COMP400 Prometheus Vision Facial Recognition

Ryan Sowa
ID: 260886668
Supervised by Prof. Joseph Vybihal

1 Abstract

Recently, facial recognition using artificial intelligence has become an increasingly growing field, especially as it relates to government and personal issues. In this paper, I will be analyzing previous works in computer vision centered on facial recognition. I will be specifically looking to address the following key questions:

1. What exact mechanisms does computer vision use to recognize a person?
2. How does computer vision specifically identify the unique features in an individual which distinguishes them from other individuals?
3. How many photos must the algorithm analyze to correctly identify an individual?
4. Do the same facial recognition algorithms work for other species such as dogs?

After reviewing several widely-used facial recognition algorithms, I discovered the key mechanisms behind facial recognition. After coding my own facial recognition program in Python, I was able to determine that it is possible to achieve high accuracy in facial recognition provided only a single image of a given face in the training set.

2 Introduction

To get a better understanding of how facial recognition works, I first started researching Convolutional Neural Networks (CNNs). Then, I researched the key mechanisms behind eigenface using Principal Component Analysis and Linear Discriminant Analysis. Finally, I researched whether facial recognition algorithms could be applied to other species (specifically dogs) as well as humans. As facial recognition will become ubiquitous in the future, it is important that we build efficient facial recognition programs for both humans as well as other species. I centered my research on Low-Shot Learning, which could be

used to revolutionize facial recognition with its extreme efficiency. I proceeded to code a facial recognition program in Python and test this program with several sample images.

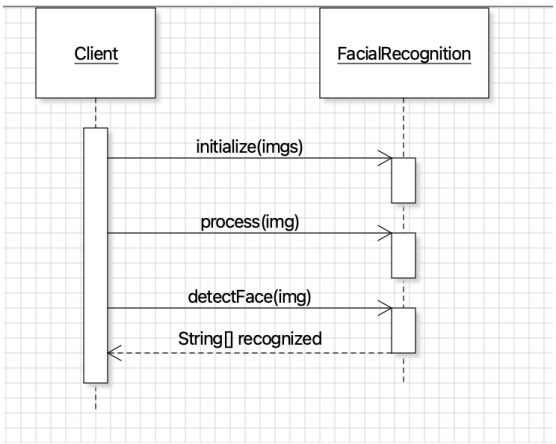
3 Materials and Methods

My deliverables schedule written before my research can be outlined as follows:

May: Review and document previous research
June: Finish research, start coding own facial recognition software
July: Finish facial recognition software, document in report

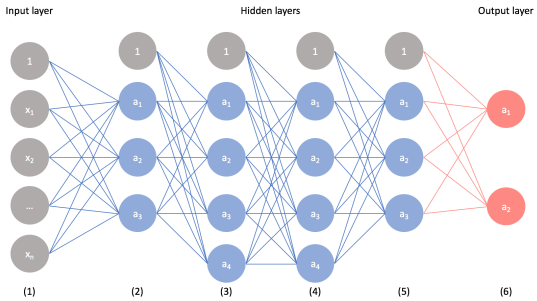
My facial recognition software will take in several images as input. After these images have been initialized and processed into the system, I will then test my program on a new set of images to see if the program recognizes the faces in the pictures previously entered as input. I will document each step of this process in detail.

There will be 3 functions in my facial recognition program: initialize(), process() and detectFaces(). First, initialize() will be called. Initialize() will initialize the images that are passed as an argument to this function. Each of these images should be assigned the name of the individual in the picture. Then, process() will process (convert into a form the program can recognize) the test images. Finally, detectFaces() will use this information to identify the individuals in a test image passed as an argument. It will output an array which contains the names of the individuals in that image. Please see the below UML sequence diagram describing this:



4 Results

Convolution Neural Networks (CNNs) are widely used in Computer Vision, especially in facial recognition. There are typically many layers in a Convolutional Neural Network, but we will mainly focus on 3 layers: the convolutional (or input) layer, the pooling (or hidden) layer, and the output layer (Ghosh, Sufian, Sultana, Chakrabarti, & De, 2020). See the image below illustrating this:



