A Survey of Facial Recognition Methods

Jonathan Frankle and Caroline Morganti
Massachusetts Institute of Technology
jfrankle@mit.edu morganti@mit.edu

1. Introduction

In this project, we explored both classical and modern algorithms for facial recognition. We researched, designed, and implemented three variants of algorithms from the 1990s and three variants of modern convolutional neural networks (CNNs). In this paper, we report our results on the Labeled Faces in the Wild (LFW) dataset. We also discuss several approaches that did not succeed.

Background. Facial recognition has been a popular machine vision task since the first successful algorithms were developed in the early 1990s. In recent years, as research has matured and found its way into industry, facial recognition has surfaced in a wide variety of commercial applications. Today, it can be found in security-sensitive environments (identity verification), retail settings (classifying and tracking customers), and law-enforcement workflows (identifying suspects).[1] The first successful, fully-automated facial recognition system was the *Eigenfaces* [13], which used Principal Component Analysis (PCA). A follow-up approach called *Fisherfaces* improved on these techniques using Fisher Discriminant Analysis (FDA) [2]. Today, most modern algorithms use convolutional neural networks (CNNs). Examples of these systems include Google's FaceNet [8] and Facebook's DeepFace [12].

In a typical facial recognition problem, a model is presented with an image of an unidentified face (a *probe*) and a database of identified faces (a *gallery*). A facial recognition algorithm is tasked with finding the most likely identity of the probe using the gallery. If the gallery contains n photos, this task can be decomposed into n one-to-one comparisons between the probe and individual photos in the gallery. For each of these n pairs of images, an algorithm might be asked to quantify their similarity or outright decide whether they contain the same person.

Our work. We sought to explore a wide range of facial recognition techniques. We implemented Eigenfaces and Fisherfaces as they were specified in the original papers, attempted to extend these techniques, and developed a series of neural networks based on the ideas in FaceNet.

[1] Many of these applications raise substantial privacy concerns [4]. We do not comment on the policy implications of facial recognition in this report, but we acknowledge the gravity of these risks and the attention they deserve.

Evaluation. There are a variety of face recognition datasets available. We chose to use the Labeled Faces in the Wild (LFW) dataset [7], a popular benchmark that is small enough to run on the constrained computational resources we had available for this project. It contains 13,233 images of 5,749 people, centered and cropped to 250 $\times$ 250 pixels. The images were collected from Yahoo News with the Viola-Jones face detector [14]. We chose to use a version of the dataset that is also aligned [6], allowing us to focus purely on the task of facial recognition. The dataset's testing regime consists of performing ten-fold cross validation on 6,000 preselected pairs sorted into ten 600-photo folds. Each fold contains 300 matching pairs and 300 non-matching pairs.

In this paper, we considered two LFW variants: unrestricted (in which the identities of the photos are provided along with the training pairs) and restricted (in which the identities are not provided). The unrestricted variant facilitates algorithms that do class-by-class analysis. It also makes it possible to generate new pairs of images out of the provided training pairs. LFW includes separate categories permitting algorithms to use outside training data (FaceNet is trained on more than 200 million images, for example), but we did not exercise this option. We often further cropped and shrank the LFW images to make facial recognition feasible.

Deliverables. In this report, we produced the following artifacts and experiments:

- A new test harness for importing the LFW dataset.[2]
- A generalized implementation of PCA and an instantiation of this technique to replicate the Eigenfaces system.
- A generalized implementation of FDA and a corresponding instantiation of the Fisherfaces system.
- A hybrid of PCA and FDA combining their advantages.
- A discussion of techniques for overcoming the limitations of Fisherfaces that failed to work on our data.
- A CNN that embeds images in Euclidean space, trained on pairs of matching or non-matching images.

[2] The developer of the sklearn harness did not fully understand the LFW training regime. One comment in the code reads, *As I am not sure as to implement the "Unrestricted" variant correctly, I left it as unsupported for now [9].* If possible, we plan to eventually contribute our harness to sklearn in lieu of the flawed library.

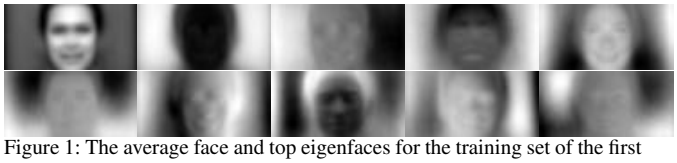

Figure 1: The average face and top eigenfaces for the training set of the first fold of LFW (size 36×75). The average face is in the upper left; the top eigenfaces are in order from left to right along the rest of the top row and the bottom row.

- A CNN that embeds images in Euclidean space, trained on triples of a matching pair and a non-matching image.
- A CNN that directly classifies whether two images are the same or different.
- A discussion of neural net approaches that failed.

All of our code is available at
`https://github.com/jfrankle/ml-project`

2. Matrix Methods

Eigenfaces. The authors of the Eigenfaces paper observe that the dimensionality of face images is often too high to use in its raw form. Instead, the authors apply PCA to face images, reducing their dimensionality to a space where they can be handled more easily. The Eigenface algorithm assumes that the images are aligned and normalized. It first extracts the average face $\bar{F}$ from the training faces $F_1, ..., F_n$. The pixel at location x, y of $\bar{F}$ is the average of the pixel values at location x, y in each of the training faces. The average face for the training data of one fold of LFW is in Figure 1.

The training faces are then centered: $G_i = F_i - \bar{F}$. These values are used to compute a covariance matrix (also referred to as a *scatter matrix*): $S = \frac{1}{n}\sum_{i=1}^{n} G_i G_i^T$. The scatter matrix is the sum of the covariances of each face. As the Eigenfaces paper observes, this computation can be expressed as $S = A^T A$, where $A = [G_1, ..., G_n]^T$. The principal components of the dataset are the eigenvectors u_i of S. These eigenvectors form a new basis into which we project the original data. Each eigenvector has the same number of pixels as a face, hence the name "eigenfaces;" example eigenfaces are in Figure 1.[3]

Since we aim to reduce the dimensionality of the data, we often choose to use only the eigenvectors with the top k eigenvalues. Given a matrix $C = [c_1, ..., c_k]^T$ containing the principal components, the projection of face F_i into this new space is $G_i^T C$. We also considered *whitening* the eigenvectors (dividing an eigenvector u_i by $\sqrt{\lambda_i}$), which ensures the data has the same variance in each dimension.

