# DSSC Segment 1 Project

Marwan Elkrewi, Sreyam Sengupta

April 28, 2021

## 1 Feature extraction using PCA

For this section we use the Ruderman dataset again, where each image file was a $256 \times 256$ array of grayscale luminance values. Let $L_i(x, y)$ represent the luminance of the pixel at $(x, y)$ position of image $i$. We take the log transform as follows:

$$L_i(x, y) \leftarrow \ln L_i(x, y) \tag{1}$$

and then subtract the mean log luminance for each image:

$$L_i(x, y) \leftarrow L_i(x, y) - \langle L_i \rangle_{x,y} \tag{2}$$

where the average in RHS of equation (2) is taken over all pixel values for a single image. The images so transformed are shown in figure (1).

Now, we pick up a large number of $7 \times 7$ patches from these mean subtracted log luminance image arrays. The details of how we do this are not important; however for the sake of completeness we mention it briefly. Each image is divided into as many $8 \times 8$ patches as we can. A $7 \times 7$ patch is chosen from each $8 \times 8$ patch by omitting the last row and last column. These patches are then stored as 49-component vectors. Because we have 45 images, and each image can be divided into $32 \times 32 = 1024$ patches ($32 \times 8 = 256$), we have a total of $1024 \times 45 = 46080$ patch vectors.

Now, we can do PCA on these patch vectors. We use the `sklearn.decomposition.PCA` utility from `scipy` for this. The 46080 patch vectors that we have collected live in a 49-dimensional space, and our PCA returns an ordered list of 49 mutually orthonormal basis vectors, called **principal components,** which span this space. The first of these is the direction along which the variance is greatest; the second is the direction of second-highest variance, and so on.

Now, we may transform these 49-dimensional components into $7 \times 7$ arrays and represent them as images. In general, relating these components to actual observable features is difficult. But because these are images, the hope is that these components correspond to certain observable and identifiable **motifs,** the simple repeating patterns that make up complex images. The components, in grayscale pixel array form, are shown in figure (2). The first few can be immediately recognized as recurring motifs from the images. The first component accounts for over 40% of the variance. Visually, it looks like a dark spot - black center, darker body, becoming lighter towards the edges. Components two and three correspond to a vertical and a horizontal edge respectively. Component four is a bright vertical line surrounded by darkness,

