ARTIFICIAL INTELLIGENCE: FACIAL RECOGNITION & THE ART OF SMALL DATA

Computer Vision (Face Recognition): Haar Wavelets, LBPH, & CNNs – the Manifold Defined in n-Dimensional Timespace

John Quinn
July 30, 2018

1. Project Definition

1.1. Project Overview

Facial recognition ("FR") has become nearly commonplace in 2018. With the introduction of the iPhone X, which can identify the phone's owner-user via his or her face, many avail themselves to this technology simply by reaching into their pockets. FR is an application of computer vision ("CV"), and CV, in turn, is a branch of Artificial intelligence ("AI"). As such, it's no mean feat.

Progress in CV, first posited at MIT fifty-five years ago (Roberts, 1963), has moved apace. Over the past handful of years, AI, including FR, has reached new heights, driven, at least in part with Google making open source TensorFlow ("TF"), the concurrent widening of the use of deep learning ("DL") techniques, and the resultant tremendous results (Isola, Zhu, Zhou, & Efros, 2017).

In this project, I track major advances in FR and apply them sequentially. In the process, I start with the Eigenfaces example, a rite of passage for anyone studying machine learning ("ML") and a built-in dataset with solution code (Pedregosa, et al., 2011). I then move through an intermediate step, ultimately culminating in a DL solution. With each newer technology I better the results from the prior.

1.2 Problem Statement

The Eigenfaces problem and database[1] includes 1,288 samples, pictures stored as JPEG files. By default, the database is downloaded, and only a person with at least seventy photos in his folder-directory is attempted to be classified. Each a notable male political figure of the era, the 1990s, this Labeled Faces in the Wild ("LFW") database (Learned-Miller, Huang, Haoxiang, & Hua, 2017) has been adopted and standardized to teach entry-level FR techniques. On any given attempt/run of the solution, the overall accuracy and related measures tends towards 80% - 85%. In my submission, I attempted to and have bettered those results with other technologies, ultimately adding 10% with DL.

1.3 Metrics

I have endeavored to achieve the best overall *precision* as my primary goal. Precision, also known as positive predictive value ("PPV"), is defined as (Szeliski, 2011):

Precision = PPV = $\frac{TP}{TP+FP}$

[1] http://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_lfw_people.html#sklearn.datasets.fetch_lfw_people

TP stands for true positive and in this context represents the number of times an image was selected and was correctly classified as the subject (i.e., the person's name.)

FP stands for false positive and here represents the number of times an image of someone else was improperly classified as the subject (i.e., someone else was falsely recognized and identified as the subject/person's name.)

PPV of 100% would mean that both: (i) each image of a person was recognized and classified as him or her and (ii) no image of another person was classified – incorrectly – as him or her. Overall (i.e., for all images tested) precision of 100% would mean that all of each person's images were recognized and properly classified as him or her and that no image of another person was improperly classified as him or her.

By including FP in the denominator, PPV includes a punitive component. For this project I have supposed a use case such as recognizing a customer by his or her image. As some take offense to being confused with others, precision takes into consideration both recognizing such a customer and not mistaking him or her for another.

2. Analysis

2.1 Dataset/Data Exploration

The entirety of the dataset includes several thousand persons and many more images. For the initial step, Eigenfaces, a default fetch_lfw_people method, however, selects images only of persons with at least seventy images each.

Throughout this project, I have worked only with this subset of LFW. This results, among other things, in 1,288 images of seven men and class imbalance (i.e., measurably more images of certain individuals than of others.)

Each of the photos is 250 px x 250 px with each pixel represented by a Blue-Green-Red ("BGR") color vector, a 3-element vector for each of these color channels. Each such B, G, or R value is stored as an integer in the range [0, 255]. This allows for the maximum 256 number of integers that can be encoded in this efficient, albeit somewhat anachronistic 8-bit/byte (i.e., $2^8 = 256$). A higher integer value represents a higher intensity of such channel/color component. The photos are all .JPEG files.

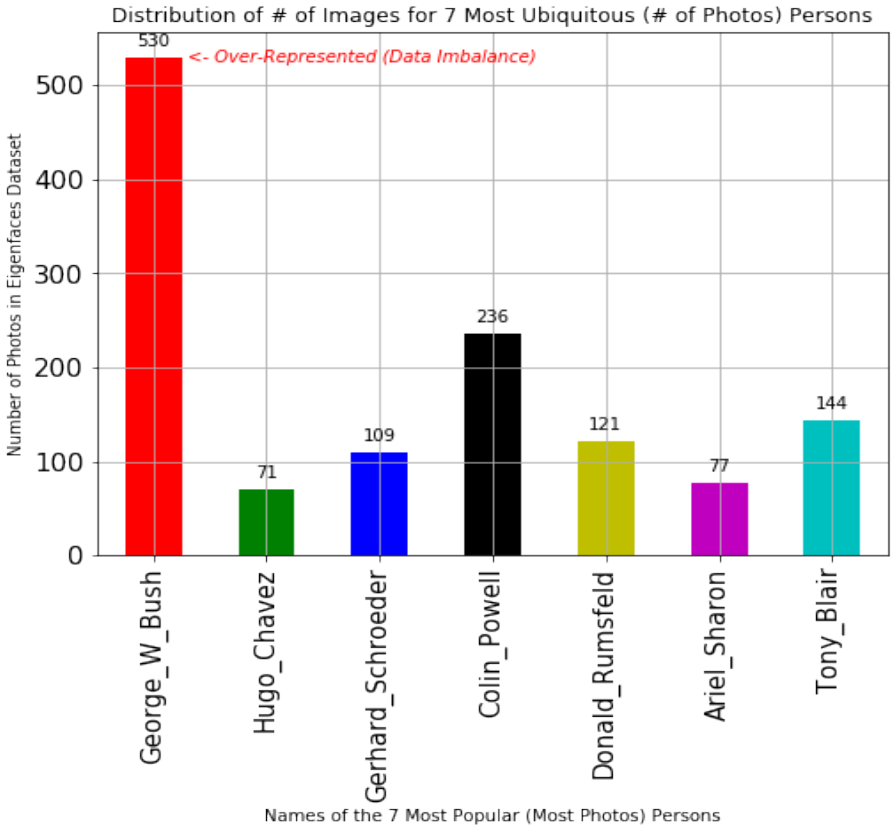
A sample of the photos can be seen below. They have been converted to RGB color scheme so that they appear normal, not Martian (i.e., mostly blue) on the viewer's screen.