Once projected into this lower-dimensional space, we can compute the distance between two faces to determine whether they are the same person. The Eigenfaces authors do not elaborate on how they selected the distance threshold that determines whether two faces are the same or different; we use logistic regression on the distance values for our training set to make this determination.

[3] The authors of the paper actually take a roundabout way of calculating eigenfaces that depends on the assumption that "in practice, the training set of face images will be relatively small" compared to the number of pixel features. In our context, those numbers are not all that different, so we calculated the eigenvectors of S directly.

components	10	25	50	75	100
scaling with color	20%	20%	10%	10%	10%
scaling without color	50%	30%	30%	20%	20%

Figure 2: The highest scaling we could achieve using PCA with the specified number of components. We timed out our tests after 180 seconds.

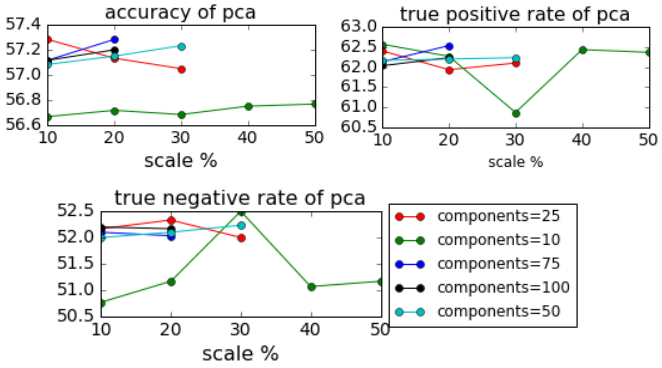

Figure 3: The accuracy, true positive rate, and true negative rate of the PCA-based algorithms on black and white images.

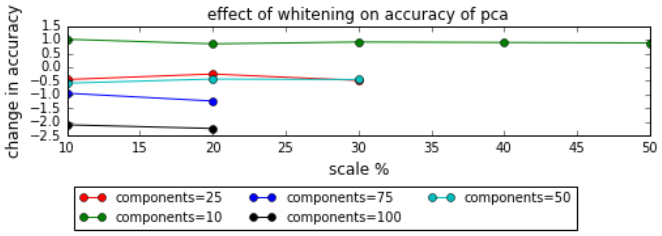

Figure 4: The difference between accuracy with whitening and accuracy without on black and white images.

The Eigenfaces algorithm does not take advantage of the fact that we know whether pairs of faces match, let alone that (in the unrestricted LFW variant) we know the identities of those faces. We will exploit this information in Fisherfaces.

To compute the top k eigenvectors and eigenvalues, we use an iterative algorithm called *Power Iteration*. We select a random unit vector x_0 and repeatedly recompute $x_{t+1} = \frac{Sx_t}{||Sx_t||}$ until $||x_{t+1} - x_t||$ is smaller than some threshold. At this point, $u = Sx_t$ is the eigenvector and $\lambda = ||Sx_t||$ is the eigenvalue. To calculate the next eigenvector, we repeat the same computation with $S' = S - \lambda u u^T$. In our experiments, we used a threshold of 10^{-6}. Our description of PCA and implementation are based on the tutorial in [3].

We evaluated the performance of PCA along a number of dimensions, changing the number of components (10, 25, 50, 75, 100), the resolution of the image (scaled to 10%, 20%, 30%, 40%, 50%), whether the image was in color, and whether the eigenvectors were whitened. Figure 2 shows the highest scaling on which our implementation could complete LFW within 180 seconds. As expected, color images, which have three times as many features, also slowed down the algorithm dramatically. As we increased the number of components, the algorithm struggled to process higher-resolution

images. The benefits of these additional components, however, are limited. As Figure 3 shows, after about 25 or 50 components, the accuracy rate did not improve. We also found that those test runs that were able to complete on color images showed negligible improvements over black and white images. As demonstrated in Figure 4, whitening was also of no benefit.

In general, the best accuracy of the PCA-based algorithms was around 57%. Interestingly, Figure 3 shows that the algorithm that performed nearly 10% better on matches than non-matches, indicating it was likely biased toward deciding that images contained the same face. This performance is far lower than that achieved in the original Eigenfaces paper, which ranged between 85% and 100%. With that said, the Eigenfaces paper used 2,500 images of 16 people in controlled settings, while the LFW dataset contains more than 13,000 uncontrolled images of nearly 6,000. The LFW leaderboard shows that Eigenfaces achieves approximately 60% accuracy, matching our results.

Fisherfaces. Several years after Eigenfaces was published, Belhumeur et al. [2] offered a substantial critique of the work. They quote from [1]: *the variations between images of the same face due to illumination and viewing direction are almost always larger than the image variations due to change in face identity.* They conclude that, since PCA "maximize[s] the total scatter across...all images of all faces" (i.e., since it ignores the identities of the faces), the low-dimensional subspace that it generates will accentuate variations in lighting, angle, and pose rather than variations in identity. In short, unless all photos were taken in identical circumstances, PCA cannot actually help to separate the images by identity, meaning it will not make recognition much easier. The LFW dataset comprises images taken "in the wild," so this criticism could certainly apply here.

Belhumeur et al. propose Fisher Discriminant Analysis (FDA) in lieu of PCA. FDA takes into account the class labels of the images, calculating both the within-class scatter and between-class scatter. Within-class scatter is the scatter matrix where each face is centered with respect to its class mean. Formally, $S_w = \sum_{i=1}^{c} \sum_{F_j \in X_i} (F_j - \bar{F}_i)(F_j - \bar{F}_i)^T$ where c is the number of classes, X_i is the set of faces in class i, and $\bar{F}_i$ is the mean face of class i. Between-class scatter is the scatter matrix where the mean of each class is centered with respect to the overall mean. Formally, $S_b = \sum_{i=1}^{c} n_i(\bar{F}_i - \bar{F})(\bar{F}_i - \bar{F})^T$, where n_i is the number of samples in class i. Fisher's linear discriminant aims to find the value of U that maximizes $\frac{||U^T S_b U||}{||U^T S_w U||}$. That is, we desire a value of U that maximizes between-class scatter as compared to within-class scatter.