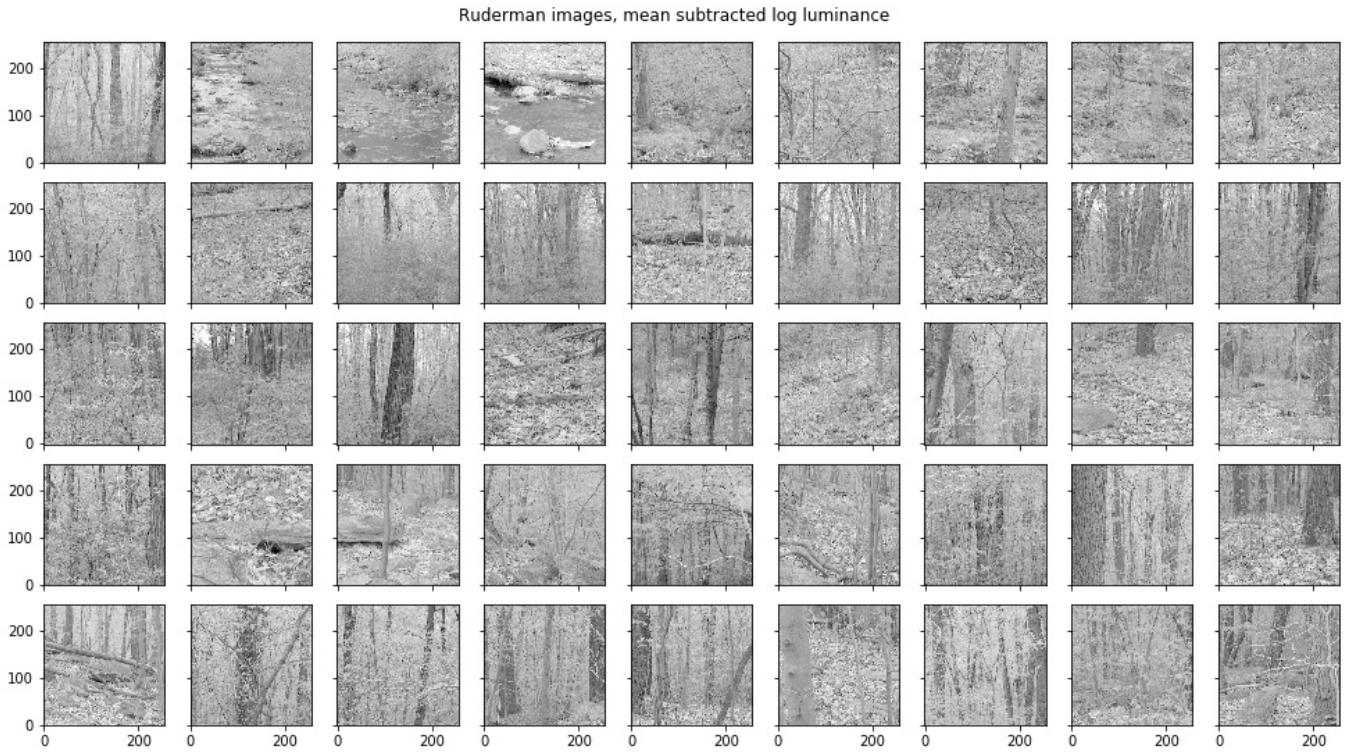


Figure 1: (Q1) The mean subtracted log luminance transform of the Ruderman images.

while component five is a dark horizontal line surrounded by bright regions. Component six is two dark corners diagonally opposite each other, while component seven is alternating bright and dark vertical lines. Latter components get progressively more complex.

The eigenvalues (which measure how much each component explains the total variance) are plotted against their rank in figure (3). The first feature is by far the most important, and alone accounts for over 40% of the total variance. We also plot a cumulative sum of variance explained by first $n$ components, taking $n$ from 1 to 49. This is showed in figure (4).

# 2 ICA: centering and whitening patch vectors

We take 11250 mutually exclusive $7 \times 7$ patches from the complete Ruderman image set. This time, we do not do the log transform, or the mean subtraction beforehand. The patches are flattened, and stored as the rows of a $11250 \times 49$ matrix $\mathbf{X}$.

First we center $\mathbf{X}$. For this, the mean over all rows is subtracted from each row of $\mathbf{X}$. Now, the 49-dimensional patch vectors that make up each row of $\mathbf{X}$ have zero mean.

Now, we whiten the data. For this, we need the $49 \times 49$ covariance matrix of $\mathbf{X}$, call it $\mathbf{C_X}$. We find the eigenvalues and corresponding right eigenvectors of $\mathbf{C_X}$. Let $\mathbf{D}$ be the diagonal matrix of eigenvalues, while $\mathbf{V}$ be the matrix where each column is the corresponding right eigenvector of $\mathbf{C_X}$.

Then, the whitening transformation of $\mathbf{X}$ becomes:

$$\mathbf{X'} = \mathbf{XVD}^{-1/2}\mathbf{V^T} \tag{3}$$

and $\mathbf{X'}$ is the whitened version that we want. We checked if $\mathbf{X'}$ really has unit covariance. The result, plotted in figure (5), shows that it does.