(Jordan)

The convolutional layer is "a set of convolutional kernels (also called filters), which get convolved with the input image (N-dimensional metrics) to generate an output feature map" (Ghosh et al., 2020, p. 6). Before starting the convolutional process, we transform the image into a matrix in which letters replace the intensity in color at every pixel. Then, we pick a kernel and a stride. The kernel we use will be a $n \times n$ matrix which we will use to take a dot product with the image matrix at each stride. After we are finished convulving the image, we will have extracted some key features from the image (such as vertical or horizontal edges) which can be outputted to the next layer of the CNN. Here is an example of convulving the image matrix $\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 3 & 8 & 9 \end{bmatrix}$ with kernel $\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$ and stride $= 1$.

$$\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} = 10 \text{ (first stride)}$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 \\ 3 & 8 \end{bmatrix} = 32 \text{ (second stride)}$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 3 \\ 8 & 9 \end{bmatrix} = 47 \text{ (third stride)}$$

Therefore, we end up with the convulved matrix below:

$$\begin{bmatrix} 10 & 32 & 47 \end{bmatrix}$$

.

After convulving our image, we would often like to reduce our convulved matrix to a more simplified version which contains most of the same characteristics. We can accomplish this using a technique called "pooling." There are many types of pooling techniques, but we will examine one of the most common types: max pooling. Max pooling involves taking the maximum value of all the entrees in the matrix at each stride. For example, using max pooling on the matrix $\begin{bmatrix} 1 & 2 & 4 & 3 \\ 0 & 3 & 8 & 9 \end{bmatrix}$ and taking stride = 1, we get:

$$\max \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} = 3$$

$$\max \begin{bmatrix} 2 & 4 \\ 3 & 8 \end{bmatrix} = 8$$

$$\max \begin{bmatrix} 4 & 3 \\ 8 & 9 \end{bmatrix} = 9$$

Therefore, we get the final max pooling matrix below:

$$\begin{bmatrix} 3 & 8 & 8 \end{bmatrix}$$

After convulving the image matrix and pooling, we end up at the last layer of the CNN: the Fully Connected (FC), or output layer. This layer generates the final output of the CNN which classifies the image. From there, we can use loss functions to calculate error in the classification.

Another common tool used for facial recognition is known as *eigenface*. The eigenface method is compromised of three key aspects: Principal Component Analysis, the eigenvector, and the face space (Lee-Morrison, 2019).

Principal Component Analysis (PCA) is used to determine the eigenvectors that "display the degree of variability or deviation between facial characteristics and an average" while reducing the dimensions of the images (Lee-Morrison, 2019, p.72). PCA works by projecting the data onto a subspace of orthogonal axes such that the new dimension $\leq$ old dimension. From there, the images in the dataset are represented as linear combinations of these eigenvectors. The greater the variance in an eigenvector, the more blurred a given image (or "eigenface") will appear. As a result, since most of our images will show large variations in eigenvectors, most of our transformed images will appear blurred.

Below is an example of a set of eigenfaces:

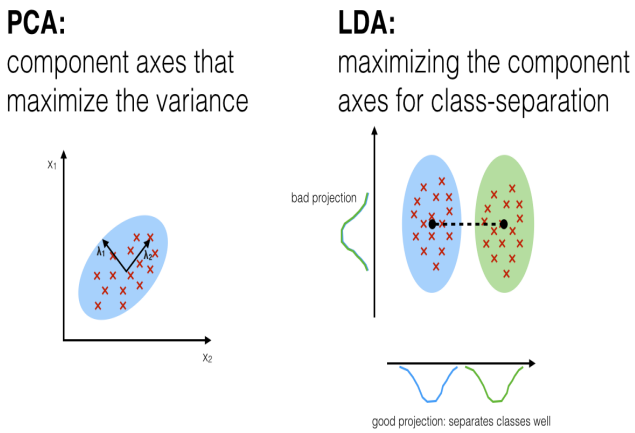
(Acar)

As mentioned above, eigenvectors are the result of performing PCA on a set of images. Eigenvectors are a measure of differences and similarities between a given face and a set of faces. Because "each eigenvector represents the greatest degree by which the facial images may vary," it is then possible to create a way of classifying faces based on their deviation from these eigenvectors.

