Supplementary Materials for "Discriminately Decreasing Discriminability with Learned Image Filters"

1 Experiment I: synthetic data – supplementary results

We compared the effectiveness of the "nullspace filter" to DDD on the horizontal and vertical lines task: For the nullspace filter we varied $k \in \{25, 50, 75, 100, 200\}$ and chose $k = 100$ to optimize performance by visual inspection. We then applied the nullspace filter to a set of 120 images (40 unfiltered, 40 filtered with DDD, and 40 filtered with the nullspace filter), presented them to 20 human observers from the Mechanical Turk, and calculated labeling accuracy of target and distractor attributes. On the unfiltered images, average labeling accuracy (across the labelers) was 99.5% for the target task and 95% for the distractor task. Using the nullspace filter, target task labeling accuracy was 98% and distractor task labeling accuracy was 86.75%. Finally, using DDD, target task labeling accuracy was 97.50% and distractor task labeling accuracy was 53.25% – in other words, DDD was substantially more effective at suppressing the distractor task than the nullspace filter.

2 Experiment II: natural face images – supplementary results

For the natural face image labeling tasks, in addition to combining opinions of the 10 workers on Mechanical Turk using Majority Vote, we also tried using the GLAD method by Whitehill, et. al [1] for combining multiple opinions when the quality of the labelers is unknown a priori. Their algorithm is called GLAD (Generative model of Labels, Abilities, and Difficulties) and its source code is available online.

In general the results were similar to Majority Vote:

Accuracy (2AFC) of labels from Amazon Mechanical Turk using GLAD [1] to combine opinions

Filter method	**Expression**	**Gender**
Unfiltered (baseline)	94%	98%
Learned filter 1: Preserve expression, suppress gender	94%	54%
Manually constructed filter: (show mouth region only)	98%	72%
Learned filter 2: Preserve gender, suppress expression	64%	92%

3 Derivatives of linear convolution filter

Here, we derive $\frac{\partial f}{\partial \theta_{ij}}(\theta, x)$ where f represents clipped 2-D image convolution. By clipped, we mean that the size of the filtered image f is the same as the size of the input image x.

Consider for the moment the case of 1-D convolution of a 3-element kernel θ with a 3-element image x:

$$\theta = \begin{bmatrix} a & b & c \end{bmatrix} \quad (1)$$

$$x = \begin{bmatrix} r & s & t \end{bmatrix} \quad (2)$$

$$\theta * x = \begin{bmatrix} ar & as+br & at+bs+cr & bt+cs & ct \end{bmatrix} \quad (3)$$

$$(4)$$

In our paper, we "clip" the convolution operation's output so that it retains the same size as the input x; hence, we define:

$$\text{Clip}(\theta * x) = \begin{bmatrix} as+br & at+bs+cr & bt+cs \end{bmatrix} \quad (5)$$

(6)

We can now differentiate $\text{Clip}(\theta * x)$ w.r.t. each dimension of θ:

$$\frac{\partial}{\partial a}(\text{Clip}(\theta * x)) = \begin{bmatrix} s & t & 0 \end{bmatrix} \quad (7)$$

$$\frac{\partial}{\partial b}(\text{Clip}(\theta * x)) = \begin{bmatrix} r & s & t \end{bmatrix} \quad (8)$$

$$\frac{\partial}{\partial c}(\text{Clip}(\theta * x)) = \begin{bmatrix} 0 & r & s \end{bmatrix} \quad (9)$$

$$(10)$$

Hence, the derivatives of the clipped convolution are computed by "sliding" the row vector x across a row of 0's and clipping at the appropriate indices. For the case of finite discrete 2-D convolution, the situation is analogous – the gradient of the 2-D convolution $\theta * x$ is found by "sliding" the image matrix x over a matrix of 0's.

While it is possible to specify in mathematical notation the exact indices used for sliding and clipping, it is tedious and unilluminating. Instead we provide a snippet of Matlab code for computing $\frac{\partial}{\partial \theta}\text{Clip}(\theta * x)$ for 2-D image convolution with 0-padding (no wrap-around):

```
function dConvdtheta = computedConvdtheta (x, theta, j)
% dConvdtheta = COMPUTEDCONVDTHETA (x, theta, j) computes the derivative with respect to theta_jj
% of conv(x, theta).

    m = sqrt(length(theta));                    % theta corresponds to m x m convolution kernel
    n = sqrt(length(x));                        % x corresponds to n x n image
    idxs = reshape(1:length(theta), [ m m ]);
    [r,c] = find(idxs == j);

    xim = reshape(x, [ n n ]);
    dConvdthetaim = zeros(m+n-1, m+n-1);
    dConvdthetaim(r:r+n-1, c:c+n-1) = xim;

    % Trim the matrix back down to only be the "center" part of the convolution result
    upperPadding = ceil((size(dConvdthetaim, 1) - n) / 2);
    leftPadding = ceil((size(dConvdthetaim, 2) - n) / 2);
    lowerPadding = size(dConvdthetaim, 1) - n - upperPadding;
    rightPadding = size(dConvdthetaim, 2) - n - leftPadding;
    dConvdthetaim = dConvdthetaim(1+upperPadding:end-lowerPadding, 1+leftPadding:end-rightPadding);
    dConvdtheta = dConvdthetaim(:);
end
```

4 Derivatives of element-wise "mask" filter

If we let the filter $f(\theta, x) = \theta x$, where θ is a diagonal matrix, then

$$\begin{aligned} \frac{\partial}{\partial \theta_{jj}} f(\theta) &= \frac{\partial}{\partial \theta_{jj}} (\theta x) \\ &= E_j x \end{aligned}$$

where E_j is a $d \times d$ matrix consisting of all 0's except the (j, j)th entry, which is 1.

References

[1] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Neural Information Processing Systems*, 2009.