The matrix U serves as the lower-dimensional space into which FDA projects faces. Solving for U, $S_w^{-1} S_b u_i = \lambda u_i$, meaning that this lower-dimensional basis comprises the eigenvectors of $S_w^{-1} S_b$. As many researchers have recognized (including Belhumeur et al.), a singular S_w can spell

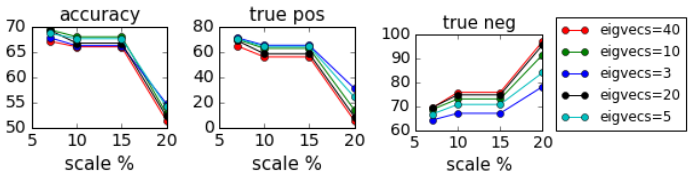

Figure 5: FDA accuracy in black and white without mirroring.

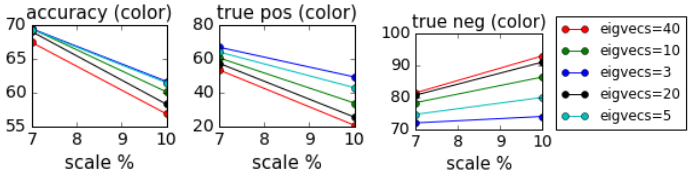

Figure 6: FDA accuracy in color without mirroring.

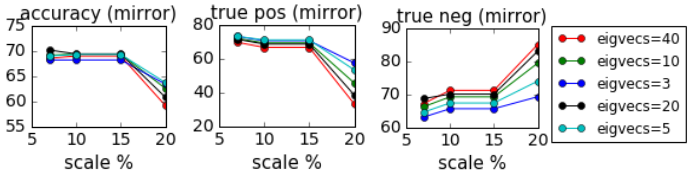

Figure 7: FDA accuracy in black and white with mirroring.

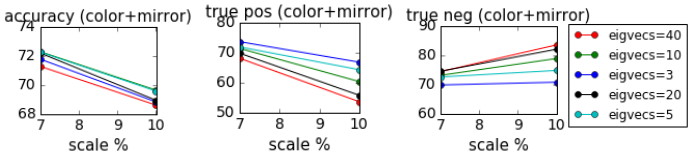

Figure 8: FDA accuracy in color with mirroring. The best results we achieved with Fisher LDA.

doom for FDA. As Zhang et al. explain, this occurs whenever "there are a small number of samples but a very large number of variables." [15] Zhang et al. go on to list several applications where this is often the case, naming facial recognition as an instance where pixels often far outnumber samples. Belhumeur, Zhang, and many others offer a variety of tactics for overcoming this limitation; we will discuss this design space in the next section. Here, we explore the performance of FDA when we limit the number of features to keep S_w invertible by scaling faces down to dozens of pixels.

We evaluated the performance of FDA with 3, 5, 10, 20, and 40 eigenvectors at 7%, 10%, 15%, and 20% scaling. Given the small size of the images we used, time was not a limiting factor. Instead, the limitation was features. The highest scaling we could achieve before $det(S_w)$ reached 0 was 15% with black and white images and 7% with color. We also tried doubling the number of samples by mirroring each image along the y-axis. This transformation is not strictly sound—faces are not symmetric—but at exceedingly low resolution, this fact is unlikely to seriously impact our accuracy. With mirroring, we were able to reach 20% scaling with black and white and 10% scaling with color.

As Figures 5, 6, 7, and 8 show, FDA validated the hypothesis of Belhumeur et al., performing dramatically better than PCA. Figure 5 contains the performance of FDA on grayscale images without mirroring for various scalings and numbers of eigenvectors. The number of eigenvectors had

little effect on the performance; after three to five, performance stopped improving and even began to decline. FDA seemed to perform best at the lowest possible scaling, 7% (8×17 pixels). At 20% scaling, accuracy dropped off as $det(S_w)$ approached (but did not reach) 0. True positive and true negative rates generally seemed evenly balanced. The same patterns can be found in Figure 7, which was tested on black and white images with mirroring. The mirroring ensured that the drop-off at 20% scaling was less severe.

The best performance we achieved was with color (Figure 6) and color+mirroring (Figure 8) at 7% scaling. With color, mirroring, and 7% scaling, we achieved about 72% accuracy on LFW no matter how many eigenvectors were used, an accuracy rate nearly 15% higher than our best result using PCA. This accuracy is high enough to (barely) beat the bottom algorithm on the LFW leaderboard for the unrestricted variant. We find it particularly surprising and notable that this accuracy was achieved on an image with only 136 pixels (albeit with three colors per pixel).

The best explanation for this result is that conjectured by Belhumeur—FDA prioritizes dimensionality reduction that tightly clusters faces within classes and separates these clusters from one another. We posit that lower resolution proved better for FDA because all of the higher resolutions we tried began to approach singular within-class scatter matrices, culminating in the drop-off at 20% scaling. If we had more images, we believe performance would improve at higher resolutions. Our mild success scaling up FDA through mirroring supports this hypothesis. Another possibility is that lower resolutions forced the algorithm to generalize, eliminating noise present at higher resolutions that might lead to overfitting or confusion on the part of the algorithm.

Overcoming singular S_w. Given success in spite of the limitations of FDA, we mined the research literature for techniques to overcome a singular S_w. Several sources suggested simply using the Moore-Penrose inverse of S_w. Doing so yielded an inverse matrix, but the performance remained no better than before, degrading to random on larger feature spaces. We also implemented formulations described by Howland et al. [5] and Zhang et al. [15]. Neither of these approaches yielded meaningful changes in performance.

Instead, we turned to a technique suggested by Belhumeur in the Fisherfaces paper: use PCA to reduce the dimensionality of the feature space to a size that FDA can handle and then run FDA. Belhumeur suggests reducing the dimensionality as little as possible—to $n - c$ dimensions (where n is the number of faces and c is the number of classes). In our case, this would involve computing hundreds of eigenvectors. Earlier, we found that, after a few dozen eigenvectors, the performance of PCA was saturated. We followed this guidance, using fewer principal components and accepting that some small amount of information might be lost in exchange for a feasible algorithm.