After PCA is performed, all the eigenvectors from the sample images are projected onto a "face space". The face space is defined as "a virtual subspace that is defined and framed by the measured distances between a collection of eigenvectors." (Lee-Morrison, 2019, p.73) When a new image is to be classified, the eigenvectors of the face space are compared with the eigenvectors of the sample image, and the image is then classified into the proper category.

An alternative often used for PCA in facial recognition is called Linear Discriminant Analysis (LDA). PCA and LDA both focus on reducing dimensionality, however instead of trying to find component axes that maximize variance, LDA focuses rather on maximizing the separation between two classes (Raschka, 2014).

Below is an illustration of this difference:



(Raschka)

In order to maximize class separation, the distance between means must be maximized and the scatter (or variation) within each class X must be minimized. In other words, if μ_1 is the mean for the first class and μ_2 is the mean for the second, then

$$|\mu_1 - \mu_2|^2$$

must be maximized. Likewise, if s_1 is the scatter for the first class and s_2 is the scatter for the second class, then

$$s_1^2 + s_2^2$$

must be minimized.

The final facial recognition tool I will be discussing and my main focus is called *Low-Shot* (or *Less Than One-Shot*) *Learning*. The facial recognition algorithms described above all require several images of a given face within a training set in order for that face to be recognized later. Low-Shot Learning, on the other hand, could be used to classify faces after analyzing just a single image. As an example of its learning capabilities, given a horse and a rhinoceros in the training set, Low-Shot Learning is able to recognize a unicorn as something in-between the two (Hao, 2020).

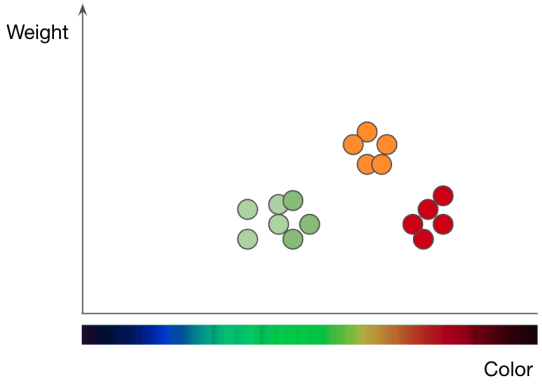
The goal of Low-Shot is to recognize N new classes given $M < N$ new examples. This classification is done with the help of "soft links". A soft link (or label) is a specific feature in an image which holds membership in several different classes. A good example of the efficacy of soft links for handwritten digits was documented in this paper by MIT researchers. It was proved that the MNIST data set, which originally contained 60,000 images, could be reduced down to just 10 images.

With the 10 "distilled" images below, an AI model could recognize handwritten digits with 94% accuracy.

(Hao)

This was well explained by Ilia Sucholutsky, a PhD student at Waterloo and lead author of Waterloo's Low-Shot research paper. He explained, rather than recognize a number as that number, it is better to use soft links to capture shared features between numbers: "instead of telling the machine, 'This image is the digit 3,' we say, 'This image is 60% the digit 3, 30% the digit 8, and 10% the digit 0.'" Using soft links, in the optimal case, we can reduce the minimal number of training images from $\mathcal{O}(N^2)$ to $\mathcal{O}(1)$ in order to separate N classes (Sucholutsky & Schonlau, 2020).

