Supplemental Material.
Visual Psychophysics for Making Face Recognition Algorithms More Explainable

Contents

1 Detailed "herder" Algorithm

In this section, we describe the details of the "herding" algorithm described in Sec. 3 of the main paper. The predominant attention of the reader should be on Supp. Figs. 1-7 and the accompanying captions, as the main text of this section will describe the high-level motivation for each step in the process. Supp. Sec. 1.1 will introduce the reader to the foundation of treating similarity matrices in a graph theory context. In Supp. Secs. 1.2-1.4, we describe the complete procedure using a similarity matrix produced with the familiar additive color wheel. By the end of Supp. Sec. 1.4, the reader will understand the theoretical foundation and procedure to "herd" sheep from the remaining *biometric menagerie* as described in Sec. 3 of the main paper.

1.1 Foundation: Similarity Matrix as a Graph

Before one can understand and appreciate the full "herding" procedure, it is necessary to understand the basics of graph theory and the place of a similarity matrix within it. Supp. Fig. 1 compares an adjacency matrix with a similarity matrix and shows they are, for all intents and purposes, one and the same. While Supp. Fig. 1 establishes similarity matrices as adjacency matrices, Supp. Fig. 2 displays the unique properties the similarity matrix has over the adjacency matrix, which enables us to "herd" *sheep* from the *biometric menagerie*.

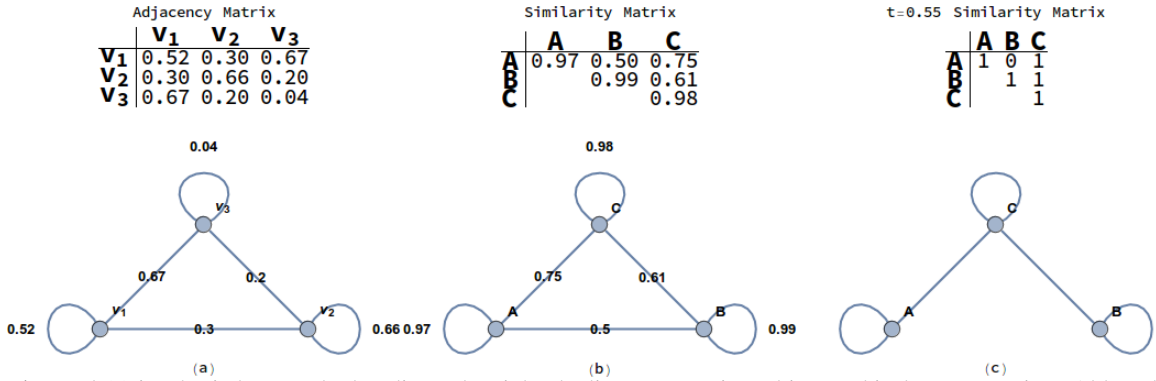

Figure 1. Panel (a) is a basic but standard undirected weighted adjacency matrix and its graphical representation. Although similarity matrices are rarely (if ever) thought of as adjacency matrices, similarity matrices can be operated on and treated as adjacency matrices. Thus, panel (b) contains a typical similarity matrix and its undirected weighted graphical representation. A thresholded similarity matrix produces a similarity matrix containing 1s (matches) and 0s (non-matches). Like the pre-thresholded similarity matrix, the thresholded matrix can be thought of as a graph or more specifically an undirected unweighted graph. Panel (c) contains an example thresholded similarity matrix and its graphical representation.

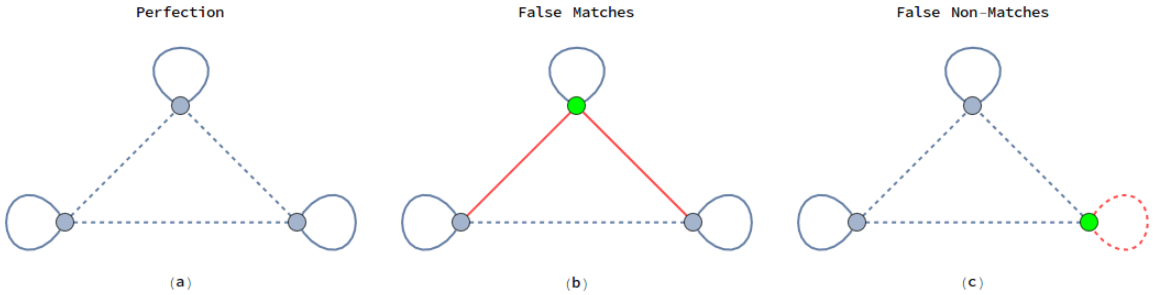

Figure 2. The three basic types of graph edges from thresholded similarity matrices. Panel (a) is the graph for when there are no false matches and no false non-matches. The solid edges represent the true matches and dashed edges represent the true non-matches. Notice, if the thresholded similarity matrix is an identity matrix, this would be its graph representation. Panel (b) is a graph with two solid edges that are supposed to be dashed, making them two false matches instead of true non-matches. Panel (c) has one false non-match where the connected edge of the vertex should be solid but is dashed. In both panels (b) and (c), red highlighted edges indicate the false edges. It is easy to see in panel (b) and (c) that removing the green highlighted vertices would eliminate the false edges. The need to remove the green vertices in order to eliminate the red edges forms the foundation for the complete "herding" algorithm.