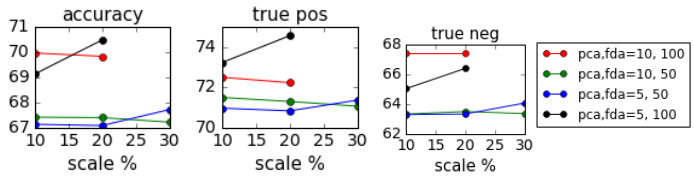

Figure 9: Accuracy of PCA followed by FDA in black and white without mirroring.

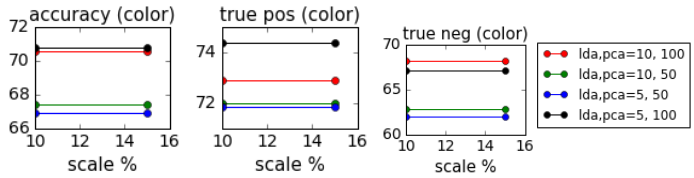

Figure 10: Accuracy of PCA followed by FDA in color without mirroring.

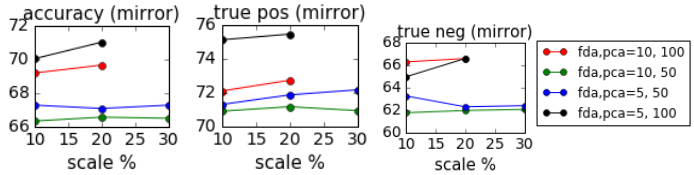

Figure 11: Accuracy of PCA followed by FDA in black and white with mirroring.

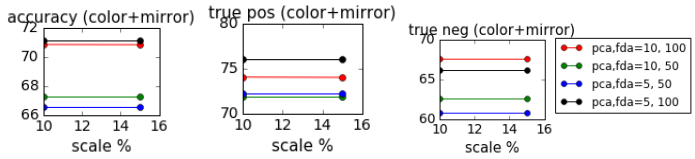

Figure 12: Accuracy of PCA followed by FDA in color with mirroring.

The results of this hybrid approach are visible in Figures 9, 10, 11, and 12. We tried multiple combinations of resolution, color, mirroring, and the number of principal components and Fisher eigenvectors. We increased the timeout threshold to 360 seconds to accommodate the fact that two algorithms were run in sequence. In general, we were able to approach but not improve upon our results from FDA alone. All four figures, which portray all combinations of color and mirroring, show that increasing the number of principal components improved the performance of the algorithm; the number of Fisher eigenvectors was largely irrelevant. Intuitively, this result indicates that the overall performance of the system depended on the amount of information that PCA was able to preserve in the lower-dimensional space; more components would allow more information to get through.

Increasing the resolution of the images had no clear effect on the data (except that, at higher resolution, many tests timed out). Notably, increasing the resolution of color had no impact on the performance of the system. There was also a substantial divide between true positive and true negative performance, indicating that the algorithm may have been biased toward guessing that the pair of images was a match.

Overall we conclude that, although this technique was able to stretch the feature-space size restrictions that constrain FDA, it did not lead to improved performance.

3. Neural Networks

Overview. Having surveyed matrix methods that dominated early facial recognition algorithms, we turn now to state-of-the-art neural networks. In reviewing the literature, we found a wide variety of training regimes in use. As such, we aimed to focus less on the particularities of the convolutional architecture (strides, filter sizes, etc. that were investigated extensively in Homework 3). Instead, we turned our attention to the way in which a network represents a face and the procedures used to train the network accordingly.

Consider some examples. Facebook's Deepface is trained as a classifier; its penultimate hidden layer serves as the feature vector for each new face [12]. We attempted to develop a network that mimicked Facebook's approach, but LFW has so few images per class that many classes might see only one or two samples during training. In addition, the testing data consisted mainly of identities that were not present in the training data, dooming even a pure classification network that tried to explicitly identify each face. Although we prototyped both of these approaches, neither merited evaluation.

Google's Facenet explicitly embeds faces in 128 dimensional Euclidean space. Each of these embeddings is a unit vector. Each training example consists of a triple: two images that match and one that is different; its loss function aims to pull the matching pair closer together while pushing away the non-match. We developed and evaluated two networks in this style, one of which is trained on pairs of images (some matching, some not) and one of which is trained in triples just as in Facenet. The advantage of this approach is that embeddings in space can be used for a wide variety of tasks (clustering, similarity, etc.) that go beyond LFW-style facial verification. To classify images, we used logistic regression on the resulting training set distances.

We also designed and evaluated a network that aims to simply decide whether two images contain the same person. Its output unit is a single sigmoid indicating the probability that the two input images are a match.

Architecture. In every test, our network has the same underlying structure. An input layer containing the image is fed to a 3×3 convolutional layer with stride 1 that produces 32 output layers. These layers are then fed to a 3×3 max pooling layer with stride 2. A second 3×3 convolutional layer with stride 1 produces 64 output layers, which are fed to another 3×3 max pooling layer with stride 1. Finally, a 75-unit fully-connected hidden layer leads to the appropriate output layer for each experiment. When we trained on multiple images at once, all image inputs shared the same weights. All hidden units are RELUs.

Evaluation. We used the unrestricted variant of LFW, meaning we had access to the identities of every image in our training set. We mixed and matched faces from the initial 5,400 pairs of faces in the LFW training set to randomly generate a stream of potentially millions of match and non-match pairs. In practice, we typically trained our network on tens of thousands of examples. In those architectures that embedded images, we generated a further 5,000 random pairs to feed to logistic regression to decide the distance threshold at which we should consider faces to match. Due to the resource-intensiveness of training and testing neural networks, we were limited in the number of trials we could run. Each full LFW run took multiple hours (or days). We therefore used the first fold (of ten) of the LFW dataset as our validation set; we only did a full LFW run after selecting the best-performing parameters. Since neural networks are randomly initialized, we averaged the results of three runs.

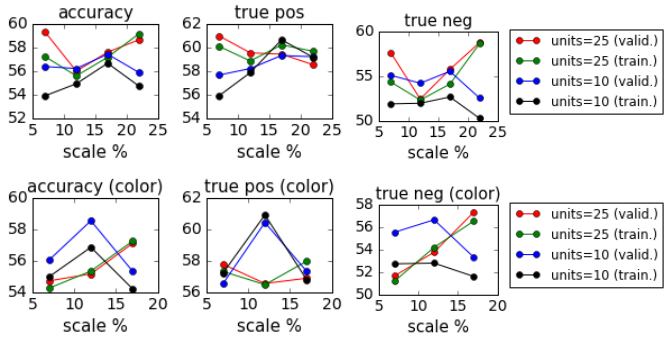

Figure 13: The effect of scaling, color, and number of output units on the performance of the pair neural network on the validation set. Averaged over three runs for each data point (20,000 examples per run).

