Yale Faces B

The code used for the following questions and figures was done in Julia and is attached to the end of this document. We start with an analysis of the cropped images.

Exercise 1. Do an SVD analysis of the images (where each image is reshaped into a column vector and each column is a new image).

Solution 1. I separated the data into a train-test set with 70% training and 30% test. Each training image was flatten into a column vector $\mathbf{x}_j$ and then combined into the training matrix $\mathbf{X}$. I then did the SVD on training data matrix $\mathbf{X}$.

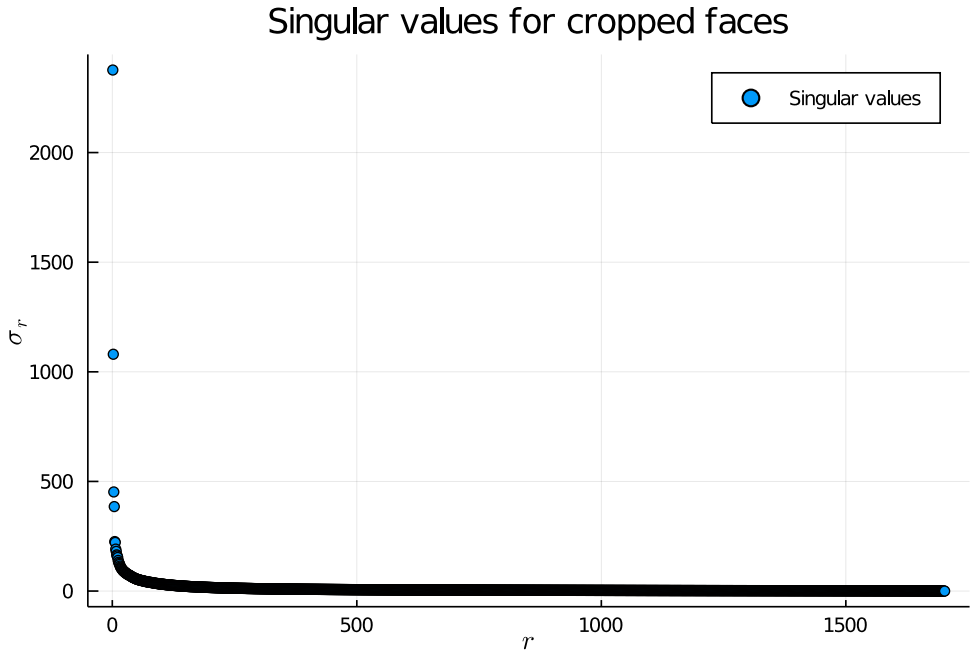

Figure 1: The singular values of the SVD of the training data matrix $\mathbf{X}$ for the cropped data set.

I also did this same analysis on the uncropped and unaligned data but with train-test split 80-20.

Exercise 2. What is the interpretation of the $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ matrices? (Plot the first few reshaped columns of $\mathbf{U}$)

Solution 2. Personally, I like to think of the SVD in the terms of the following equation.

$$\mathbf{XV} = \mathbf{U\Sigma}. \tag{1}$$

In this interpretation, $\mathbf{V}$ is a change of coordinates of $\mathbf{X}$, so that $\mathbf{X}$ can be reduced to rotations and scalings. This way, since each column of $\mathbf{X}$ corresponds to a face, we can think about the matrix $\mathbf{U}$ as describing a basis in a new vector space of faces. We can plot its column vectors $\mathbf{u}_j$ as faces as in Fig. 3 and Fig. 4 since they have the same shape as our input images. The singular values σ_j are a measure of the weights of the corresponding column $\mathbf{u}_j$.