1.2 Similarity Matrix to Graph Representation

To make the algorithm easier to follow and visualize, we use the additive color wheel as identities rather than faces. Supp. Figs. 3-4 reintroduce the properties described in Supp. Fig. 2 — using the additive color wheel — to describe the *biometric menagerie* in the context of the graph representation. The reader should note, although visually speaking we use the exact color

wheel (plus black), the entries in the similarity matrix in Supp. Fig. 3 were altered for the purposes of covering all the cases in the algorithm. For realism, we chose white and black identities as the more problematic identities. Because white is the center of the color wheel, it could potentially be too similar or too different to any of the other colors. While white is on the color wheel, we chose black to be an example of an identity which matches poorly with itself because it is not on the color wheel.

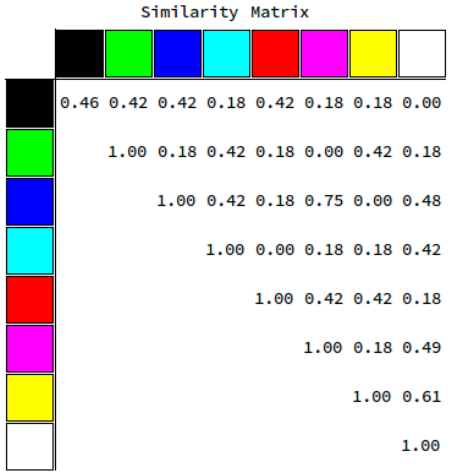

Figure 3. A similarity matrix using the additive color wheel to represent eight identities. Although in the main paper we focus on face identities, in theory, these colors could represent any type of identity such as fingerprints, iris, etc. The values provided in this similarity matrix predominantly come from the RGB representational vector for each color, with a few slight manual alterations to demonstrate the functionality of the described algorithm, e.g., it is unlikely that the black color identity would score so low – relatively speaking – when compared to itself, but by manually altering that comparison value, the matrix provides a possible false non-match for Supp. Fig. 4. As such, the values in the similarity matrix are for demonstration only and have minimal validity within the context of the actual additive color wheel.

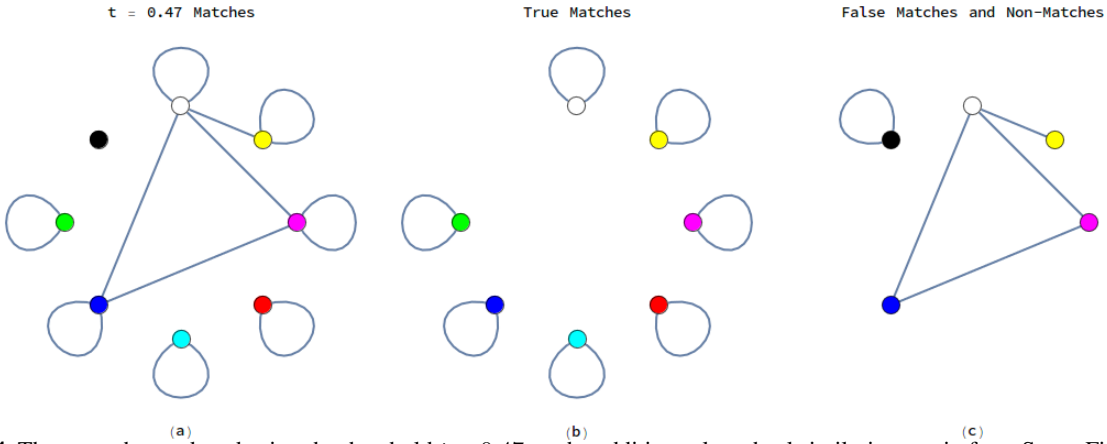

Figure 4. Three graphs produced using the threshold $t = 0.47$ on the additive color wheel similarity matrix from Supp. Fig. 3. Panel (a) contains all of the match edges produced. Panel (b) contains all of the true match edges. Panel (c) contains the false match edges and false non-match edges (missing edges) from (a) — it is equivalent to the thresholded additive color wheel similarity matrix XORed with the identity matrix. The importance of panel (c) is found in understanding that it contains all of the potential *goats*, *lambs*, and *wolves*. A naive approach would opt to remove all the identities in panel (c) thinking the remaining identities would be *sheep*. Although, the assumption that the remaining identities are *sheep* is correct, the naive approach fails to recognize that panel (c) also can (and likely does) contain *sheep* which one should not wish to remove. The algorithm described in Supp. Sects. 1.3-1.4 and Supp. Figs. 5-7 uses the graph in panel (c) to identify and remove non-*sheep*.

1.3 Marking Identities for Removal with a Minimum Spanning Tree

From Supp. Fig. 4, one can see which identities to remove, however as the number of identities linearly increases, the number of edges potentially increase quadratically and thus it would be extremely cumbersome to visually or manually determine which identities should be removed. Accordingly, the focus of this section's figures is to describe the technique for efficiently

and deterministically marking the minimum number of identities for removal, such that absolutely no false matches and non-matches exist within the remaining identities — ensuring all remaining identities meet the criteria of being *sheep*.