Low-Shot Learning uses soft links to classify objects with a machine-learning algorithm called "k-nearest neighbors", or "kNN." Given two classes, kNN finds characteristics (say weight and height) which differ between the two classes and plots them on a set of axes. Below, see an example of such a plot:



(Hao)

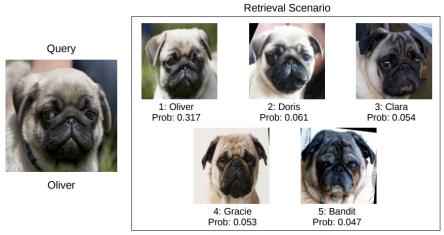
After selecting axes and plotting the data, a boundary line is drawn on the 2D chart between the two classes. The algorithm can then decide how to classify new objects based on where they lie in relation to this boundary line.

Of course, there are also some drawbacks, particularly human error, to Low-Shot Learning. As it is necessary for people to label the soft links in the data, the algorithm is subject to unconscious mistakes. Therefore, in order for this

algorithm to operate successfully, it is necessary that great precision measures are taken.

Low-Shot Learning could easily be applied to facial recognition. By identifying the soft links in images classifying faces using the kNN algorithm, it becomes easier to identify a person given only one or a few images of a given individual which may be subject to expression, hairstyle, lighting, etc. (Brownlee, 2019). Because of the ability of Low-Shot to track similar features in images, it may also be possible to use Low-Shot to classify relationships between relatives. For example, Low-Shot may have the ability to, given the photos of the mother and the father, classify whether a given image is the child.

Facial recognition for dogs could be very useful in identifying wild dogs and retrieving lost animals. Using a combination of CNNs, activation functions, pooling, and normalization, it was proved that facial recognition for dogs can also be achieved. Below are the results of an experiment researchers Thierry Pinheiro Moreira, Mauricio Lisboa, Perez, Rafael de Oliveira Werneck, and Eduardo Valle performed to successfully identify a dog, Oliver:



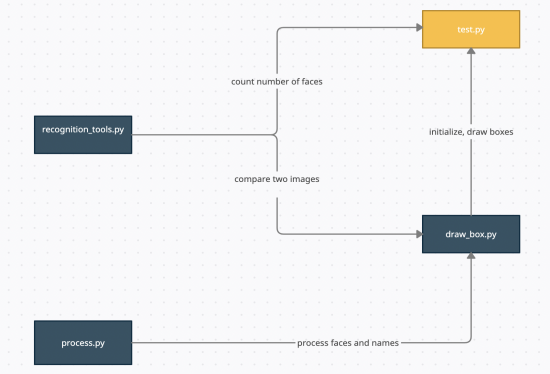
(Moreina, Lisboa, Werneck, & Valle, 2016).

After doing this research, I starting coding my own facial recognition program with a key question in mind: are many pictures of an individual required in a data set for relatively accurate recognition of that individual? Seeking to utilise something similar to Low-Shot Learning, I decided to write my facial recognition program in Python using the *face_recognition* module. The face_recognition module uses an algorithm very similar to eigenface by first mapping the faces to faces to vector space and then checking for distinguishing features of these faces. More precisely, this algorithm "maps an image of a human face to a 128 dimensional vector space where images of the same person are near to each other and images from different people are far apart ('how to use dlib's face recognition tool')." Unlike the eigenface algorithm, however, this algorithm only requires one of each face rather than several in the training set. In this way, this algorithm acts like Low-Shot Learning. As a serious chess player and someone who is passionate about following top chess tournaments, I thought it would be interesting to try my facial recognition program on top chess players. I initialized some of the most famous modern-day chess players in my training set and initialized my identification set with (mostly) group pic-

tures of these players.

My facial recognition algorithm did the following: 1. Initialize and process the training images, 2. Count the number of faces in each training image 3. Initialize and process test images 4. Compare the faces in the training and test set 5. Draw boxes around recognized faces (unrecognized faces display "Not Recognized").

My program contained 4 Python modules: process, recognition_tools, draw_box, and test. Process provided the ability to process faces and names of the individuals in the training set, recognition_tools provided the ability to count the number of faces for each picture in a given directory and compare two images which have already been processed, and draw_box provided functionality for initializing known faces and known names as well as drawing a box and a name around the faces of individuals who are recognized from the training set. Please see the below box diagram illustrating how these classes operate together:



I trained my algorithm with 24 images and then, I tested my algorithm with 11 images.