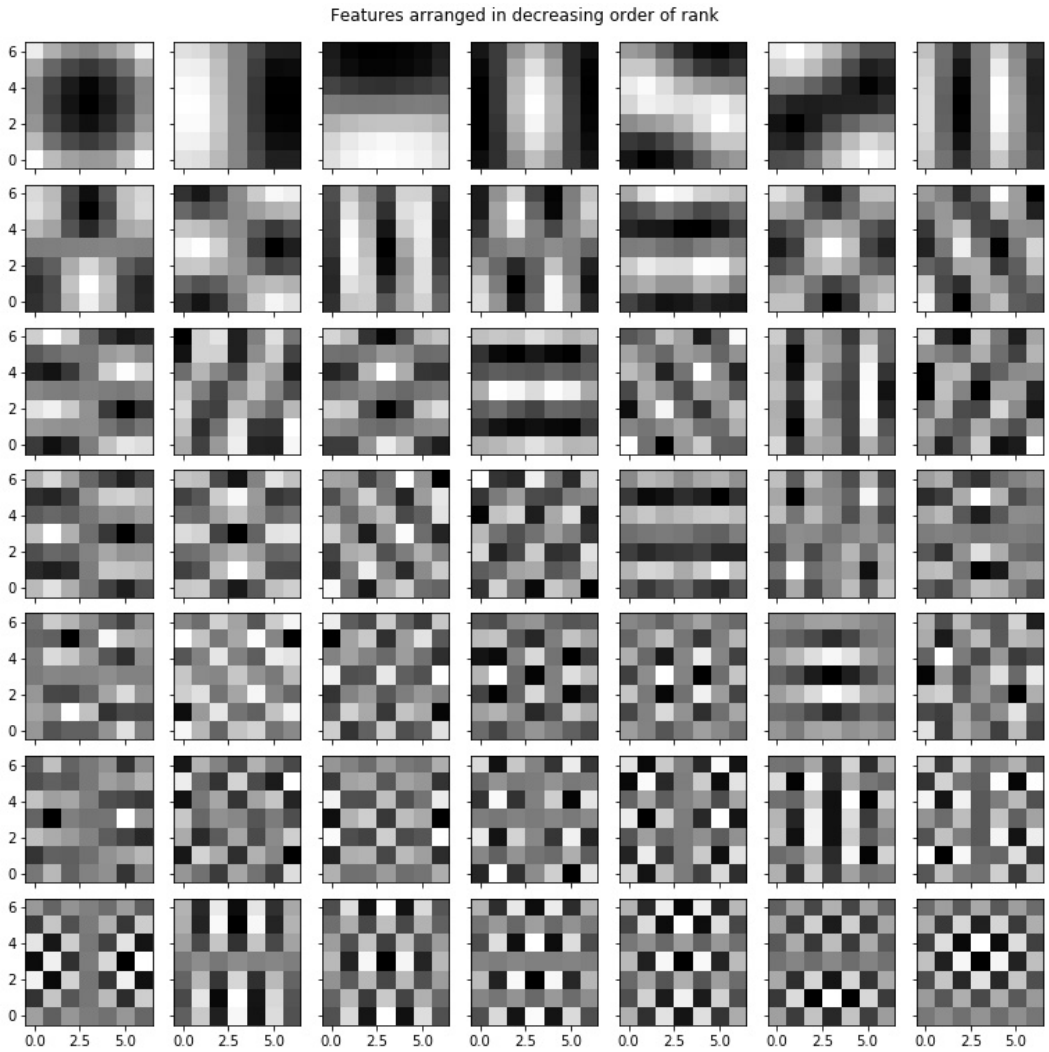


Figure 2: (Q1) The 49 principal components, in $7 \times 7$ grayscale pixel array form, arranged in decreasing order of variance explained (eigenvalue), left to right and top to bottom. Component one seems to be a dark spot in a brighter background, two and three are vertical and horizontal edges respectively, four is a bright vertical line in dark background, five is a dark horizontal line in bright background. Latter components get more complex.

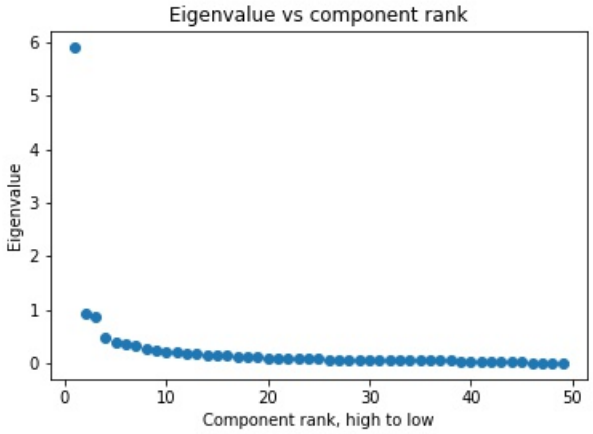


Figure 3: (Q1) The eigenvalues of the 49 principal components, arranged according to rank. We see the first is by far the largest, and in fact accounts for over 40% of the total variance.

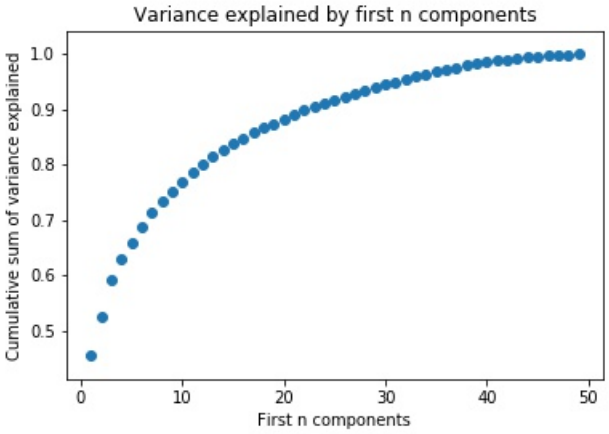


Figure 4: (Q1) A cumulative sum of variance explained by first $n$ components vs $n$, for $n \in \{1, 2, \ldots, 49\}$.