This procedure is broken up into two primary steps. The first step (Supp. Fig. 5) is the process of appropriately weighting vertices to ensure that in the second step (Supp. Fig. 6), the minimum spanning tree (MST) algorithm used — Kruskal's algorithm — marks the optimal identities for removal. Recall that an MST is a spanning tree (a tree that includes all vertices) that has the minimal possible sum of weights along the edges of the tree. Thus, by properly weighting the edges in the first step we ensure identities that are more problematic will be more likely marked for removal.

Returning to the first step, it is important to know that Kruskal's algorithm does not consider weighted vertices, so weighting the vertices and computing the MST will not provide what is needed. To combat this issue, we weight the vertices and invert the graph — meaning vertices become edges and edges become vertices. This new graph is what we perform Kruskal's algorithm on in step two, rather than the original graph. Supp. Figs. 5 and 6 work in conjunction with each other to demonstrate the marking of non-sheep identities from the *biometric menagerie*.

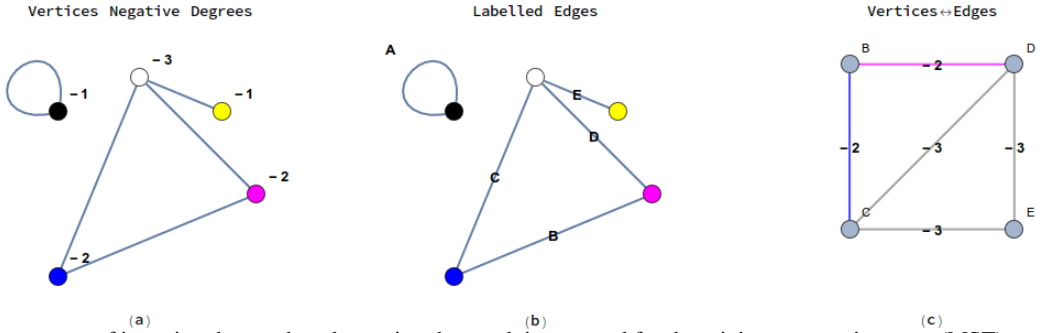

Figure 5. The process of inverting the graph and ensuring the graph is prepared for the minimum spanning trees (MST) process described in Supp. Fig. 6. The most important panel in this figure is panel (c). Panel (c) is the inverted graph of Supp. Fig. 4(c) with the two adjustments from panel (a) and panel (b). Panel (a) contains weighted vertices that become weighted edges in panel (c). Each vertex is weighted by its degree (number of connecting edges) and negated. The assumption is that the higher number of edges a vertex has the more problematic the identity. Negating the value enforces vertices with the highest degree to have the lowest value in the inverted graph, ensuring the MST algorithm will give it favor. Panel (b) is primarily to provide visual labels to be used in panel (c), however one will notice edge A is not present in panel (c). Although the MST algorithm used will handle loops like A, they are easy to identify and removed before inversion. Any loop that is present and then removed is a *goat* within the biometric menagerie. The weights in panel (a), the edge labels in panel (b), and the vertex colors in both correspond to the weights, labels, and colors in panel (c).

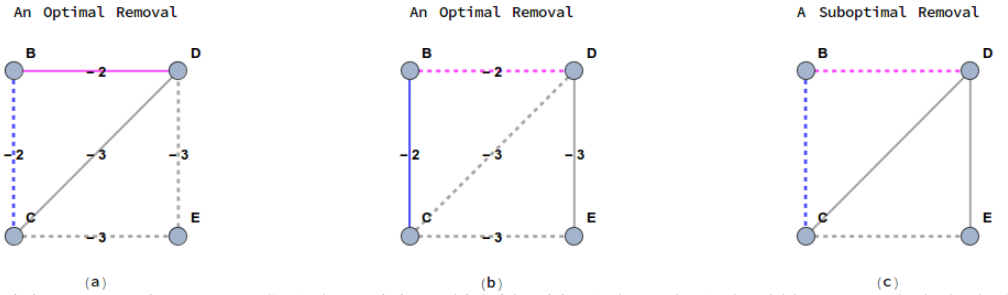

Figure 6. Three minimum spanning trees (MSTs) determining which identities (edge colors) should be removed (dashed edges). Panels (a) and (b) show two of the three possible MSTs using the weighted graph in Supp. Fig. 5(c). Panel (c) is a suboptimal MST resulting from not weighting the edges with the negative degree of the corresponding vertices in Supp. Fig. 5(a) — notice that three identities are marked for removal instead of two. One property of the Kruskal's algorithm used is it only returns one MST and not all possibilities. However, this is perfectly valid for our purposes as our goal here is only to remove the least possible number of identities rather than an exhaustive list of combinations. For example, panel (a) is an MST that marks white and blue color identities for removal whereas panel (b) marks white and magenta color identities. If the graph in Supp. Fig. 5(c) is not connected, Kruskal's algorithm will produce a minimum spanning forest where each connected component is the equivalent of an MST — each sub-MST is valid for marking identities for removal. Finally, marked color identities are either *lambs* or *wolves* — it is impossible to distinguish between them with this procedure. However, we only care about their removal and thus distinguishing *lambs* from *wolves* is not necessary.