A Simple Selection of RGB Images

Interestingly, when working with the unvarnished images there seemed to be too much information/variance in the images. Pores, for example, seemed to obfuscate the true essence of each face set/class, and that is why I implemented my **denoise_img** function noted in Section 3.1.2 below. The crux of this function is employing the OpenCV function detectMultiScale(gray_img, 1.1, 6)

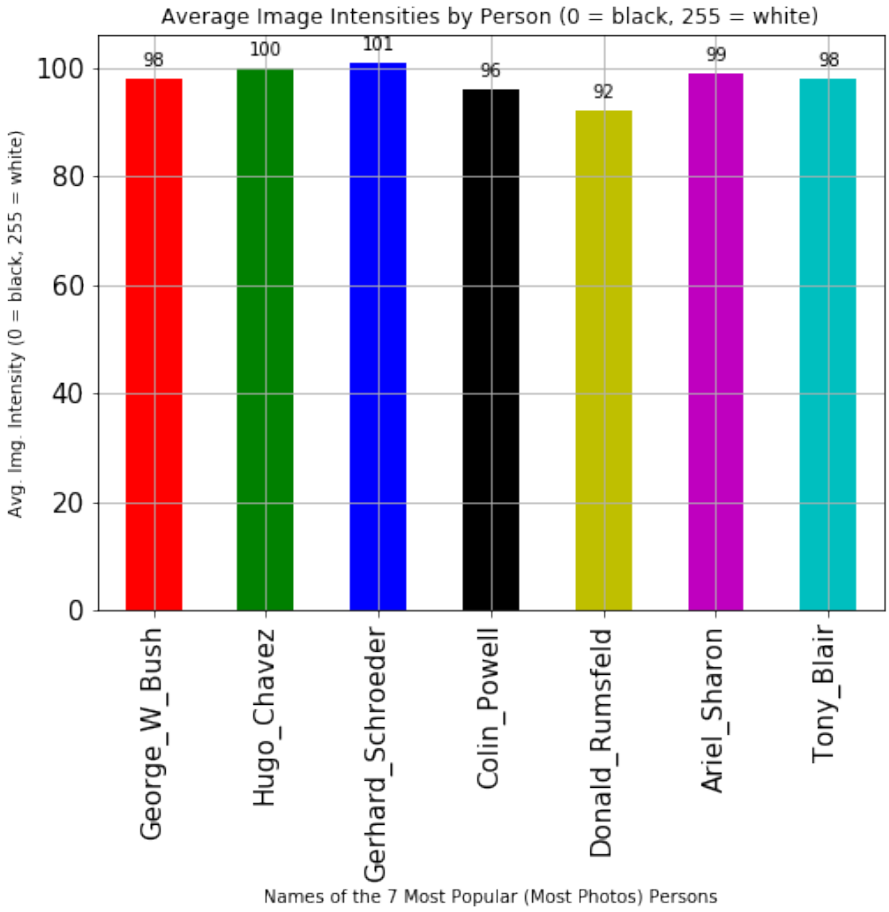
2.2 Exploratory Visualization

Below is a graph of the distribution/number of images per person:



From the graph above and from the sample of the images in Section 2.1, one can see that to compound the challenge of data imbalance the seven classes/persons are a relatively homogenous group, considering age, gender, clothing, and visceral appearance. This visceral Impression of relative homogeneity (of pixel intensity across classes/persons) is borne out by the below graphical analysis.

Below is summary of the mean monochromatic pixel intensities by person:



Finally, below are some summary statistics regarding all such photos:

Count	7.000000
Mean	97.714286
Std	2.984085
Min	92.000000
25%	97.000000
50%	98.000000
75%	99.500000
Max	101.000000

These summary statistics comport with the table above, and ostensibly the same results were obtained on an RGB basis. There is little overall pixel intensity variance between the persons/classes.

Notwithstanding the fact that five of these persons are believed to be white, Anglo-Saxon Protestants, the other two persons showed little differentiability from the other five in this respect. Colin Powell, e.g., is believed, at least in part, to be African American, and Hugo Chavez is thought to be Latino. The above pixel intensity dispersion, however, provides few associated clues.

In simply comparing the first two images in Section 2.1, one might make some conjectures to explain this seeming lack of data diversity. Those first two images are both of George_W_Bush, and the second is markedly brighter, in particular more noticeably red, than the first. Variable ambient conditions, such as lighting and temperature, likely played a substantial role in averaging pixel intensity across subjects/persons.

2.3 Algorithms and Techniques

2.3.1 The canonical Eigenfaces problem

In this Jupyter notebook is scikit-learn's own solution. This run resulted in typical overall precision of 83% in Cell 7.

The solution is a relatively straightforward one. To appreciate it, one needs, however, to understand some linear algebra[2], specifically as it relates to eigenvectors-eigenvalues, as well as some of the related principal component analysis ("PCA".)

Pictures – really CV generally – can be quite computationally expensive, meaning that many machine operations-calculations need to be performed by the processor in order to reach a result. This is so because a picture is represented electronically as a rank-3 tensor with many elements, and pictures sets are represented as rank-4 tensors.

The reason for the high cardinality of elements, even for one image, is that pictures are digitized by pixel, and each pixel, unless it's monochromatic, is, in turn, represented by a vector of color channels. The LFW pictures are a representative example.

Each image is stored as 250px x 250 px, each with its associated BGR color channel vector. In other words, each image is 250 pixels wide and 250 pixels high, and each pixel is represented by 3 channels in depth.

With the abstraction from such banalities as binary math – further that a zero is represented electronically as an "off" charge while a 1 is an "on" charge – one can quickly lose track of the engineering; inside the machine is still just circuits, charges, etc. Were one to treat each such byte from an unvarnished picture from the LFW database, each image would be represented as 187,500 dimensions (250 px wide x 250 px high x 3 color channels.)

In the Eigenfaces solution, the authors take a few steps to simply the problem and reduce such a curse of dimensionality. The first is simply an image pre-processing technique to reduce, in CV terms "resize", each image. The software libraries provided do this automatically. One must keep in mind, however, that there is an averaging effect, in essence retaining in the lesser-sized picture's byte representations an average of its surrounding pixels, with some resultant loss of detail.

The second step taken by the Eigenfaces solution is to convert the color image to monochromatic. This does, of course, reduce the complexity of identifying and quantifying the underling patterns that comprise, e.g., the typical George_W_Bush image. Furthermore, black-and-white images both comport with this older, shallow learning technique while concurrently eliminating what can be considered distractions, hue changes between a subject's images. Loss of color, however, inevitably results in loss of ultimate attainable accuracy and

[2] Hereinafter, I will assume at least some facility with mathematics and statistics.

precision. This loss is analogous to failing to consider time (i.e., the fourth/temporal dimension of timespace.) The right place at the wrong time makes all the difference.