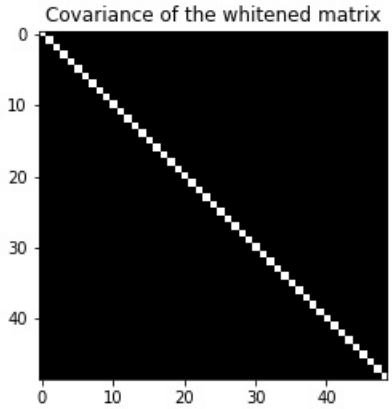


Figure 5: (Q2) The covariance of the centered and whitened data matrix $\mathbf{X}'$. We see only the diagonal elements remain and are equal - everything else is zero.

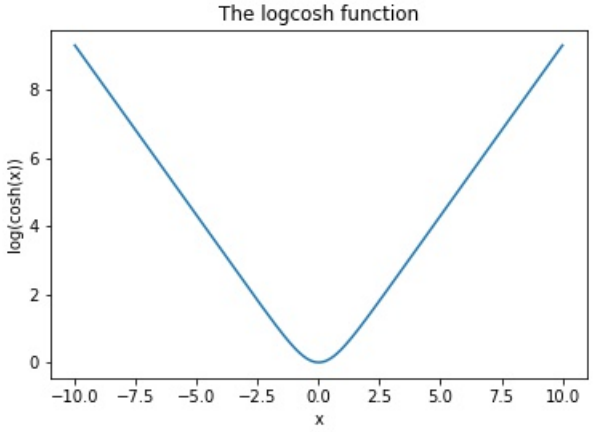


Figure 6: (Q3) A graph of $\log\cosh(x)$ vs $x$, showing the various properties discussed above. We see that for small $x$, it resembles a quadratic in $x$, while for large $x$, it looks like the absolute value of $x$.

# 3 ICA: Optimizing the objective function

Now that we have a data matrix $\mathbf{X}'$ that has been suitably centered and whitened, its first and second moments are trivial - zero mean and unit variance. The task now is to look for higher-order statistical structure. This we shall do using a linear filter $\mathbf{w}$. This filter is a 49-component unit vector chosen in such a way that the projections of the data matrix $\mathbf{X}'$ on $\mathbf{w}$, i.e. the quantities $\mathbf{X}'\mathbf{w}$ are sparse - mostly zero, with a few entries of large magnitude compared to what is expected from a Gaussian. This filter is our test for non-Gaussianity.

The question is, what is a good choice for the filter $\mathbf{w}$? We follow the ML paradigm - define a scalar function of $\mathbf{w}$ (of course involving $\mathbf{X}'\mathbf{w}$ somehow as a fixed parameter) and optimize this function, called the **objective function.** If this objective function is convex, this guarantees the existence and uniqueness of a global minimum. In our case, we use the logcosh function. This has some nice properties: as $x \to 0$, $\log\cosh(x) \approx x^2/2$, while for very large x, $\log\cosh(x) \approx |x| - \log 2$. So in the small $x$ regime, this function is differentiable, unlike the mean absolute error (MAE) or L1 loss. And in the large $x$ regime, $\log\cosh(x) \sim |x|$ and not $x^2$, which makes this loss function more robust against large outliers than the mean squared error (MSE) or L2 loss. Also, the logcosh function is globally convex and twice differentiable everywhere, which is important because many optimization algorithms require the second derivative of the objective function. A plot of $\log\cosh(x)$ vs $x$ is shown in figure (6).

We tried three different methods to solve this problem. Two were using pre-defined FastICA algorithms, while the third was by optimizing an objective function ourselves.