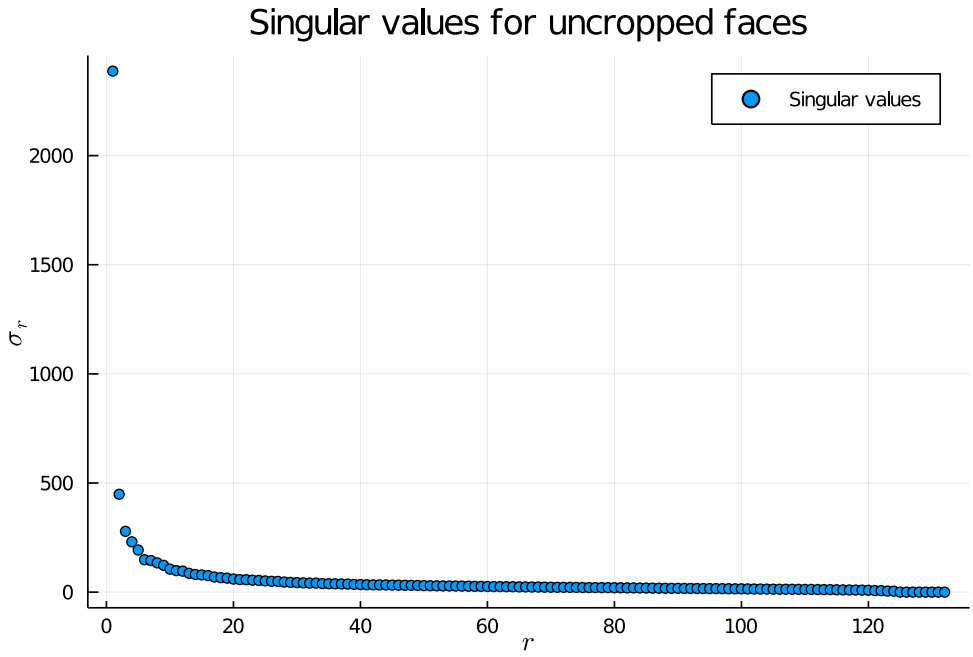

Figure 2: The singular values of the SVD of the training data matrix $\mathbf{X}$ for the uncroppped data set.

Exercise 3. What does the singular value spectrum look like and how many modes are necessary for good image reconstructions using the PCA basis? (i.e. what is the rank r of the face space?)

Solution 3. The singular value spectrum for the cropped faces is plotted in 1 and 2. With both data sets, we can use the test set to analyze how rank is need to be able to sucessfully recreate a face. We can see this tested in Fig. 5 and Fig. 6. When attempting to reconstruct new faces with the cropped data set, common features like noses and eyes quickly appear, but finer details like moles, scars, eyebrow shapes require higher rank approximations. When it comes to the uncropped data set, it's a bit of a disaster which I'll discuss more in the next problem.
I also experimented with determining the cutoff necessary for a good reconstruction using percentage energy to cutoff. That is, I wanted to find the minimal rank r so that

$$\sum_{i=1}^{r} \sigma_i < \lambda \sum_{i=1}^{n} \sigma_i, \tag{2}$$

for some threshold percentage $\lambda \in (0, 1)$. I visualize where these thresholds fall for the cropped data in Fig. 7. This method allowed me to generate percent energy reconstructions as in Fig. 9 and Fig. 10.

Exercise 4. Compare the difference between the cropped (and aligned) versus uncropped images in terms of singular value decay and reconstruction capabilities.

Solution 4. We can compare the rate of singular value decay by using the thresholding discussed in the previous exercise. Comparing figures Fig. 7 and Fig. 8, we see that the

uncropped data has a lower increase in energy. Additionally, from our attempts at reconstruction, we can see that the unaligned data performs poorly because the variance in the pixels for each image is not attributable to differences in the faces themselves but mostly due to misalignment of the faces. This misalignment is also seen in the uncropped eigenfaces in Fig. 4. This leads to a poor fit as the SVD cannot pick up on the underlying structure of the faces.

Theorems

Exercise 5. The non-zero singular values of $\mathbf{A}$ are the square roots of the non-zero eigenvalue of $\mathbf{AA}^*$ or $\mathbf{A}^*\mathbf{A}$.