The final trick employed by the Eigenfaces approach is PCA. PCA's primary raison d'être is to reduce dimensionality further still. If there exists, e.g., an eigenbasis for an image(s), then one can often use only certain of the information encoded in the image from such transformed representation of that image. By putting the matrix in pure row echelon form, one has obtained the eigenvalues along the primary trace of this diagonalized transformation matrix. Those eigenvalues with the highest absolute values, each together with its corresponding eigenvector, can be used in descending order (of absolute value) to define the numerically-encoded image. A larger eigenvalue tells us that its eigenvector is more telling of the image (i.e., encodes more of the image's overall variance.) As such, picking such top few eigenvalues-eigenvectors is often sufficient to capture the essence of the image and can allow for substantial reduction in dimensionality.

In this solution, the images were resized to 40% of their original 250px x 250 px size. The solution then made use of the top 1,850 features/principal components. The confluence of the two resulted in solid results with a dimensionality reduction of over two orders of magnitude, no mean feat and a great simplification computationally.

The Eigenface solution employs a basic support vector machine ("SVM") as its classifier on such dimensionally-reduced data/images. An SVM is a stochastic algorithm that finds boundaries separating the classes/persons by finding those cases closest to the boundary on known/training data. If there is a perfect, or near-perfect linear separation/separator of the classes, the SVM simply defines such n-dimensional line where n is at most the number of dimensions in the dataset, which has already been transformed. If, on the other hand, such an n-dimensional separator doesn't exist, then the SVM employs the kernel trick, which can transform the data again, this time into a higher-dimensional (i.e., of dimension > n)[3] hyperplane. In such a hyperplane the separator of dimension greater than n can find a boundary between the classes (i.e., a high-dimensional line that defines what is a George_W_Bush, a Colin_Powell, etc.)

The optimal parameters for the SVM were found via grid search. The putative optimal combination so found was:

C = 1,000, cache_size = 200, class_weight = 'balanced', coef0 = 0.0,
decision_function_shape = 'ovr', degree = 3, gamma = 0.001, kernel = 'rbf',
max_iter = -1, probability = False, random_state = None, shrinking = True,
tol = 0.001, verbose = False

2.3.2 The Haar cascade approach

The Haar cascade approach was developed largely by viewing an image as Haar wavelets. (Viola & Jones, 2004). This is another transformation/basis change. Its efficiency is not to be understated.

The Haar transformation allows for a solution (i.e., a defining function, in the instant case of these seven persons' images) that is orthonormal and Lebesgue integrable

[3] Interestingly, unlike quite a number of problems faced in real analysis, the hyperplane for this SVM transformation can even be of infinite dimensions.

(indirectly) in the transformed Banach space[4]. Altogether this allows for function finding (i.e., discovery of the underlying sequence encoding the essence of the person's face).

The imported libraries, like those for Solution 1, work only with monochromatic images. By removing color, the statistical learner can identify 1D (i.e., scalar) variations of simple image intensity of a pixel vis-à-vis its surrounding pixels.

The implementation borrows from models tried and tested, each an eXtensible Markup Language ("XML") model meant to capture the face, the nose, the mouth, etc. In each case it does so be defining the coordinates of the 2D bounding box of such object. As is common in CV, such coordinates are given in what would be the fourth quadrant of the simple Cartesian plane. The origin (0,0) and the horizontal (i.e., x) axis are unchanged, but the enumeration of the vertical (i.e., y) axis is positive, rather than negative. Thus, a pixel further below the origin is enumerated with a higher, not a lower, vertical/y coordinate.

These XML models are quite detailed and large. Each is the result of substantial testing and refinement by problem domain. Those employed can be found with the folder entitled detector_architectures, and within such files and in the Bibliography to this paper are the related citations.[5]

Because of weakness of finding facial parts, in particular, my implementation takes what would in essence be a resultant non-starter from using the Haar wavelet alone. It continues on with this class of matching algorithms from the whole of the images[6] with the same ostensible approach of identifying intensity change patterns but building up rather than down. Where the Haar cascade approach emphasizes the group of images, the Local Binary Patterns Histogram ("LBPH") emphasizes the individual image. The two really are more than complements. Each looks for the same sequences, but the former encodes them from the image set while the latter does so from each image.

2.3.3 The DL approach (naïve first)

When a data scientist faces a CV problem and seeks a DL solution for it, most commonly the ML algorithm employed is itself or at least includes a Convolutional Neural Network ("CNN"). The CNN is derived from the mathematical convolution theorem and implemented efficiently in binary-oriented computer science as follows:

$$\vec{a} \oplus \vec{b} = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(\vec{a}) \bullet \mathrm{DFT}_{2n}(\vec{b}))$$

Where:

- $\vec{a}$ and $\vec{b}$ are of length n
- n is a power of 2
- $\vec{a}$ and $\vec{b}$ are padded with zeroes to length 2n

[4] By this I do not mean that we necessarily have C^1 analytical differentiability; rather the combination of an appropriate topological manifold and numerical approximator on local charts of the atlas should suffice for a utilitarian differential structure

[5] Many are found at http://github.com/MeDzIeDaRbI/Haar-dataSets, wherein polyglot naming conventions have been employed. Mouth.xml is, of course, the English-named model for a mouth while ojoI.xml is the Spanish-named *ojo izquierdo* (left eye in English) is also included therein.

[6] More precisely, the statistical learner is actually gleaning the dominant features from the training set

- DFT and DFT^{-1} denote Discrete Fourier Transform and DFT inverse, respectively
- • denotes the component-wise product of the 2n-element vectors &
- $\oplus$ denotes the convolution itself
  (Cormen, Leiserson, Rivest, & Stein, 2009)

For at least four years, CNNs have been recognized as the go-to ML model/architecture for visual recognition (Razavian, Azizpour, Sullivan, & Carlsson, 2014).

Given the paucity of data (i.e., simple low cardinality of images), I employed transfer learning (Goodfellow, Bengio, & Courville, 2016). With CV, especially, the learner is simply looking for patterns in the numbers. A CNN in the to and fro that is convolution without some initial guidance would require in the order of several thousand images per class to incarnate anew an effective classifier.

Most generally, the deep learner seeks to find an objective function mapping the sample spaces (train, validation, and test) encoded in their rank-4 tensors to differential manifolds existing in transformed tensor fields.

To start, one usually uses a standard model, in this case one that simply takes in this digitized model of matter and convolves its encoding through a series of somewhat-known and other still black-box/hidden layers to uncover this mapping. I have consistently found success in one of the simplest of these, VGG-16, a pre-trained model already built into Keras and similar architectures built atop TensorFlow ("TF").