1.4 Resulting Graph, Similarity Matrix, and "sheep"

At this point, it has been determined which identities should be removed to remove as few as possible while eliminating all false matches and non-matches, so this section does not provide any new steps in the algorithm rather it highlights the resulting

graph and corresponding similarity matrix. Notice that Supp. Fig. 7 contains the remaining identities for the chosen score threshold $t = 0.47$. However, this can be repeated for any threshold. Each threshold will produce a different set of *sheep*. Thus, it is important to use the entire "herding" procedure described in the main paper Sect. 3 to find the optimal threshold that "herds" the largest number of *sheep*.

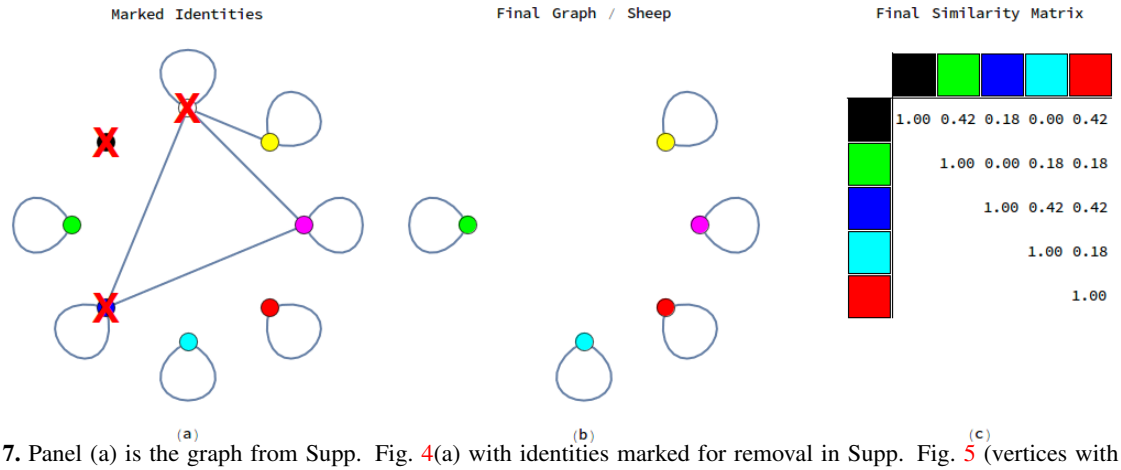

Figure 7. Panel (a) is the graph from Supp. Fig. 4(a) with identities marked for removal in Supp. Fig. 5 (vertices with loops / false non-matches) and identities marked in Supp. Fig. 6(a) (vertices that caused false matches). Panel (b) is the resulting and final graph after problematic vertices were removed. Notice, no false matches or non-matches remain. Had we used Supp. Fig. 6(c) as the marking procedure, only four vertices would remain instead of five. Panel (c) is the final similarity matrix which contains all of our *sheep* and no other *biometric menagerie* identity type.

2 Selected "sheep" 2D Experiments

Each face recognition algorithm required either an aligned face or an unaligned aligned face. FaceNet, OpenFace, and slmsimple use images from the aligned LFW published dataset. OpenBR and VGG-Face use images from the unaligned LFW dataset. See spreadsheet 2D-Sheep.xlxs for the complete list of initial input images that were selected as sheep.

3 Selected "sheep" 3D Experiments

Like in the 2D experiments, each face recognition algorithm required either an aligned face or an unaligned face. However, with FaceGen all faces were by default aligned. For the algorithms that required alignment, we cropped the 3D images down to just the face for consistency with the published 2D alignment. For the algorithms that did not require alignment, we cropped to the entire head. See spreadsheet 3D-Sheep.xlxs for the complete list of initial input images that were selected as sheep.

4 Parameters of Image Transformations

We used the following transformations for our perturbation parameters: Gaussian blur, salt & pepper noise, sharpness, contrast, color, brightness, rotation, scale, linear occlusion, pink noise, and brown noise. To perturb using Gaussian blur, we used OpenCV's `GaussianBlur` module [1], and set the blur kernel σ in the range $[0, 15]$. Salt & pepper noise utilized the `util` module from scikit-image [2], and a random configuration of noise was selected as a percentage of the total image pixels. Sharpness, contrast, color, and brightness were controlled using the `ImageEnhance` module of PIL [3]. The `Sharpness` was chosen within the range $[0.0, 2.0]$. For both the `Contrast` and `Brightness` functions, the controlling parameter was set between $[0.0, 15]$.

There are three non-standard transformations (not a single API call), linear occlusion, pink noise, and brown noise. The linear occlusion technique is the method for linear occlusion described in RichardWebster et al. [4]. Pink and Brown noise use a Python implementation of [5] but modified to be a percent of noise added to base pixel value. (noise * percent * pixel + pixel). We defined β input parameters described in [5] as $\beta = 1$ for pink noise and $\beta = 2$ for brown noise — both recommended in the comments of the referenced algorithm.