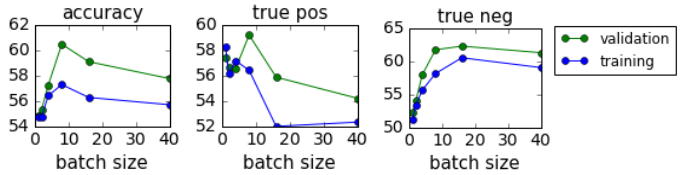

Figure 14: The effect of batching on accuracy on the pair neural network with images at 12% scaling in color. Averaged over three runs for each data point. Each network saw 20,000 examples in different sized batches.

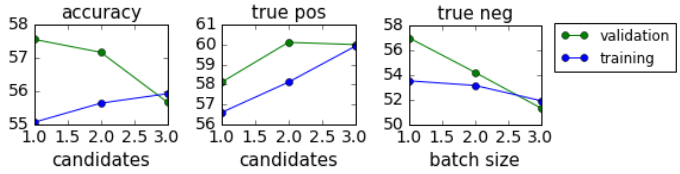

Figure 15: The effect of adversarial examples on accuracy on the pair neural network with images at 12% scaling in color. Averaged over three runs for each data point. Each network saw 40,000 examples.

Pair embedding. Our first experiment involved employing Facenet's approach of embedding faces in Euclidean space. Facenet used hundreds of millions of photos and thousands of CPU-hours to train a network with tens of layers and a 128-unit output layer. We attempted to scale down this approach, training a network with a few layers on tens of thousands of photos with a correspondingly smaller output.

Before attempting to implement the triplet loss function from the paper, we tried a simpler approach: on each training iteration, the network is (with equal probability) fed either a pair of matching or non-matching faces. If the faces match, we minimize the L2 norm of the difference between the

faces' embeddings; if they do not match, we maximize it. Facenet cites [10], which employs a similar approach.

Initially, we explored the optimal combination of embedding size, input image resolution, and input image color. We tested embeddings of 10 and 25 nodes with scalings of 7%, 12%, 17%, and 22% with and without color (22% timed out on color). We allowed the network to run for 20,000 training iterations. These results appear in Figure 13. The number of output units seemed to have little impact on performance; we preferred 10 output units, which trained faster. The black and white image seemed to perform best at the extremes (7% and 22% scaling), while the network performed best overall on images scaled to 12% in color. This data was quite noisy, and the trends may have been a casualty of the random initialization of neural networks, even with averaging over three runs.

These initial runs were all conducted by running one example through the network at a time. In contrast, most papers (including Facenet) suggest batching many training examples together to improve performance and produce gradients that better represent the full data. Figure 14 shows the effect of batching on accuracy and running time using images with 12% scaling in color and a network with 10 output units. Each run saw 20,000 images in batches of increasing sizes. Accuracy increased with batch size up to batches of about 8, after which it began to drop off. Bigger batches may have had too little opportunity to influence the network's weights, since they saw correspondingly fewer training iterations. They may also have too heavily diluted the effect of individual training examples. Interestingly, nearly all of the change in batch accuracy came from true positives—as batch sizes reached 16, the true positive rate plummeted. It is possible that the network was still learning to distinguish faces when training stopped and that increasing the number of iterations may have improved performance. Conveniently, batching also improves the running time of the algorithm, since each Tensorflow training iteration has significant overhead. The best-performing batch size (8) finished almost four times faster than when no batches were used.

The authors of Facenet suggest one more optimization: choosing hard examples. They note that many training examples will already be easy to classify (since images are already close together or far apart), wasting computation on useless gradient calculations. Instead, they adversarially sought out examples that the network was misclassifying, making each gradient computation count. We adapted that approach here: at each step, we selected an anchor face and randomly selected k candidate faces with which to pair it. When we wanted to train on a match, we selected the candidate that was furthest from the anchor according to our network; on a non-match, we selected the candidate that was closest. The results of applying this approach for $k = 1$, 2, and 3 after 40,000 iterations is in Figure 15. Although the data was not entirely conclusive (given the small number of runs and the computational expense of conducting each

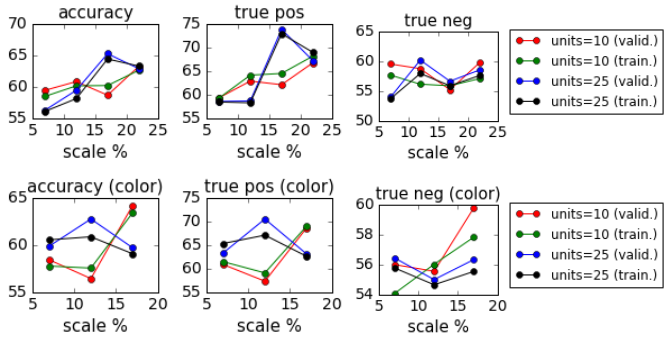

Figure 16: The effect of scaling, color, and number of output units on the performance of the triple neural network on the validation set. Averaged over three runs for each data point (20,000 examples per run).

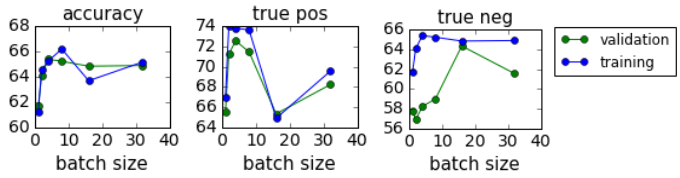

Figure 17: The effect of batching on accuracy on the triple neural network with images at 17% scaling without color. Averaged over three runs for each data point. Each network saw 40,000 examples in different sized batches.

experiment), adding candidates surprisingly harmed performance. It is possible that, by choosing more difficult examples, we slowed the network's rate of convergence. We will return to this topic in the next section, where training for longer did, indeed, benefit from longer candidate lists.