## 3.1 Optimization of the logcosh function

Let $\mathbf{x}_t$ denote the $t$-th whitened patch, i.e. the $t$-th row of $\mathbf{X}'$. We want to project every whitened patch $\mathbf{x}_t$ on the filter $\mathbf{w}$ by taking their dot product $\mathbf{x}_t \cdot \mathbf{w}$. This is then passed to a logcosh function and the average over all patches are calculated, a quantity of the form $T^{-1}\sum_{t=1}^{T}\log\cosh(\mathbf{x}_t \cdot \mathbf{w})$. In terms of the whitened data matrix $\mathbf{X}'$, we define the objective

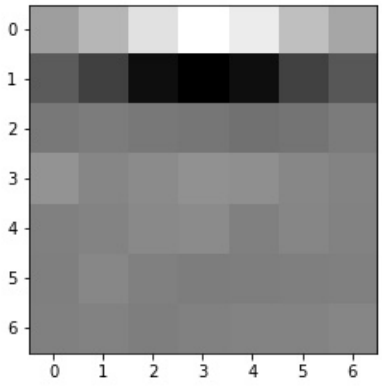


Figure 7: (Q3) $\mathbf{w}^*$ obtained from an initial value $\mathbf{w}_0 = (0, 0, 1, 0, \ldots, 0)^T$. As we can see, most of the entries are pretty close to zero, but a few, near the top center, have highly positive or negative values. This is what we might expect from a sparse filter.

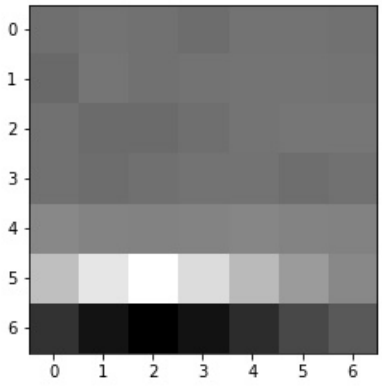


Figure 8: (Q3) $\mathbf{w}^*$ obtained from an initial value $\mathbf{w}_0$ where the 38th element is one, and all others zero. This looks qualitatively similar to figure (7), except that the prominent feature is towards the bottom left of the image.

function as follows:

$$f = \frac{1}{T} \sum_{i=1}^{T} \log \cosh \left( \sum_{j=1}^{49} X'_{ij} w_j \right) \tag{4}$$

where $X'_{ij}$ and $w_j$ are the $(i, j)$-th and $j$-th components of $\mathbf{X}'$ and $\mathbf{w}$ respectively. Note that we have used a different sign convention compared to the text - we used the `minimize` function from the `scipy.optimize` library, so we minimized the negative of the function being maximized in the text. The function $f$ is optimized subject to the constraint that $\|\mathbf{w}\| \in [1 - \epsilon, 1 + \epsilon]$, and the tolerance chosen as $\epsilon = 10^{-10}$.

While the logcosh function certainly has a unique optimum, how close our numerical optimizer actually reaches to that optimum depends on, among other factors, the initial guess $\mathbf{w}_0$. Also, different initial values could lead to slightly different results. We ran the optimizer with several different initial conditions. The resulting optimal values of $\mathbf{w}^*$ are plotted as $7 \times 7$ images, shown in figures (7), (8), (9), and (10).

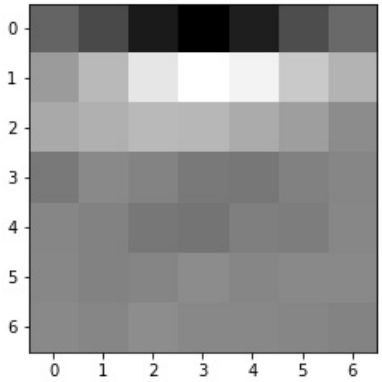


Figure 9: (Q3) $\mathbf{w}^*$ obtained from an initial value $\mathbf{w}_0 = (1/7, 1/7, 1/7, \ldots, 1/7)$.

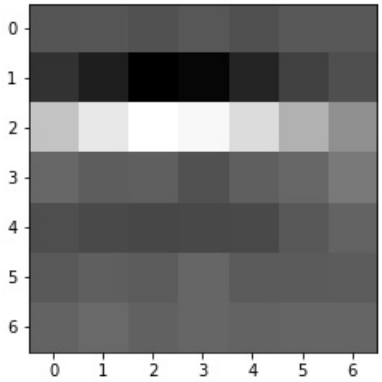


Figure 10: (Q3) Finally, $\mathbf{w}^*$ obtained from a random $\mathbf{w}_0$. Its elements were chosen at random and then it was normalized.