5 Supplemental Figures 2D Experiments (Plots)

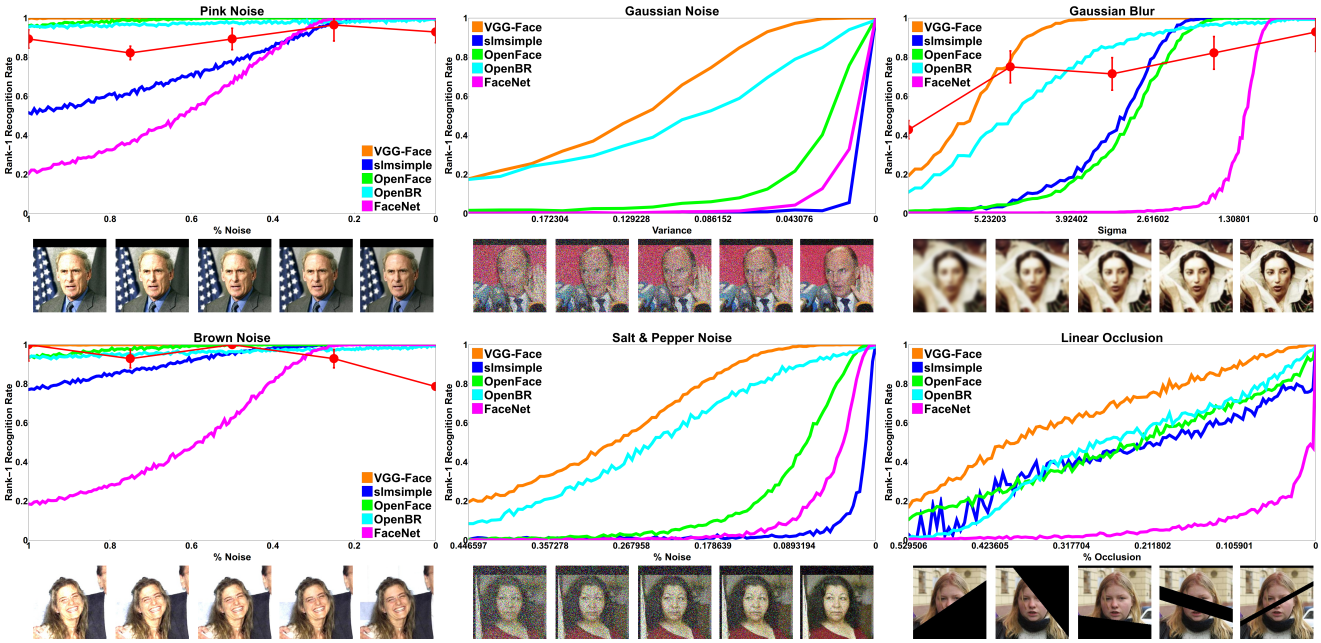

Figure 8. 2D item-response curves for transformation functions: noise, occlusion, and blur

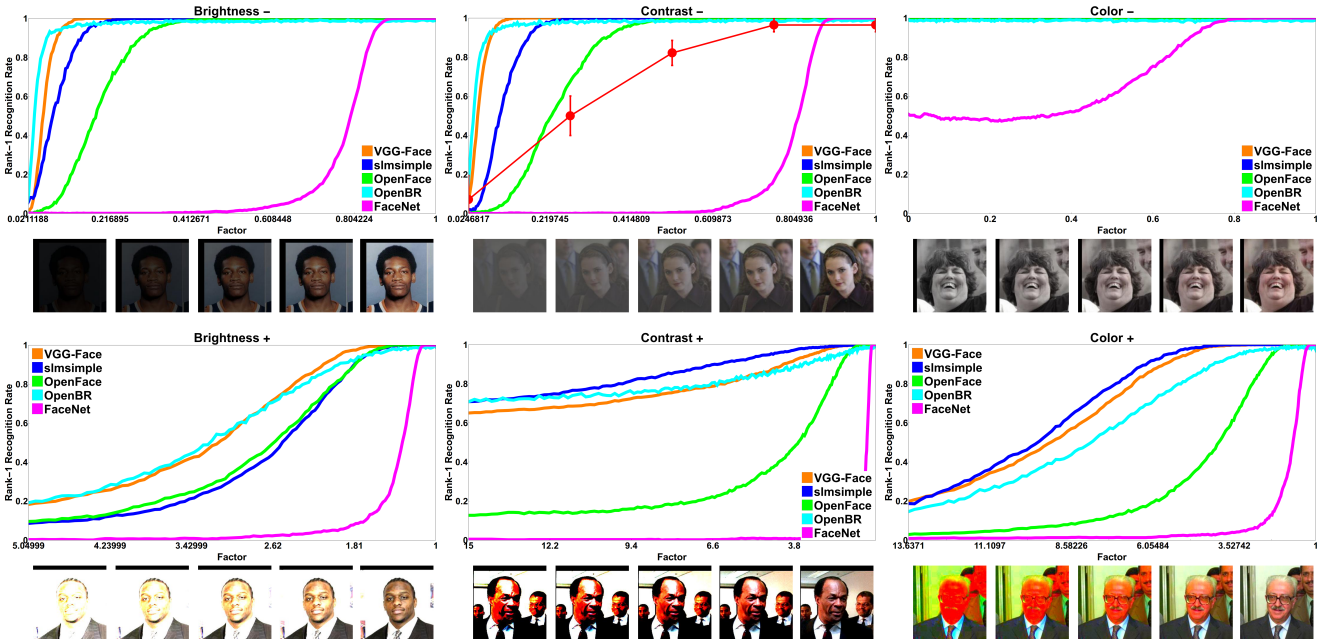

Figure 9. 2D item-response curves for transformation functions: `ImageEnhance` module of PIL [3]. Saturation omitted because of uninformative results.