VGG-16 fairly regularly ranks high in the top-1 and top-5 (i.e., 1st image is a match, 1 or more of the 1st five images are a match) and is an excellent finding of the Visual Geometry Group ("VGG") at Oxford University. (Ilango, 2017). VGG-16, furthermore, includes the lowest depth/fewest total layers, 23, reducing the opacity we hope one day to unravel. [7] VGG-16 does include a relatively high number of parameters, but at least n many more nodes-parameters are of lesser indiscernibility than more layers. The former simply means more variables/components. The latter infers more transitions (more easily contemplated from the perspective of discretized, parameterized time) or more dimensions to understand within spacetime. If one were to choose a DL architecture based upon Occam's Razor, VGG-16 would be the one selected due to its parsimonious depth.[8]

I was doubly fortunate in researching for a model to start the transfer learning to find a version of VGG-16 specifically optimized for facial recognition for Keras, keras-vggface (Malli, 2018). It is a complete implementation and adaptation of the project

[7] Ibid

[8] As a matter of pure, but certainly interesting, conjecture one might compare VGG-16's architecture (http://www.researchgate.net/figure/VGG16-architecture-16_fig2_321829624) to some views of timespace from theoretical physicists. From that architecture, one could remove any red (max pooling) layer, as it's simply a mathematical means to reduce dimensionality from preceding layers. Similarly, one could remove the final, softmax layer, as that too is simply a mathematical tool to emulate electronically the confluence of neurotransmitters and the associated neuronal electric response/transmission in the human brain. Finally, one could remove VGG-16's three fully-connected + ReLu layers just before the softmax layer, as these three layers simply represent the model's rendering of matter in the transformed 3D world. This would leave 11 truly hidden layers within VGG-16. Interestingly, string theorists argue that the world must include more than three dimensions and argue most frequently about 10 or 11 before devolving into the quantum world. (Witten, 1995)

undertaken by VGG using different technologies and platforms, e.g., MATLAB, Torch, etc. (Parkhi, Vedaldi, & Zisserman, 1995).

My first incarnation of employing transfer learning and the specific benefits of keras-vggface, Solution 3/Part 3, was decidedly naïve. Apart from some data wrangling, data structure conversions, and the like, I simply took all 1,288 photos, ran the non-testing sets through keras-vggface for the bottleneck features, and took such "blobs" as input to the top model.

The design of the top-level model is simple and intuitive, but I did a good bit of testing on optimizers and their parameters. The parameters, with optimizer Adam, are:

learning_rate = 0.001, epochs = 50 , beta_one = 0.9, beta_two = 0.999,
epsilon = 1e-4, decay_rate = 0.025, amsgrad_bool = False, batch_size = 16

The overall architecture of the top model are as follows:

```
top_vgg_f_model = Sequential()
top_vgg_f_model.add(Flatten(input_shape=codes_train_bn.shape[1:]))
top_vgg_f_model.add(Dense(512, activation='relu'))
top_vgg_f_model.add(Dropout(0.5))
top_vgg_f_model.add(Dense(256, activation='relu'))
top_vgg_f_model.add(Dropout(0.5))
top_vgg_f_model.add(Dense(n_classes, activation='relu'))
top_vgg_f_model.summary()
```

2.3.4 Refinement (the final DL approach)

The final incarnation is found in Solution 3A/Part 3A. In it, I employ:

- the denoise_img function,
- balancing the datasets using only under-sampling (with the first 70 images/class),
- refinement to the Adam optimizer, increasing ε to 1e-2 and epochs to 100

2.4 Benchmark

Again, the benchmark I have used throughout was precision. Specifically, I have sought with each iteration to beat the roughly 83% precision typically yielded by the canonical Eigenface approach (i.e., Solution 1/Part 1.) I have succeeded in this endeavor and, in fact, improved such result/metric with each new approach/subsequent Jupyter notebook.[9]

[9] In this regard, Solution 3/Part 3 should not itself be viewed as an incremental step. Rather, Solution 3/Part 3 should be considered only as a prelude/practice for Solution 3A/Part 3A. The latter, but not the former, does, in fact, improve upon the results of Solution 2/Part 2.

3. Methodology

3.1 Data Preprocessing

3.1.1 Eigenfaces

Solution 1 does much pre-processing on its own, as noted above. This Eigenfaces approach, nonetheless, is presented without change as a benchmark and starting point for the later approaches.

From Solution 1, I have saved as pickle files certain data useful for later Solutions. The Solution 1 (i.e., Eigenfaces) code, moreover, downloads the images, by default storing them on the user's home directory. These default images, in addition, are stored within directories/folders whose names are the labels of the so-enclosed images. This is common practice for CV and helpful for other, especially big-data implementations.

The default Solution 1, moreover, stores the original files as funneled images. By this, the faces are aligned (i.e., mostly centered and forward facing, rather than, e.g., profiles.) I have used such funneled image throughout later Solutions.

3.1.2 Haar Cascades

With Solution 2/Part 2 I took the funneled images from Eigenfaces for the top 7 persons and was able to implement/find face bounding boxes but not facial part bounding boxes. While many of the XML models were provided by Scikit-Learn itself, along with the dataset, others were found from persons trying to perfect facial part models, especially (Castrillon-Santana, 2016).

With the ability to find a face bounding box, I used its coordinates. Specifically, I used the associated region of interest ("ROI") of the image for this model.

I also used the denoise_img function. That function, some helper functions it can call, and some facial feature bounding box functions not employed can be found in haar_helper_functions.py. Ultimately, this distilled information was used to better the final (for Solution 2) LBPH model. Details of the denoise_img function's parameters can be found in Section 2.1.

3.1.3 DL – naïve implementation

Again, with this simple, naïve implementation of a CNN I did little more than read in all 1,288 image files, employ transfer learning, design a simple and intuitive top-end model, and test some optimizer-parameter combination choices.