Below is an example of successfully portraying 10 faces recognized in a group test image from the 2017 Paris Grand Chess Tour after initializing the training set:

Although my algorithm was not 100% accurate in identifying the faces in the training set, I achieved accuracy about 98% of the time. I believe to achieve accuracy closer to 99%, an algorithm closer to One-Shot Learning must be used.

5 Discussion

In conclusion, in analyzing Convolutional Neural Networks, eigenface, and Low-Shot learning, I discovered the exact mechanisms each of these algorithms

used to recognize a person and distinguish them from other individuals. Although most facial recognition algorithms required many images of an individual in the training set, Low-Shot Learning could correctly identify individuals after only analyzing a single image in the training set. I showed that recognition after analysis of a single image is possible using the *face_recognition* module in Python. After initializing a single image of each individual, I was able to achieve about 98% accuracy in recognizing those individuals. Lastly, I discovered that facial recognition is not only possible for humans, but also possible for other species such as dogs.

6 Literature Cited

Ghosh, Anirudha, et al. "Fundamental Concepts of Convolutional Neural Network." *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, edited by Balas, Valentina E., et al., Springer, 2020, pp. 519-567.

Jordan, Jeremy. "Convolutional Neural Networks." *Jeremy Jordan*, 26 July 2017, https://www.jeremyjordan.me/convolutional-neural-networks/.

Lee-Morrison, Lila. "Portraits of Automated Facial Recognition", *Bielefeld: transcript Verlag*, 2019. https://doi.org/10.14361/9783839448465.

Acar, Nev. "Figure #4." *towards data science*, 21 August 2018, https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184.

Raschka, Sebastian. "PCA vs. LDA." *sebastianraschka*, 3 August 2014, https://sebastianraschka.com/Articles/2014_python_lda.html.

Raschka, Sebastian. "Linear Discriminant Analysis", *sebastianraschka*, Sebastian Raschka, 3 August 2014, https://sebastianraschka.com/Articles/2014_python_lda.html.

Hao, Karen. "A radical new technique lets AI learn with practically no data." *The New York Times*, 22 May 2007, https://www.technologyreview.com/2020/10/16/1010566/ai-machine-learning-with-tiny-data/. Accessed 8 July 2020.

Hao, Karen. "The 10 images "distilled" from MNIST that can train an AI model to achieve 94% recognition accuracy on handwritten digits." *MIT Technology Review*, 16 October 2020, https://www.technologyreview.com/2020/10/16/1010566/ai-

machine-learning-with-tiny-data/.

Hao, Karen. "Plotting apples (green and red dots) and oranges (orange dots) by weight and color." *MIT Technology Review*, 16 October 2020, https://www.technologyreview.com/2020/10/ machine-learning-with-tiny-data/.

Sucholutsky, I., and M. Schonlau. "'Less Than One'-Shot Learning: Learning N Classes From M < N Samples". *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, May 2021, pp. 9739-46, https://ojs.aaai.org/index.php/AAAI/article/view/17171.

Brownlee, Jason., "One-Shot Learning for Face Recognition ", *Machine Learning Mastery*, Deep Learning for Computer Vision, 11 June 2019, https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face -recognition/.

Moreira, T. et al. "Where is my puppy? Retrieving lost dogs by facial features." *Multimedia Tools and Applications* 76 (2016): 15325-15340.

Moreira, T. et al. *Fig.7.* 2017. *Where is my puppy? Retrieving lost dogs by facial features*, Moreina, Verlag, Springer. 2017.

General Information on the New York Mets. *dlib.net*, dlib.net, http://dlib.net/face_recognition.py.html.

"Paris GCT." *Paris Grand Chess Tour*, Paris Grand Chess Tour, 2017, https://grandchesstour.org/2017-grand-chess-tour/paris.