Deep learning – group practice 1

Team: Aaron Jay Fleishman, Arturo Ford, Edgar Daniel Toro Levano, Manon Stévenne, Max Caro, Santos Saguier
Project: One-shot face recognition
March 2023

Index

1. Introduction

1.1 The problem

As a group assignment, we were asked to implement a one-shot face recognition system. No specific guidelines were given so we had the freedom to explore and employ the models and algorithms we liked most. We did have to use state-of-the-art convolutional neural nets in the implementation. We first researched several papers, repositories, and models to get an understanding of the current most used technologies to build a one-shot face recognition model.

1.2 Literature Review

As a result, we found that the most popular model for face detection is the Siamese neural network (SNN) with its multiple variations. The use of GANs is also being implemented for model enhancement.

The Siamese neural network works as follows. Two or more separate inputs are processed through identical neural network architectures before being combined. The output of this combined processing is then used to determine whether the two inputs are similar or dissimilar. In face recognition, the Siamese neural network is trained on a large dataset of labeled face images to learn to recognize whether the images are of the same person or of different people. One of the advantages is that it can handle variabilities in the pictures, such as pose and light. On the downside, it is considered computationally expensive to train the SNN when working with large datasets.

The two most popular loss functions for training the SNN are the contrastive loss and the triplet loss. The contrastive loss uses the distance between the feature representations of pairs of input images computed. The model is then trained to minimize the distance if the images belong to the same person and maximize the distance if the images belong to different people. The triplets loss function takes three face images as input - an "anchor" image, a "positive" image of the same person as the anchor image, and a "negative" image of a different person. The network then learns to map these images into the feature space, such that the distance between the anchor and positive images is minimized, while the distance between the anchor and negative images is maximized. This helps the network learn efficiently to differentiate between different faces. We chose this architecture for our implementation and more details will be provided in subsequent sections.

Finally, as mentioned earlier, there has been an increasing trend in the use of Generative Adversarial Networks (GANs) in order to enhance face-recognition models. There is a particular use of GANs that is called “Face Hallucination” where the network learns to create high-resolution versions of low-resolution face images. This is particularly useful when using a scarce dataset or with low-resolution images since this type of data augmentation would lead to better performance on the final face recognition model.

2. The Model

2.1 Theoretical Overview

For the implementation, we selected the python library ***face_recognition***, which offers a simplified pipeline in order to perform a variety of face recognition implementations out of which we chose to perform real-time face recognition. Under the hood, the face_recognition library runs

another library called ***dlib*** (C++) which breaks down the face recognition issue into smaller step-by-step problems in a cunning way:

First, the algorithm has to be able to detect faces within a picture. Face detection implementations have been around for more than 20 years with effective results. In our particular case, the method implemented is called Histogram of Oriented Gradients (HOG). It does its task by turning the image into a grayscale and then making a pixel-by-pixel analysis when compared to its surrounding. Specifically, it computes the gradient of the image intensity at each pixel and groups these gradients into "cells." The gradient magnitude and direction in each cell are used to construct a histogram of gradient directions. These histograms are then normalized over larger blocks of cells to increase robustness to changes in lighting and contrast. Ultimately, HOG features are used in conjunction with a support vector machine (SVM) classifier to distinguish between face and non-face regions (eyes, nose, mouth, jaw). The SVM is trained on a dataset of positive examples (faces) and negative examples (non-faces), using the HOG features as input. During testing, the HOG features of the input image are extracted and fed into the SVM, which outputs a prediction of whether or not the image contains a face.

Second, the library has to deal with the issue that faces in images have a high variability due to changes in the person's pose, light, etc. During this step, it is applying basic transformations of stretching, rotating, and scaling the image in order to "normalize" as much as possible all faces so that the most important face landmarks (eyes, mouth, nose) are centered.

Third, the images have to be encoded so that they are readable by the model. The encoding works as a unique compressed representation of each of our faces and is achieved using the SNN with triplet loss. Training such a neural net is highly computationally expensive. However, its results can be implemented relatively easily once it has been done (there are a lot of available open-source trained nets) and that is exactly what our model is doing. The model outputs 128 measurements as the encoded representation of each face.

Finally, once the complex deep learning process is done, we simply need to make a machine learning classifier that is able to correctly predict if two embedded representations of faces belong to the same person or not. In the case of the library, we are using the implementation as an SVM classifier.

2.2 The Pipeline

After reviewing the process and algorithms that run in the library, this section details the pipeline and steps taken in the code. As stated earlier, the library is quite user-friendly and simplified so it doesn't require a large code implementation.

2.2.1 *Dataset and pre-processing*

We begin with importing the relevant libraries. On the pre-processing side, we just needed to create a repository with all the images of the people we would like the program to recognize. For investigation purposes, we collected images of class students from a variety of sources with different qualities to see how the model performed in each case. The name of the images was the name of the students themself, so this was used for labelling purposes.

2.2.2 Implementation

The face encoding of the images in the repository is generated with the trained SNN and stored. We then proceed to the real-time detection where the camera is activated to retrieve live-stream. Once it detects a face (as explained in the theoretical overview) all the mentioned steps are triggered and we end up comparing our encoded repository to the encoded representation of the

person that appears in the camera frame. If the encoded versions match, then the person's name will be displayed along with a probability given by the model.

2.2.3 Evaluation & Results

When testing the system, results were highly satisfactory in detecting people's faces correctly. We would like to highlight some aspects of the implementation.

- To begin with, all demos were performed in indoor environments which we would expect to favour better results.
- The model performs quite well even when people have distinctive items such as glasses for example.
- The model, as explained by the authors of the code themselves, still has some limitations in cases of aging for example, or when dealing with face recognition in kids.

An issue that always arises when dealing with this kind of model is the dichotomy between speed and precision. One would assume that in most cases face recognition is implemented in circumstances such as enforcement and security issues, entrance access, etc. where a speedy process is required but where the precision of the model is highly important as well. In this regard, we can play with the tolerance of the model and indicate how strict or permissive we want it to be by setting personalized thresholds.

3. Conclusion

In conclusion, this paper discusses the implementation of a one-shot face recognition system using state-of-the-art convolutional neural networks. The Siamese neural network with triplet loss was chosen for the implementation, which learns to recognize whether images are of the same person or of different people. The implementation uses the python library face_recognition, which offers a simplified pipeline for face recognition, including face detection using Histogram of Oriented Gradients (HOG), normalization of faces, encoding using the SNN with triplet loss, and classification using a support vector machine (SVM) classifier. The paper also discusses the use of Generative Adversarial Networks (GANs) for model enhancement and highlights the benefits of face hallucination for better performance on low-resolution images. Overall, the paper presents a comprehensive overview of the one-shot face recognition system and its implementation using the chosen technologies.

4. References

Model overview: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

Face_recognition library: https://pypi.org/project/face-recognition/

Dlib library: http://dlib.net/

HOG: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Siamese Repo: https://github.com/morkertis/One-Shot-Face-Recognition

Contrastive-paired vs triplet loss:

https://arxiv.org/pdf/2004.04674.pdf#:~:text=The%20triplet%20loss%20considers%20the,anchor%2Ddistant%20pairs%20of%20samples.

Keras implementation of SNN with triplet loss:

https://keras.io/examples/vision/siamese_network/

Centering faces: https://www.csc.kth.se/~vahidk/papers/KazemiCVPR14.pdf