Solution 5. Let $\mathbf{A} = \mathbf{U\Sigma V}^*$. Then we can write

$$\mathbf{AA}^* = \mathbf{U\Sigma\Sigma}^*\mathbf{U}^{-1}, \tag{3}$$

$$\mathbf{A}^*\mathbf{A} = \mathbf{V\Sigma\Sigma}^*\mathbf{V}^{-1} \tag{4}$$

using the fact that $(\mathbf{AB})^* = \mathbf{B}^*\mathbf{A}^*$ and that both $\mathbf{U}$ and $\mathbf{V}$ are unitary. Notice that that both $\mathbf{\Sigma\Sigma}^*$ and $\mathbf{\Sigma}^*\mathbf{\Sigma}$ are both diagonal with entries $(\sigma_1^2, \sigma_2^2, \ldots)$ which are the square of the non-zero singular values σ_i. Looking at the above equations, we additionally see that $\mathbf{AA}^*$ and $\mathbf{A}^*\mathbf{A}$ are diagonalizable. Therefore, the diagonal entries of the inner matrices $\mathbf{\Sigma\Sigma}^*$ and $\mathbf{\Sigma}^*\mathbf{\Sigma}$ give their eigenvalues λ_i. This gives us that

$$\lambda_i = \sigma_i^2 \implies \sigma_i = \sqrt{\lambda_i}. \tag{5}$$

Exercise 6. If $\mathbf{A} = \mathbf{A}^*$, then the singlar values are the absolute values of the eigenvalues of $\mathbf{A}$.

Solution 6. Since the singular values are the square roots of the non-zero eigenvalues of $\mathbf{A}^*\mathbf{A}$ and $\mathbf{A}$ is Hermitian, we have

$$\mathbf{A}^2 = \mathbf{A}^*\mathbf{A}. \tag{6}$$

Therefore, the singular values of singular values are simple the square root of the eigenvalues of $\mathbf{A}^2$. Since $\mathbf{A}$ is diagonalizable, we have that

$$\mathbf{A}^2 = (\mathbf{P\Lambda P}^{-1})^2 = \mathbf{P\Lambda}^2\mathbf{P}. \tag{7}$$

Therefore, the eigenvalues $\mathbf{A}^2$ are simply the square of the eigenvalues of $\mathbf{A}$. This means that each sngular value is given by

$$\sigma_i = \sqrt{\lambda_i^2} = |\lambda_i|, \tag{8}$$

where λ_i is the i-th eigenvalue of $\mathbf{A}$.

Exercise 7. Given that the determinant of a unitary matrix is 1, show that $|\det(\mathbf{A})| = \prod_{i=1}^n \sigma_i$.

Solution 7. Writing the singular value decomposition of $\mathbf{A}$ as $\mathbf{A} = \mathbf{U\Sigma V}^*$, we can see that

$$|\det(\mathbf{A})| = |\det(\mathbf{U})|\,|\det(\mathbf{\Sigma})|\,|\det(\mathbf{V}^*)|\,. \tag{9}$$

Since both $\mathbf{U}$ and $\mathbf{V}$ are unitary, this reduces to

$$|\det(\mathbf{A})| = |\det(\mathbf{\Sigma})|\,. \tag{10}$$

Since $\mathbf{\Sigma}$ is diagonal, its determinant is simply the product of its diagonal entries which are the singular values σ_i. Therefore,

