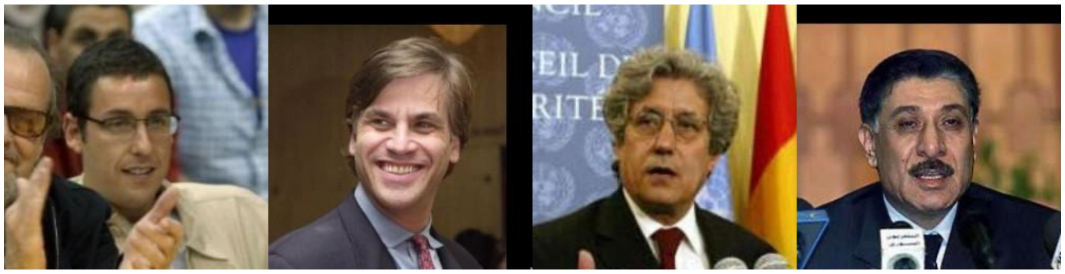
Figure 2. Sample False Positives.

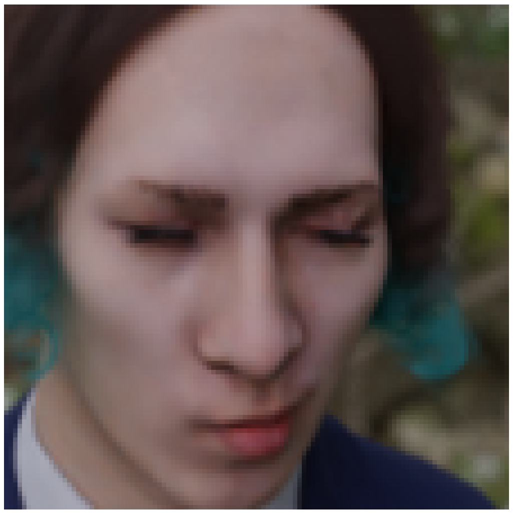
Figure 3. Sample image from DigiFace-1M dataset

from a convolutional layer, with a tensor of shape [batch size, depth, width, height] to a linear layer which requires the tensor to be reshaped into 2d with dimensions [batch size, depth x width x height]. When I first attempted to build the network I simply set the parameters to the desired depth and assumed that pytorch would automatically perform the correct operation. Similarly I found that I lacked a full understanding of how to vectorize algorithms which led me to utilize pre-written triplett selection code from Adam Bielski's siamese-triplet github repository. This experience has taught me that there is still a lot more to learn in this field and I anticipate further growth as I continue my educational and professional career.

6. References

https://machinelearning.apple.com/research/recognizing-people-photos
https://github.com/adambielski/siamese-triplet
https://github.com/timesler/facenet-pytorch
Facenet: https://arxiv.org/abs/1503.03832
Deepface:

`https://scontent-cgk1-1.xx.fbcdn.net/v/t39.8562-6/2`

Airface:

`https://arxiv.org/pdf/1907.12256.pdf`

Eigenfaces:

`https://watermark.silverchair.com/jocn.1991.3.1.71.`

Zieler:

`https://arxiv.org/abs/1311.2901`

Szegedy:

`https://arxiv.org/abs/1409.4842`