Finally, we ran the best-performing configuration (12% scaling, 10 outputs, color) with the best-performing batch size (8) and number of candidates (1) on the full LFW benchmark. We achieved 55.6% accuracy, worse than Eigenfaces and narrowly better than random. Overall, this performance is within the realm of Eigenfaces, a far cry from the promised performance of Facenet and a far more computationally intensive way of achieving a relatively poor result.

Triple embedding. In this section, we repeat the same experiments as before with a different training procedure and loss function. The Facenet paper suggests training on triples—an anchor image F^a, a positive example F^p that matches, and a negative example F^n that does not match. Their loss function is below given N images in a batch: $\sum_{i=1}^{N} \left(||f(F_i^a) - f(F_i^p)||_2^2 - ||f(F_i^a) - f(F_i^n)||_2^2 + \alpha \right)$. In other words, it aims to minimize the difference between the proximity of F^p to the anchor and the proximity of F^n to the anchor plus a margin α that it tries to enforce. We follow the paper's suggestion to set α to 0.2. The network in this section was otherwise identical to the previous one.

Our performance on the validation set (the first fold of LFW) across scales, color, and number of output units is in Figure 16. Switching from the pairwise training to the triple training seemed to improve performance by a few percent across the board. At best, the network reached about 65% accuracy for images scaled to 17% in black and white with 25 output units. Just as in the pair network, the resulting

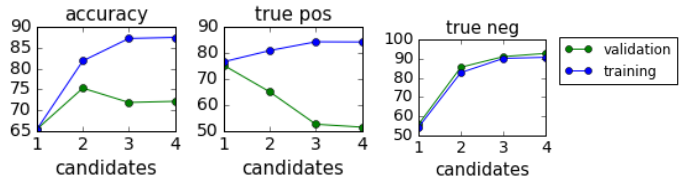

Figure 18: The effect of adversarial examples on accuracy on the triple neural network with images at 17% scaling without color. Averaged over three runs for each data point. Each network saw 60,000 examples.

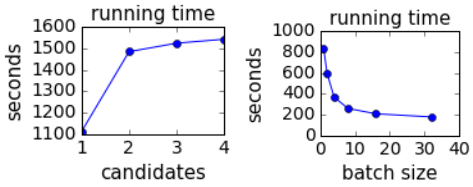

Figure 19: The running time of the experiments in Figures 17 and 18.

classifiers have about 10% higher accuracy on matches than on non-matches. Performance on training data and validation data were almost identical.

As in the previous section, we also experimented with the benefits of batching (Figure 17) on the best-performing configuration from Figure 16. Our results mirrored those from the pair network—assuming the network always saw 40,000 pairs of faces, it performed best when trained on batches of abour 4 to 8. With batches that were any larger, performance dropped off. As Figure 19 shows, batching is also good for performance. There is a "sweet spot" for batching in which accuracy improves by a few percent while training runs four times faster.

We also repeated the experiment of adversarially selecting difficult training examples for the network. This time, we trained the network longer—60,000 examples without batches. Given an anchor face, the network chose the most difficult positive and negative faces each from lists of k candidates. Figure 18 shows the results. Unlike the pair network, the triple network excelled when examples were chosen adversarially. Performance improved nearly 10% when choosing from among two candidates rather than one. Interestingly, much of this increase came from a dramatically improved true negative rate, suggesting that the difficult examples helped to undo the imbalance between true negative and true positive rates. The graphs do indicate, however, that the network may have overfit after 60,000 iterations—training accuracy far outstrips validation accuracy. Unfortunately, this improved performance came at a cost. Figure 19 shows that switching from one candidate to multiple candidates increased the training running time by nearly 50%. The overhead of using candidates was nearly constant, however—after using two candidates, the cost of each additional candidate was small.

We can gain some insight into the mechanism behind this behavior by examining Figure 20, which shows the distances between matching (above) and non-matching (below) pairs of faces at each iteration. The graph is taken from Tensorboard (a visualization tool for Tensorflow) and is exponentially smoothed to make the trend easier to see. The orange, teal, purple, and blue lines represent trials with candidates = $1, 2, 3,$ and 4. As the number of candidates increased, the network took longer to overcome initial poor performance and converge to a reasonable result (the lines rising from zero to their eventual stable levels). We hypothesize that, given more training time, networks with longer candidate lists would also have converged (and perhaps performed better). Facenet, which was trained over thousands of CPU hours, could have taken full advantage of this behavior.

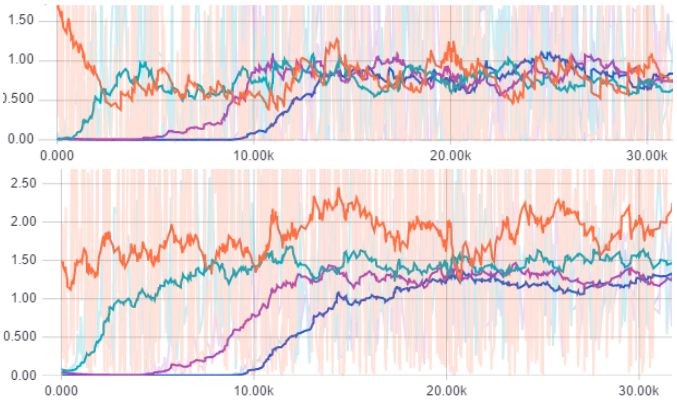

Figure 20: Given a triple of training images F^a, F^p, and F^n, the top (bottom) graph shows the squared distance between F^a and F^p (F^a and F^n) with respect to the number of training iterations that have occurred so far. Each line shows a different trial run on grayscale images scaled to 17% with one triple per batch. The orange, teal, purple, and blue lines represent trials with candidates = $1, 2, 3,$ and 4. The lines are exponentially smoothed to clarify the trends. Graphs were taken from Tensorboard. The top chart has lines between .5 and 1, while the bottom chart has lines above 1, illustrating how the network learns to distinguish matching and nonmatching pairs over time.