6 Supplemental Figures 3D Experiments (Plots)

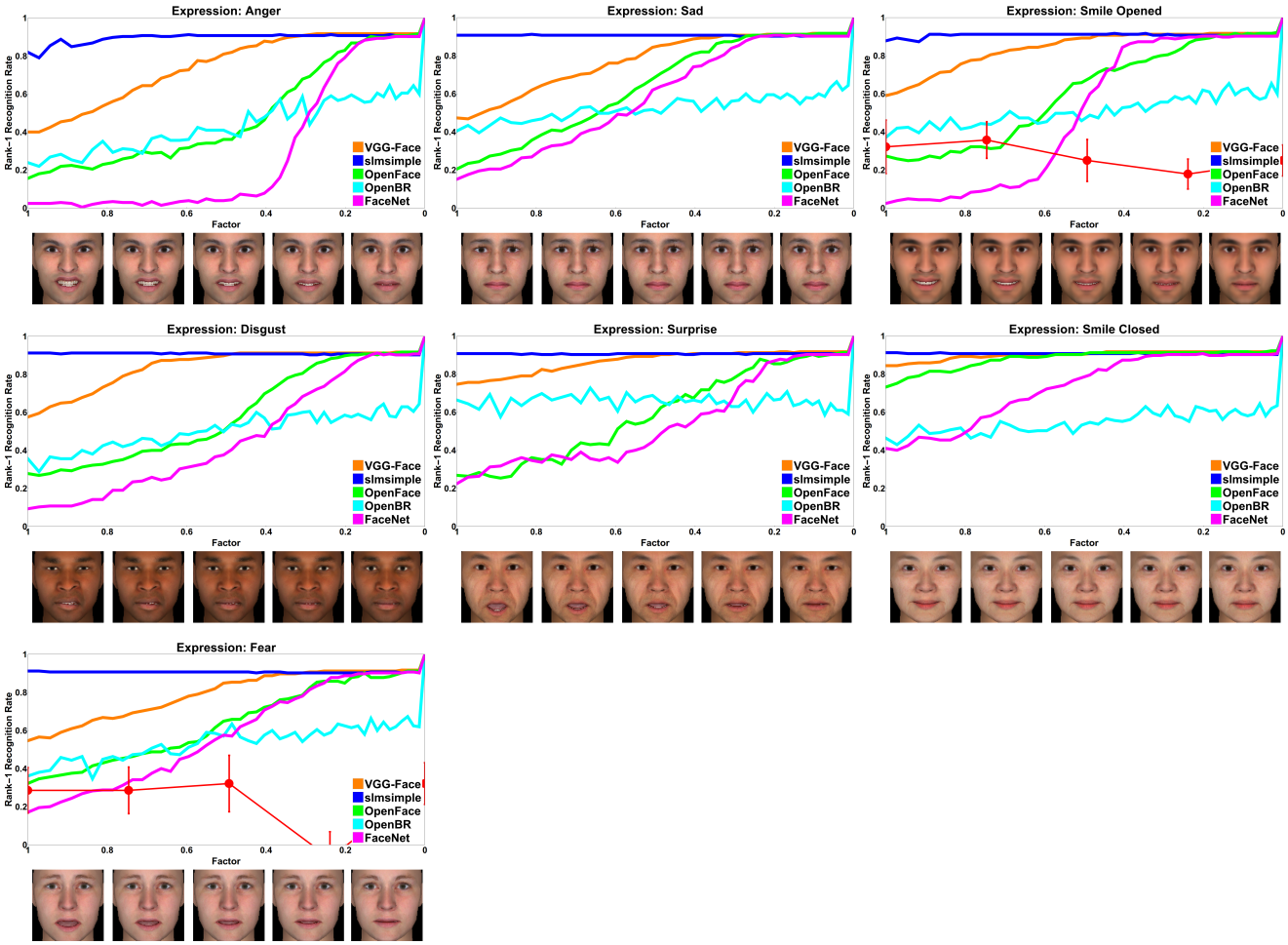

Figure 10. 3D item-response curves for FaceGen `Expression`