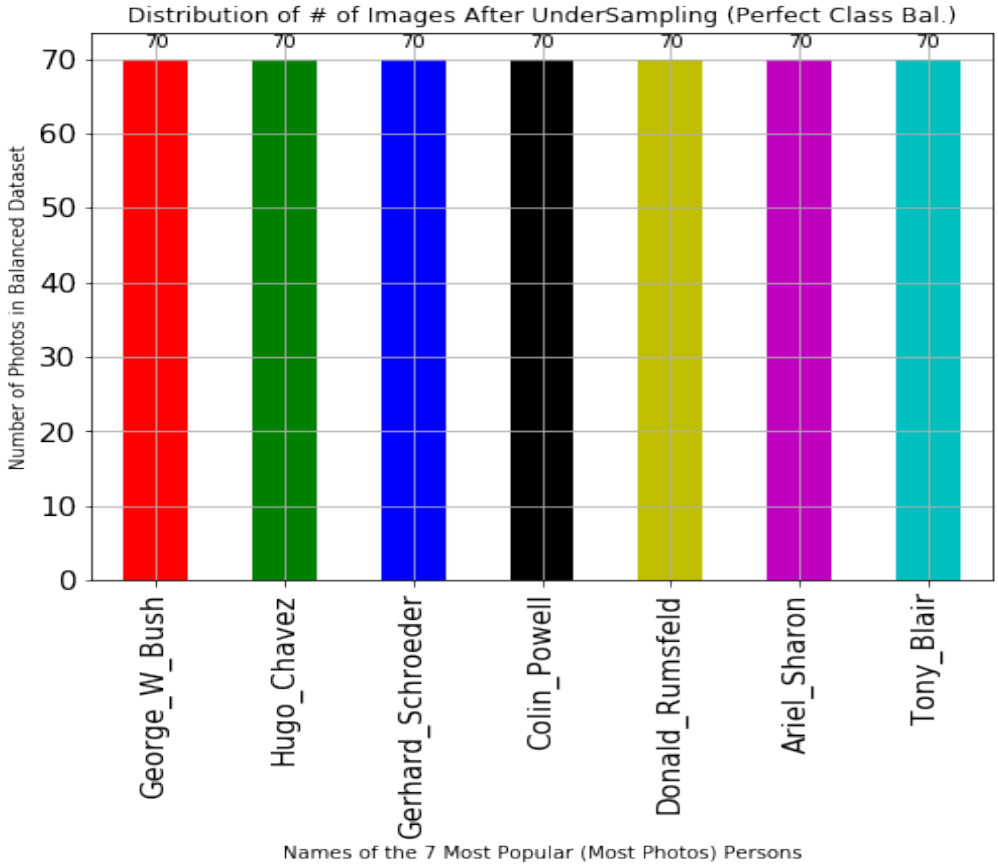
The transfer learning model, however, expects images to be of the size 224 x 224 x 3. Via examination of keras_vggface.utils.py, specifically its preprocess_input function, I was able to determine that the model expected to read in RGB files, resize them, convert them to BGR files, and "normalize" the BGR color vector for each pixel triplet by subtracting from it the average of such associated color magnitude in the model which is summoned, downloaded, and loaded into the architecture on its first use. I handled each of these with functions from the opencv_python_contrib libraries and numpy methods within the Jupyter notebook myself.

With only 7 classes, callbacks, TF as the backend and the like, all labels were one hot encoded ("OHEd".) This makes for only 0 or 1 comparison of a 7-dimension y/response variable, avoids library conflicts, and obviates, e.g., errors that could arise from string/name handling.

3.1.4 DL – final implementation

In this final implementation, I employed the preprocessing tool deemed most helpful in prior iterations, denoise_img. I also simply eliminated the class imbalance problem altogether, by selecting 70 images for each person in total for training, validation, and testing.

The resultant distribution looks as follows:



I also, of course, resized the images, converted them to BGR, and normalized each color channel as required by keras_vggface. OHE was employed here as well.

3.1.4.1 Refinement

The Section 3 architecture and optimizer were left unchanged with two exceptions. Optimizer parameter epsilon was increased to 1e-2, and epochs was doubled, to 100, to adjust for the resultant slower learning from this increased stochasticity. More details can be found below in Section 5.

4. Results

Each of these models, of course, employs stochasticity. By employing a random seed, the same search/function finding initialization, etc., one could reproduce the exact results in each of the four notebooks now.

That would provide easy verifiability, but at the likely cost of loss of generality, especially extensibility. By building a stable system that works there can be relatively minor changes in results, but overall results, especially over a handful of runs, should hold true.

Of note here, however, is the fact that Solution 3 and especially Solution 3A are vulnerable to changes in any single result, born of employing methods readily applied to big data on a dataset that could easily be called the opposite, small data. The test set of Solution 3A, e.g., includes just 49 images, and a stochastic start along a better or worse path to map the manifold could result in 1 or 2 images more being classified correctly or incorrectly than is the case in these notebooks now. That translates into a potential betterment or worsening of those results of a couple, even a few percent. In essence, due to paucity of data, stochastic modeling can yield differing, actually discrete results from one run to the other.

4.1 Eigenfaces

The Eigenfaces solution yielded 83% precision, largely as expected. A few sample runs yielded results little different from these.

4.2 Haar Cascades

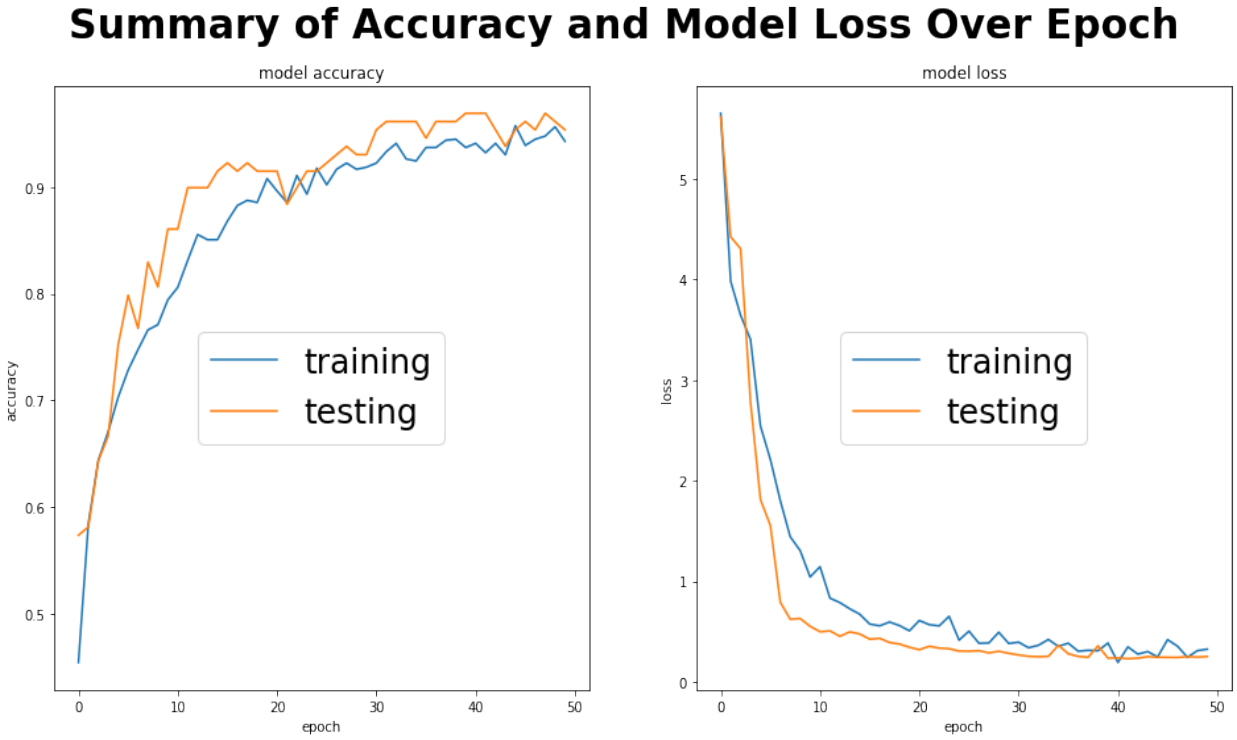
Solution 2 resulted in precision of 93%, which is impressive. Of course, with rather pristine, funneled pictures, and a shallow learning algorithm that doesn't require massive training data, this too is largely as expected.