Choosing the optimal parameters as determined above, we ran the full LFW benchmark on images scaled to 17% in black and white with 25 output units. The network was trained for 20,000 iterations in batches of 4 (80,000 triples in total) with candidate lists of length 2. Averaged over three runs, the network achieved 71.7% accuracy with a true positive rate of 58.7% and a true negative rate of 84.7%, roughly matching the performance of the best FDA run. Where the FDA algorithm ran in minutes, however, three full runs of the LFW benchmark on this neural network configuration took about 35 CPU-hours. [4]

Classifier. Our final neural network involves a classifier network that attempts to classify whether a pair of faces belong to the same person. The architecture consists of an input layer which takes in a matrix of two images, concatenated along their width. Similar to the neural networks described in the previous section, there are two convolutional hidden layers with RELU activation (each with a corresponding pooling later) and a fully connected hidden layer (also with RELU units). The output layer consists of a single sigmoid

[4] This extraordinary computational cost prevented us from doing full LFW runs on each neural network that we tested and generally constrained our ability to fully explore the design space of neural networks.

unit which gives the probability that the pair of input images are from the same person. (During the testing phase, we classify image pairs as the same or different by rounding the value of this output unit.) When designing this network, we tried several different architectures, all of which involved keeping the data from the two faces separate before concatenating them in one of the hidden layers of the network. However, these other architectures failed to converge so we used the simpler architecture in which the two images are concatenated before being fed into the input layer.

We trained and tested using a single fold of the LFW images, for three trials each, using different numbers of iterations, scalings, and batch sizes both with and without color. We averaged the three trials to better see the trends. Unlike the other networks, we did not test using a batch size of 1. This was because, during our preliminary trials, the weights of the network either exploded or decreased to 0, classifying images as either all positive or all negative.

Figure 21 shows the accuracy, true positive rate, and true negative rate for the training and test sets. The cases in which the training and validation metrics are equal are those in which the neural network failed to train (the weights exploded, causing all data points to be classified identically).

The number of iterations had little effect on the validation accuracy of the classifier. Comparing the accuracy graphs (for both batch size and color possibilities), we do not see that the trendlines for validation accuracy are significantly different for $10,000$ and $20,000$ iterations. On the other hand, accuracy on the training set is generally higher for $20,000$ iterations, which is to be expected.

The true positive rate for the validation set was generally higher for $20,000$ iterations when the images were larger, while being similar for both iteration possibilities when images were smaller; in the case of the color data, $10,000$ iterations often outperformed $20,000$ iterations. We would expect that datasets with smaller images take fewer iterations of training to reach the optimal weights. Running for $20,000$ iterations on smaller images appeared to overtrain the network, decreasing performance on the validation set. Interestingly, the true negative rate often showed the opposite trend. We theorize that, in the beginning of training, our network tends to over-identify matching pairs of faces, whereas later in training, the graph becomes better at recognizing non-matching faces, at the expense of more misses.

Figure 22 shows the time required to train the network. We only show the graphs for the non-color data, as the graphs for the color data are similar. As we would expect, training networks on larger images takes longer, as does training for more iterations. Increasing the batch size while maintaining the same number of iterations did not tangibly affect the training time.

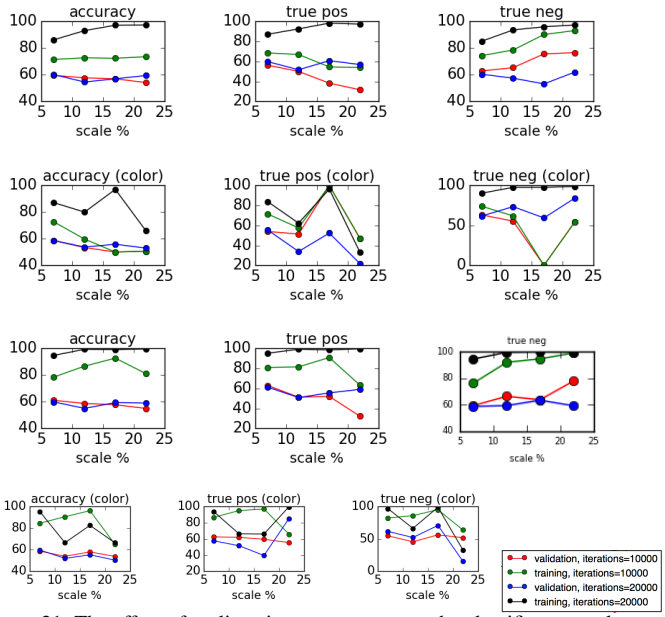

Figure 21: The effect of scaling size on accuracy on the classifier network with images at scaling values of 0.07, 0.12, 0.17, 0.22 with and without color. The top two rows use a batch size of 5 and the bottom two rows use a batch size of 10. Averaged over three runs.

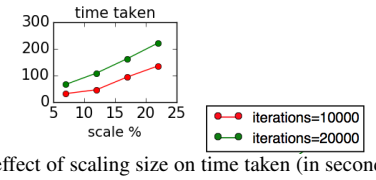

Figure 22: The effect of scaling size on time taken (in seconds) to train the classifier network with images at scaling values of 0.07, 0.12, 0.17, 0.22 without color. The left graph uses a batch size of 5 and the right graph uses a batch size of 10. Averaged over three runs.

4. Discussion.

Explaining neural net performance. We conclude with a question: why did neural networks perform so poorly in our experiments when they dominate vision benchmarks and consistently achieve perfect scores on LFW? It is possible that there was a fundamental flaw with our default network architecture that doomed performance. However, we used a standard convolutional neural network, so we doubt this could entirely explain the poor performance. Instead, we turn to two other limitations: data and computation.

Our neural networks were trained using only the LFW training sets, each of which contained just over 10,000 images. In contrast, Google trained Facenet on hundreds of millions of images taken from its vast corpus of user photos. Our neural networks were trained for one or two CPU hours at most, while Facenet was trained for 1,000-2,000 CPU hours. Facenet also had an order of magnitude more layers than our network. Neural networks have a reputation for needing a massive amount of training data. This is especially true in the context of facial recognition—faces have immense variation in pose, expression, and illumination within classes and skin tone and facial structure between classes. Our training data may simply have been too limited to extract better perfor-