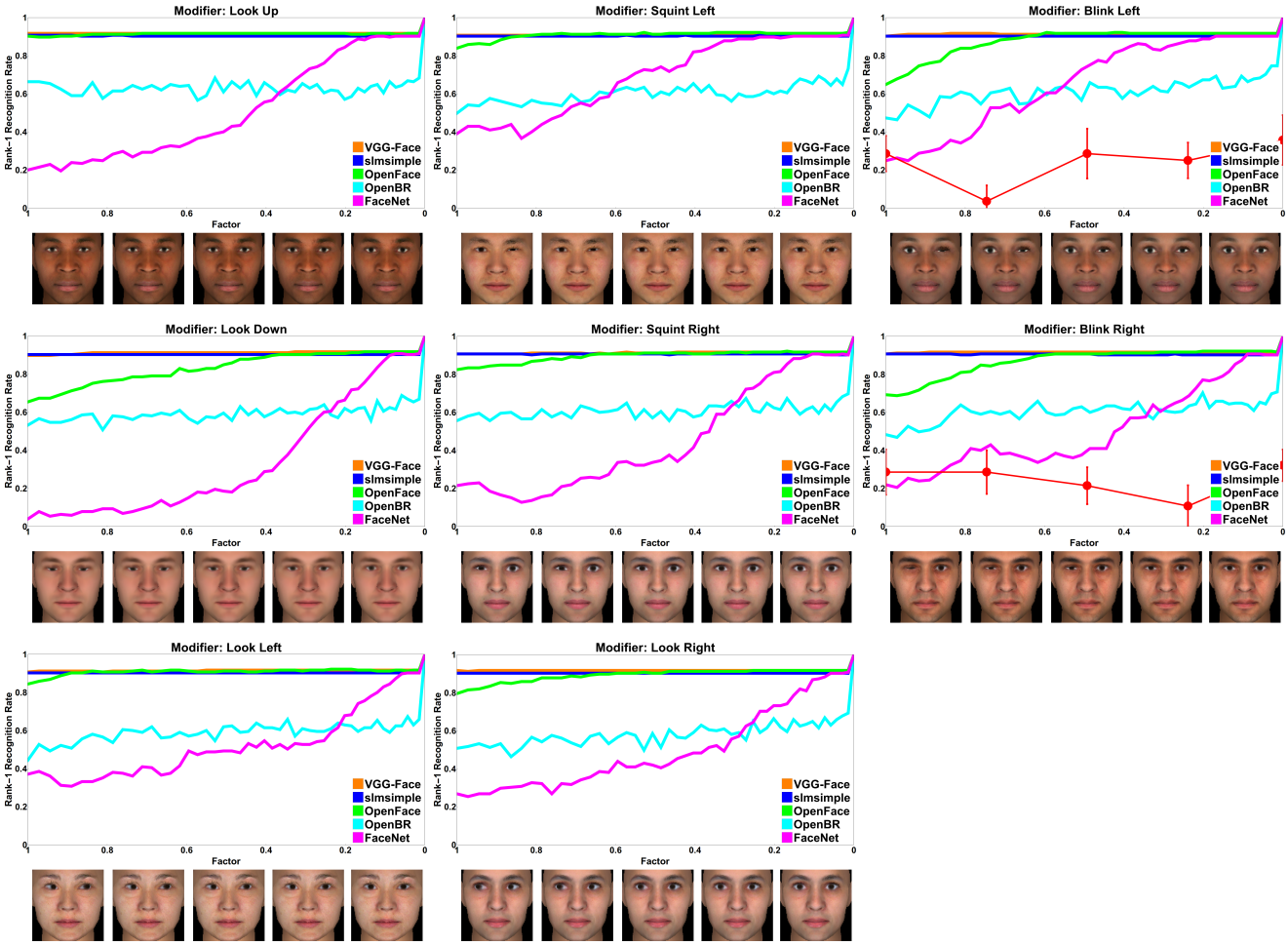

Figure 11. 3D item-response curves for FaceGen `Modifier`. FaceGen images created with `Modifier:Brow*` omitted because of uninformative results.