4.3 DL – naïve implementation

Solution 3 yielded precision of 92%, not as good as Solution 3, but still impressive given:

- the tiny (from the perspective of D), size of the dataset,
- pre-processing only as needed to be fed into keras-vggface, and
- an intuitive, lightly-tested approach to top-level model architecture and parameters.

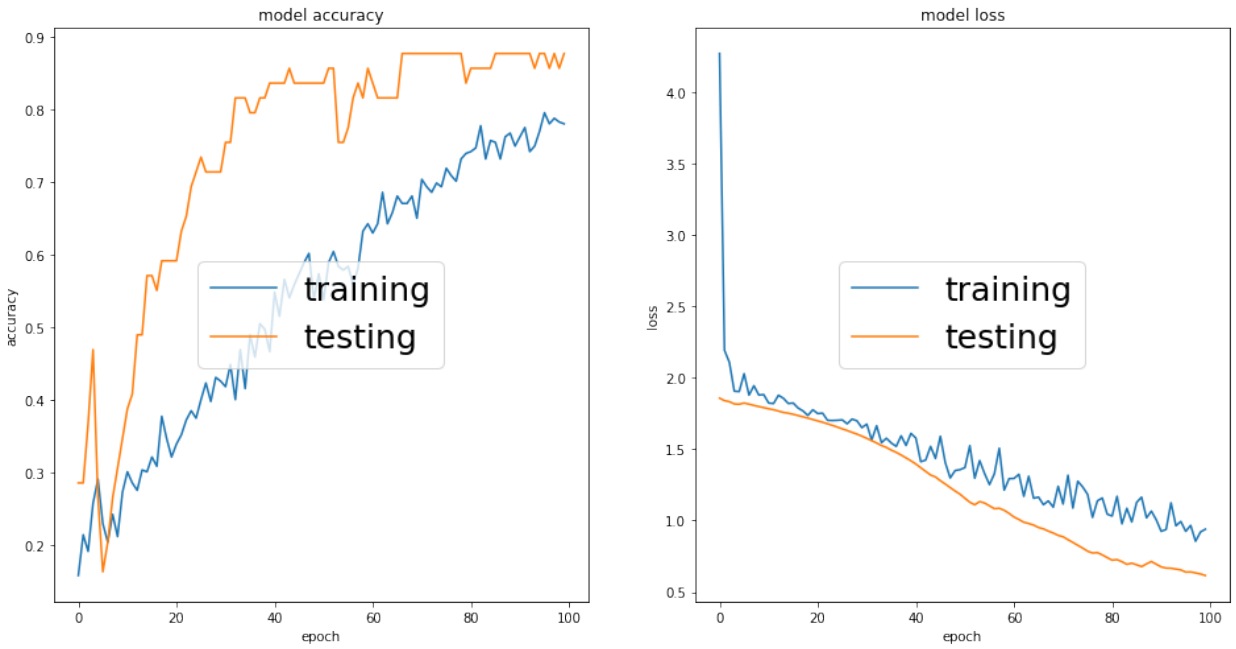
The below graphs do show that Solution 3 was convergent, relatively early in the training process at that, and that evaluation and training losses and accuracy don't appreciably diverge. This evinces little-important overtraining and ostensible lack of bias:



4.4 DL – final implementation

Solution 3A reflects 94% precision. Over a few trial runs, on average, the precision was higher still, and only on a few occasions did it fall to the lower 90s.

Below are graphs of Solution 3A like those above for Solution 3:



5. Conclusion

5.1 Model Evaluation and Validation

In the end, I have relied on one metric, precision, to compare each of the four models. In the putative use case, identifying a customer by his or her face, I believe this to be an appropriate one and better than simple accuracy, as with precision one captures both the system so recognizing the customer correctly as a positive and also captures the system falsely calling/identifying the customer as someone else. Customer Carol being greeted as Customer Gale could easily put Carol off; people generally like to think of themselves as unique and increasingly seem to want AI and personnel enabled with it to recognize them as such.

The below table summarizes the upshot of the four models and their support:

Solution/Part #	Solution Algorithm	Precision (%)	Support (# of Test Imgs)
1	Eigenfaces	83%	1,288
2	Haar Cascades	93%	1,288
3	DL – naïve implem.	91%	1,288
3A	DL – final implem.	94%	490

Inasmuch as I sought, at the outset, simply to beat the 83% precision of Eigenfaces, the benchmark, the final solution, Part 3A is more than reasonable and comports with expectations. DL, especially a CNN, should outperform a shallow learner.

The learning graphs above evince little divergence from model training and testing, both with respect to model accuracy and loss. As such, Solution 3A should generalize well for unforeseen data. The final model, moreover, was derived largely from transfer learning, employing a well-trained model designed specifically for this use case, facial recognition, atop a model created by VGG, experts in the field on CV.

By increasing epsilon, which is supported by the literature cited above, the model/system has been made more stable than would otherwise be the case. One might run into different results on this relatively tiny test set, but such should not be the case in a big data setting; changes from one run to the next should be marginal.

5.2 Justification

The final model clearly is justified. The benchmark has been handily beaten. Transfer learning from a model optimized to this specific domain was employed. Such underlying model employs a CNN, the go-to classifier for images/CV. In all respects, precision in the mid-nineties (percentage accuracy and/or precision) is perfectly reasonable when compensating for such a small dataset. Using the *entirety* of this dataset, the authors of Eigenfaces and creators of LFW itself have noted CNN's achieving just points below perfection (i.e., just below 100%) with CNNs.[10]

5.3 Reflection

I learned and employed quite a number of things, in particular in completing Solution 3A. They follow in no specific order.