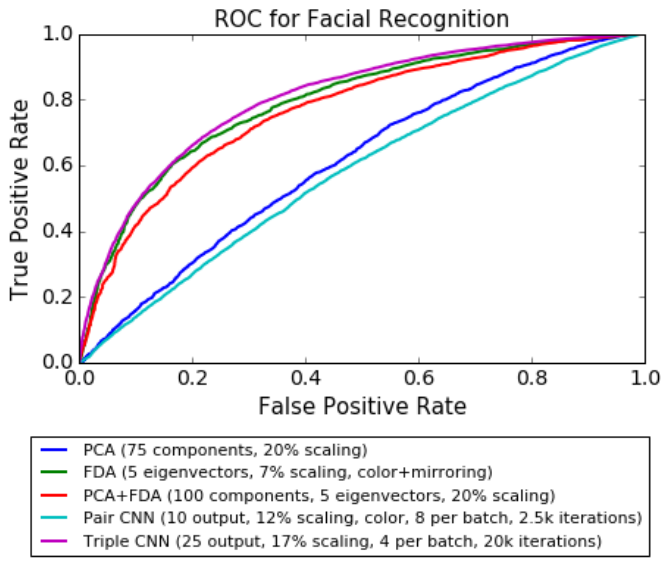

Figure 23: ROCs for the best parameters in each model category.

mance from a neural network—our networks likely saw too few iterations and too many instances of the same faces. We could have turned to larger datasets (CASIA, MegaFace) for more images, but even then, our tiny computational budget would have crippled our ability to exploit the images.

Poor overall numbers notwithstanding, we are pleased to have observed several interesting trends that likely have implications for much bigger networks. The neural networks seemed to prefer smaller color images to larger black and white images. Batching improved performance to a point, but always improved running time. Using adversarial training with candidates improved the performance of the triplet embedding but also slowed convergence; it seemed to harm the performance in the pair embedding. The triplet loss function led to a network that performed markedly better than that trained on the pairwise loss function. Although we were unable to exceed 99% accuracy as Facenet did, our results still provide a number of useful lessons for designing neural networks for facial recognition.

In the context of our experimental configuration and limitations, however, the matrix methods (particularly Fisherfaces) outperformed the neural networks (often dramatically) and did so with much faster training.

Receiver operating characteristic. We close by presenting the receiver operating characteristic (ROC) curves for the best networks in each of our experiments in Figure 23. ROCs show the tradeoff between true positives and false positives as we adjust the distance threshold for deciding whether two faces are a match. Ideally, the false positive rate should be near 0 and the true positive rate should be near 1, corresponding to the upper left corner of the ROC. The FDA and FDA+PCA graphs are the closest to optimal, bowing inwards. The neural network and PCA graphs are further from ideal. For example, the PCA graph shows that the network can only achieve false positive rate of 10% if its true positive rate is about 15%; the corresponding true positive rate for FDA is about 50%.

Future work/aspirational goals. There are several extensions to this project that we wanted to pursue but did not have the time to complete.

- Would the performance of FDA have improved with a bigger dataset? How would the performance have changed on the CASIA or MegaFace datasets, which have hundreds of thousands of images but may have made computing eigenvectors harder?
- What was the effect of specific neural network architecture choices on our performance? Would changing the number of layers or the sizes/strides of the convolutional and pooling layers have made a substantial difference in performance?
- Would performance have benefited from other popular techniques in computer vision neural networks? Could we have used boosting as in [10], or inception modules as in [11]? We were curious about implementing a neural network with an inception architecture (like Facenet did), but did not have time to do so.
- Could we have obviated the need to use pre-aligned images by implementing (or running) the Viola-Jones face detector and building our own neural net for aligning?

5. Division of Labor

Jonathan Frankle. Reimplemented a test harness for downloading LFW images, resizing them, and performing the LFW evaluation according to the official procedure. Explored and aggregated related work. Implemented PCA. Implemented Fisher dimensionality reduction. Implemented the joint PCA/Fisher model. Implemented the pairwise neural network. Implemented the triple neural network. Evaluated the aforementioned systems. Wrote all sections of the paper except the subsection about the neural classifier.

Caroline Morganti. Implemented and evaluated the classifier neural network. Wrote the corresponding section of the paper. Explored related work. Edited all sections of the paper.

References

[1] Yael Adini, Yael Moses, and Shimon Ullman. “Face Recognition: The Problem of Compensating for Changes in Illumination Direction”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 19.7 (July 1997), pp. 721–732. ISSN: 0162-8828. DOI: 10.1109/34.598229. URL: http://dx.doi.org/10.1109/34.598229.

[2] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997), pp. 711–720.

[3] Greg Coombe. “An introduction to principal component analysis and online singular value decomposition”. In: *graduate paper, University of North Carolina, Chapel Hill, NC* (2006).

[4] Clare Garvie, Alvaro Bedoya, and Jonathan Frankle. *The Perpetual Lineup: Unregulated Police Face Recognition in America*. Tech. rep. URL: www.perpetuallineup.org.

[5] Peg Howland, Moongu Jeon, and Haesun Park. “Structure preserving dimension reduction for clustered text data based on the generalized singular value decomposition”. In: *SIAM Journal on Matrix Analysis and Applications* 25.1 (2003), pp. 165–179.

[6] Gary B. Huang, Vidit Jain, and Erik Learned-Miller. “Unsupervised Joint Alignment of Complex Images”. In: *ICCV*. 2007.

[7] Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 07-49. University of Massachusetts, Amherst, Oct. 2007.

[8] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823.

[9] *sklearn.datasets.lfw*. URL: https://github.com/scikit-learn/scikit-learn/blob/b7e3702/sklearn/datasets/lfw.py#L415 (visited on 12/09/2016).

[10] Yi Sun et al. “Deep learning face representation by joint identification-verification”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 1988–1996.

[11] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[12] Yaniv Taigman et al. “Deepface: Closing the gap to human-level performance in face verification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708.

[13] Matthew Turk and Alex Pentland. “Eigenfaces for recognition”. In: *Journal of cognitive neuroscience* 3.1 (1991), pp. 71–86.

[14] Paul Viola and Michael Jones. “Rapid object detection using a boosted cascade of simple features”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

[15] Zhihua Zhang, Guang Dai, and Michael I. Jordan. “A Flexible and Efficient Algorithm for Regularized Fisher Discriminant Analysis”. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*. ECML PKDD ’09. Bled, Slovenia: Springer-Verlag, 2009, pp. 632–647. ISBN: 978-3-642-04173-0. DOI: 10.1007/978-3-642-04174-7_41. URL: http://dx.doi.org/10.1007/978-3-642-04174-7_41.