7 Supplemental Figures Weight Perturbation Experiments (Plots)

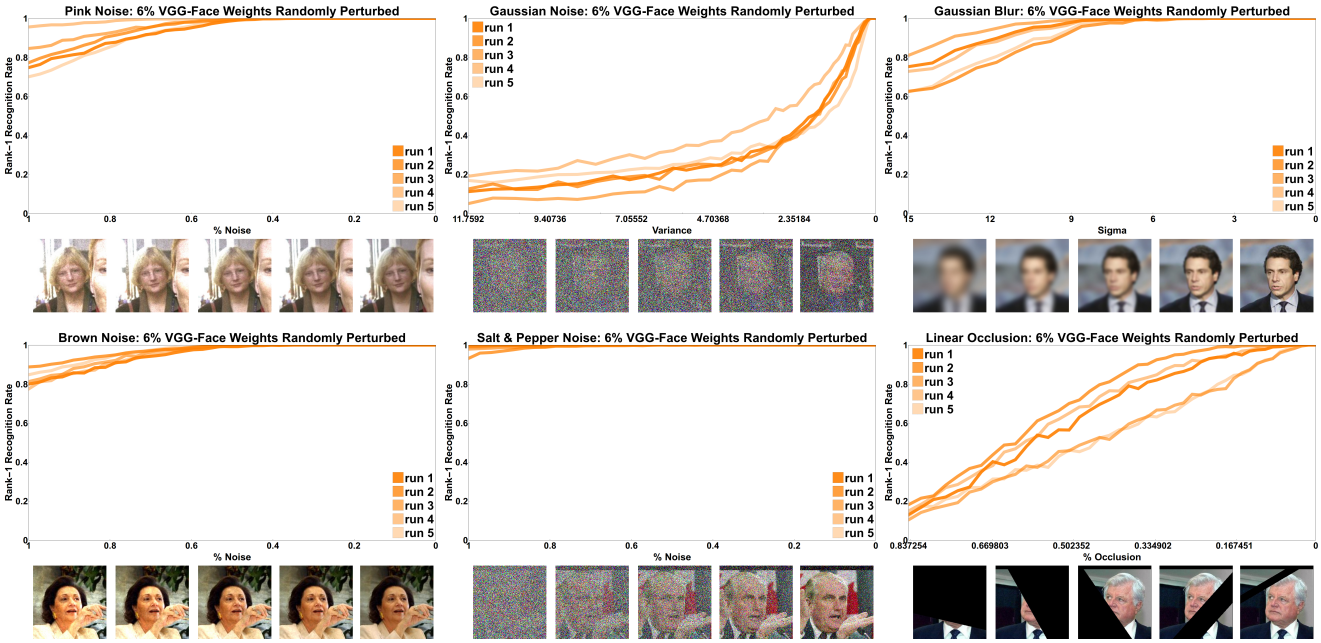

Figure 12. 6% Weight Perturbation item-response curves for transformation functions: noise, occlusion, and blur

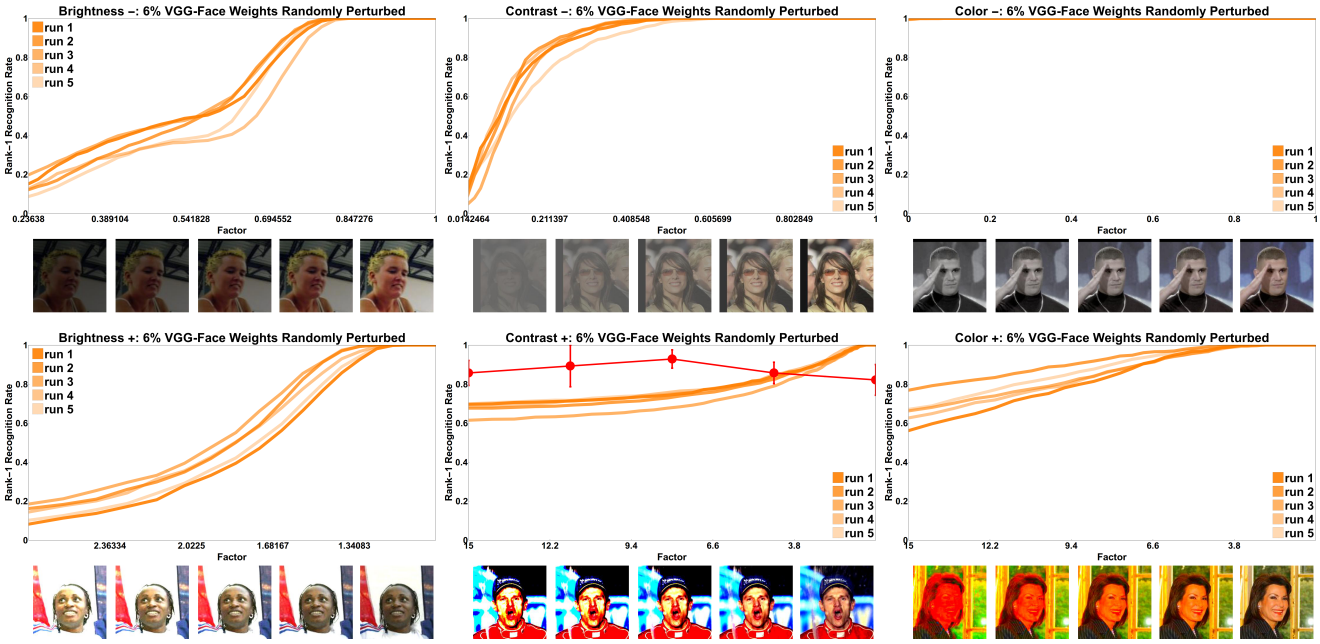

Figure 13. 6% Weight Perturbation item-response curves for transformation functions: `ImageEnhance` module of PIL [3]. Saturation omitted because of uninformative results.

References

[1] G. Bradski, "OpenCV," *Dr. Dobb's Journal of Software Tools*, 2000. 5

[2] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 2014. 5

[3] Pillow (PIL Fork), "https://pillow.readthedocs.io/en/3.0.0/index.html," Accessed: 2016-11-05. 5, 6, 9

[4] B. RichardWebster, S. E. Anthony, and W. J. Scheirer, "Psyphy: a psychophysics driven evaluation framework for visual recognition," *arXiv preprint arXiv:1611.06448*, 2016. 5

[5] J. Yearsley, "spatialpattern," May 2004. [Online]. Available: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/5091/versions/2/previews/spatialPattern.m/index.html?access_key= 5