In working through Haar cascades (i.e., Solution 2) I had *assumed* that ubiquitous

[10] http://vis-www.cs.umass.edu/lfw/roc-files/convnet-rbm_unrestrict_iccv13.txt

studies, code example, etc. and, even more so, the availability and fairly recent updates to XML models for facial parts automatically inferred that this technology was effective. I later learned that it is not, certainly as a standalone implementation, an effective endgame. In hindsight I wished I'd researched its effectiveness, in particular with respect to the nose, a key element in mapping a face, before setting out on this quixotic journey. This technology simply isn't that effective on its own (Castrillon-Santana, Deniz-Suarez, & Anton-Canalis, 2008).

There were a few nuances of Jupyter notebook and of TF itself that needed some time to discern, debug, and surmount, as well. Within Keras's preprocessing.image.ImageDataGenerator and related libraries there are many excellent tools, but the confluence of OpenCV, Python itself, and the like caused me, e.g., to abandon the flow_from_directory method and instead build my own solution around the simpler flow method.

Along the way in building my solution using the flow method I built other methods within the Jupyter notebooks for Solution 2 and 3, in particular, in dl_3A_helper_functions.py. Within that file are a purposefully overzealous, custom way to solve this problem that is readily extensible to harder ones. In particular, those functions allow for complete control over the selection and building of training, validation, and testing sets. Furthermore, some rudimentary other methods/functions (e.g., yield/lazy evaluation, caching with speedy Cython, etc.) could be coupled with them to apply this "small data" solution to a big data one.

The numerical Jupyter notebook Solutions/Parts are specifically designed, as well, to denote the serial nature in the building (i.e., software engineering) of the answers, ultimately culminating in Solution 3A. In addition, this allows easy comparison of the different technologies/mathematical approaches to the problem. With each step, e.g., I started putting already-solved helper functions in the helper .py files. What I did not realize is that default caching algorithms of Jupyter pre-read images of such imported helper files, so when I changed any such helper file I was at first perplexed by the odd, unchanged behavior of the "parent" Jupyter notebook. The straightforward cure for this problem can be found in the last three lines of Cell 1 in Solution 3A (i.e., dl_face_sol.ipynb)

Similarly, I quickly learned that TF, by default, allocates essentially all memory available to it, at least when run on a graphics processing unit ("GPU"). This can easily lead to out-of-memory ("OOM") and similar problems. At least one simple solution to this, causing TF to allocate GPU memory dynamically, as needed, can be found within the import statements of Cell 1 of Solutions 3 and 3A and in the calling of those functions within those two Jupyter notebooks.

In designing the DL solution (i.e., Solution 3 and, more importantly, Solution 3A) I at first wrangled with the many design options of the top-end model and then returned to the halving-funnel design noted in those solutions. In the end the KISS[11] principle proved best.

In that same vein, I tested many optimizers only to remember that it is really the confluence of a given optimizer and its parameters that matters most. While one might find a solution, e.g., 7 standard deviations more accurate and/or more precise than the mean solution, rather than 6, that may not be necessary and could, in fact, invoke problems, including production delay, overfitting, and the like. Ultimately, I consulted a concise and helpful table to deign a solution not designed for interstellar travel – a return to reality (Kristiadi, 2016).

[11] Euphemistically, "Keep It Simple, Stupid" or, to the scientist, Occam's Razor

I went with the Adam optimizer (Kingma & Ba, 2017).[12] This optimizer includes several parameters, and in the end measurable divergence from the default and recommended ones, epsilon, was required. After some fairly exhaustive research I found justification for such divergence right on TF's instruction pages (Google LLC, 2017).

This TF entry *cited above* reiterated that Adam's update rules works as follows:

t := t + 1

lr_t : = extlearningrate * $\sqrt{(1 - beta_{2)}^t / (1 - \text{bet}a_1^t)}$

m_t := beta_1 * m_{t-1} + (1 - beta_1) * g

v_t := beta_2 * v_{t-1} + (1 - beta_2) * g * g

variable := variable - lr_t * m_t / ($\sqrt{v_t}$ + ***ϵ***) (emphasis added, to ***epsilon***)

The cited entry provides more color and explanation, and its progeny (since its publication last year) go deep dive into its details. For the sake of clarity, below nonetheless is a short summary of most items above:

t: The fairly standard discretized time, or index/count of iteration

lr_t , extlearningrate: learning rate at time t, learning rate input parameter

m_t , v_t: 1st moment vector, 2nd moment vector (each at time t)

beta_1 , beta_2 : each a throttle, the 1st for the 1st moment vector and the 2nd for the 2nd moment vector for discretized-time, recursive update β_1 , β_2 notation presumably not used to comport with kwargs[13]

g: The gradient, commonly denoted $\vec{\nabla}$ (Hubbard & Hubbard, 2009)

Epsilon is little more than a tiny fudge factor to help ensure the avoidance of division by zero. While its default value is 1e-8, setting it at several orders of magnitude higher, even at 1.0, may be required. In the instant case I discovered that 1e-2 worked best and that the introduction of such increased stochasticity, almost paradoxically, stabilized the model/system measurably. Of course, doing so also slowed the learning process, so I had to double the epochs, from 50 in Solution 3 to 100 in Solution 3A.

I now provide the end-to-end ultimate problem solution (Solution/Part 3A). Details of parameters, network architecture, and the like have been excluded for the sake of concision and can be found earlier in this document, as well as within the code itself.

5.2.1 For each of the 7 classes/persons/identities I kept only 70 images. The auto-downloaded database of images showed no sign that the order of a particular image in a person's folder had any meaning, and, in fact, the numerical suffixes to those files' names belied any such sequencing. As such I simply took the 1st 70 such images for each.

5.2.2 As the files-pictures were read into memory by folder/class label, the 490 images required and were randomized and split into train (80%), validation (10%), and test (10%) sets.

5.2.3 The string labels (i.e., the persons' names) were OHEd, as with only 7 classes this adds only 6 dimensions, each of which is comprised only of a Bernoulli-distributed scalar.

5.2.4 All associated numpy arrays were then converted to rank-4 tensors (for the three image sets) and rank-2 tensors (for their associated OHEd label vector sets). Prior to such

[12] The name Adam is a concatenation of certain parts of ADAptive Moment estimation

[13] Key Word ARgumentS, i. e., arguments (when called)/parameters (when instantiated) requiring known labels

somewhat mechanical processing the images were also resized, denoised, converted to the required BGR color scheme, and "normalized" with the BGR_means vector. This vector, from the preprocess_input method of keras-vggface/utils.py, serves merely to subtract from each BGR vector element the associated average of such color for the LFW database. It is not a true normalization, inasmuch as it does not further scale the adjusted values, such as to a z-score basis, i.e., transform each, on an equivalence basis, to [0,1]. This likely is the largest contributor for increased ϵ noted above; without the imputation of such increased stochasticity, small or even zero-valued color bytes could be dominated and lost in convolution-iteration.

5.2.5 I then employed this domain-specific transfer learning. Specifically, I obtained the features ("vgg_features") as bottleneck features, obtained codes/blobs by using such bottleneck model on the train and the validation tensors, built a top-end model (Cells 15 & 16), trained that top-end model on such train and validation blobs, and then predicted classes/persons/identities with the test images.

5.3 Improvement

The model could be improved in several ways. A few ideas follow.

First, my fairly crude means of balancing the dataset via under-sampling likely could be bettered. Rather than simply chopping class cardinality to 70, with some trial runs using all those images causing boundary confusion or alternatively, those doing just the opposite could be identified, included, or excluded to see the effect on the model.

In lieu of choosing or even emphasizing under-sampling or over-sampling some simple tools could be tried, such as flattening images and using class_weights, an imperfect solution (Chollet, 2016). In most cases, the introduction of high-dimensional synthetic instances is impossible or of little benefit (Blagus & Lusa, 2013).

Second, I have not implemented any over-sampling approaches. From keras.preprocessing.image.ImageDataGenerator, e.g., adding flips, rotations, and the like might help. In keeping with this line of reasoning, I considered but passed on the creation of synthetic images. One open-source library, e.g., might provide a new and improved algorithm, SMRT (Smith, 2017). I have not tested whether this algorithm is, in fact, more effective than its predecessor, SMOTE or its many brethren published as part of the scikit-learn-contrib project (Lematre, Nogueira, & Aridas, 2018).

Third, I haven't tried other, somewhat novel set cardinality increase tools. Were one to test these, he or should could try domain-specific approaches, e.g., (Karras, Aila, Laine, & Lehtinen, 2018) or the more general statistical learning-game theory approach, especially Deep Generative Adversarial Networks (Radford, Metz, & Chintala, 2016).

Finally, I have followed a largely intuitive path in designing the top-model architecture. One might take a more systematic approach, such as grid search and/or cross-validation.

Bibliography

Blagus, R., & Lusa, L. (2013, March 22). *SMOTE for high-dimensional class-imbalanced data.* Retrieved from BMC Bioinformatics: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-106

Castrillon-Santana, M. (2016, August 25). *Github, Inc.* Retrieved from MeDzIeDaRbI/Haar-dataSets: http://github.com/MeDzIeDaRbI/Haar-dataSets

Castrillon-Santana, M., Deniz-Suarez, O., & Anton-Canalis, L. (2008). Face and Facial Feature Detection Evaluation - Performance of Haar Detectors for Facial and Facial Feature Detection. *VISAPP (International Conference on Computer Vision Theory and Application).* Las Palmas de Gran Canaria, Spain: Institute of Intelligent Systems and Numerical Applicatioins in Engineering.

Chollet, F. (2016, August 31). *Keras - how to use class_weight with 3D data #3653*. Retrieved from Github, Inc.: http://github.com/keras-team/keras/issues/3653

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, 3rd Edition.* Cambridge, London, Mass., England, USA, UK: The MIT Press.

Goodfellow, I., Bengio, Y., & Courville, a. (2016). *Deep Learning.* Cambridge, London, Mass., England, USA, UK: The MIT Press.

Google LLC. (2017, October 25). *tf.train.AdamOptimizer*. Retrieved from TensorFlow: http://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

Hubbard, J., & Hubbard, B. B. (2009). *Vector Calculus, Linear Algebra, and Differential Forms.* Ithaca, New York, USA: Matrix Editions.

Ilango, G. (2017, March 20). *<gi/>*. Retrieved from Using Keras Pre-trained Deep Learning models for your own dataset: http://gogul09.github.io/software/flower-recognition-deep-learning

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017, November 22). *Image-to-Image Translation with Conditional Adversarial Networks.* Retrieved from Cornel University Library: arXiv.org

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018, April 30). *Progressive Growing of GANs for Improved Quality, Stability, and Variation.* Retrieved from nvidia research: http://research.nvidia.com/publication/2017-10_Progressive-Growing-of

Kingma, D., & Ba, J. (2017, January 30). *Adam: A Method for Stochastic Optimization.* Retrieved from Cornell University Library: http://arxiv.org/abs/1412.6980

Kristiadi, A. (2016, June 22). *Beyond SGD: Gradient Descent with Momentum and Adaptive Learning Rate*. Retrieved from Augustinus Kristiadi's Blog: http://wiseodd.github.io/techblog/2016/06/22/nn-optimization/

Learned-Miller, E., Huang, G., Haoxiang, L., & Hua, G. (2017, May 9). *Labeled Faces in the Wild*. Retrieved from University of Massachusetts, Amherst: http://vis-www.cs.umass.edu/lfw/

Lematre, G., Nogueira, F., & Aridas, C. (2018, June 13). *scikit-learn-contrib/imbalanced-learn*. Retrieved from Github, Inc.: http://github.com/scikit-learn-contrib/imbalanced-learn

Malli, R. C. (2018, March 26). *Github, Inc.* Retrieved from rcmalli/keras-vggface: http://github.com/rcmalli/keras-vggface

Parkhi, O. M., Vedaldi, A., & Zisserman, A. (1995). Deep Face Recognition. *British Machine Vision Conference.* Oxford: University of Oxford, Dept. of Engineering Science, Visual Geometry Group.

Pedregosa, F., Varaquaux, G., Michel, V., Thirion, B., Grisel, O., & Blondel, M. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825-2830.

Retrieved from Scikit-learn: Machine Learning in Python: http://scikit-learn.org/stable/about.html#citing-scikit-learn

Radford, A., Metz, L., & Chintala, S. (2016, January 7). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* Retrieved from Cornell University Libraries: http://arxiv.org/pdf/1511.06434.pdf

Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (pp. 806-813). Stockholm: Computer Vision Foundation.

Roberts, L. (1963). *Machine Perception of Three-Dimensional Solids.* Massachussets Institute of Technology, Electrical Engineering. Cambridge, Massachussetts: ResearchGate.

Smith, T. (2017, June 22). *Synthetic Minority Reconstruction Technique (SMRT)*. Retrieved from Github, Inc.: http://github.com/tgsmith61591/smrt

Szeliski, R. (2011). *Computer Vision Algorithms and Applications.* Redmond: Springer-Verlag London Limited.

Viola, P., & Jones, M. (2004, May). *Rapid Object Detection Using a Boosted Cascade of Simple Features.* Retrieved from Mitsubishi Electric Research Laboraties: http://www.merl.com/publications/docs/TR2004-043.pdf

Witten, E. (1995). *Nuclear Physics B* (Vol. 443). Princeton, New Jersey, USA: Elsevier B.V.