$$|\det(\mathbf{A})| = \prod_{i=1}^{n} \sigma_i. \tag{11}$$

Code used

The code used is entirely in Julia. I find it very similar to Python and Matlab and I generally prefer it. In case you're not familiar with it, I tried to be thorough with my comments throughout. You can find the code as a jupyter notebook on my github in case you want to see this with additional markdown text and in line plots.

```
# Load packages
using LinearAlgebra # Contains functions for SVD and dealing with matrix factorizations
using Plots # General plotting libray
using Images, Colors # Handling Images
using LaTeXStrings # Latex in plots

# Use readdir to get file names
CroppedDataDir = "../data/CroppedYale/"
YaleFolders = readdir(CroppedDataDir)[occursin.("yale", readdir(CroppedDataDir))]

# Loop over folders and read images
img_array = []
for folder in YaleFolders
    FolderDir = CroppedDataDir * folder
    FileNames = readdir(FolderDir)
    for file in FileNames
        if occursin("pgm", file)
            push!(img_array, load(CroppedDataDir * folder * "/" * file))
        end
    end
end

# Save image sizes for conversion to image later
m,n = size(img_array[1])
test_img_vec = vec(img_array[1]);

# Covert greyscale images to column vector
img_array = vec.(img_array);

# Assigning 70% of data to 'train' set for the SVD
train_length = trunc(Int,0.7*length(img_array))

# Combine column vectors of training set to matrix.
#Here, each column corresponds to an observation or individual face.
X_train = hcat(img_array[1:train_length]...);

# Combining remaining columnb vectors to test matrix
X_test = hcat(img_array[(train_length+1):end]...);
```

```
# Compute the SVD of the matrix
U, S, V = F = svd(X_train);

# Helper functions

# Convert column vector to face matrix
function vec_to_face(v, m,n)
    return reshape(v,m,n)
end

# Project face into U space
function project_face(F::SVD, v, r)
    # F.U is the matrix U under the SVD
    new_v = F.U[:,1:r]*(F.U[:, 1:r]' * v)
    return new_v
end

# Recover basis coefficients in U space
function project_alpha(F::SVD, v, r)
    # F.U is the matrix U under the SVD
    new_alpha = F.U[:, 1:r]' * v
    return new_alpha
end

# Rank approximation
function rank_approx(F::SVD, r)
    U, S, V = F
    M = U[:, 1:r] * Diagonal(S[1:r]) * V[:, 1:r]'
end;

# Find corresponding ranks for vector of precent thresholds
function rank_threshold(F::SVD, percent_thres)
    percent_energy = cumsum(F.S)/sum(F.S)
    r_thres = []
    for thres in percent_thres
        push!(r_thres, findfirst( percent_energy .> thres) )
    end
    return r_thres
end

# Plot singular value fall off
scatter(S,
    xlabel = L"r",
    ylabel = L"\sigma_r",
```

```
    label = "Singular values",
    title = "Singular values for cropped faces")
savefig("../figures/hw-2-cropped-singular-values.pdf")

# Visualizing top 20 eigenfaces
r = 20

P = [] # List of plots
for i in 1:r
    Face = vec_to_face(U[:,i], m, n)
    push!(P, heatmap(Face, color = :greys, yflip=true, axis = nothing, legend=false))
end
plot(P[:]..., layout = (5,4), size = (600, 800))
savefig("../figures/hw-2-cropped-eigenfaces.pdf")

r = [5, 10, 50, 100, 200, 400, 1000, 1500] # Vector of ranks used to approximate test f

# Extracting and plotting test face
test_face = 5 # Test index
real_face = vec_to_face(X_test[:, test_face], m, n)
real_plot = heatmap(real_face, color = :greys, yflip=true, axis = nothing, legend=false,

# Approximating this face at various ranks
rankr_approx = []
for rank in r
    # Project face into U space with rank r
    proj = project_face(F, X_test[:, test_face], rank)
    proj_face = vec_to_face(proj, m, n)

    # Adding to approximation to plot
    push!(rankr_approx, heatmap(proj_face, color = :greys, yflip=true, axis = nothing, l
end

plot(real_plot, rankr_approx..., layout = 1 + length(r), size = (800, 1000))
savefig("../figures/hw-2-cropped-rank-approximation.pdf")

# Visualizing weights
rankr_weights = []
for rank in r
    # Project face into U space with rank r
    proj_alpha = convert.(Float64, project_alpha(F, X_test[:, test_face], rank))

    # Adding to approximation to plot
    push!(rankr_weights, bar(proj_alpha, label=false, title="r=$(rank) approximation",xl
end
```

```
plot(rankr_weights..., layout = (4, 2), size = (800, 800))
savefig("../figures/hw-2-cropped-rank-weights.pdf")

# Percent of spectrum described by modes
percent_energy = cumsum(S)/sum(S)

plot(percent_energy,
    xlabel = L"\textrm{Rank } r",
    ylabel = L"\textrm{Percentage of total energy}",
    legend=false);

# We can use the cummulative sum of the energy to put a threshold on the rank r.

percent_thres = [0.50 0.75 0.90 0.95 0.99]

# Finds the minimal rank r for which total energy is less than threshold
r_thres = []
for thres in percent_thres
    push!(r_thres, findfirst( percent_energy .> thres) )
end

# Plotting location on previous plot
plot(percent_energy,
    linewidth = 2,
    xlabel = L"\textrm{Rank } r",
    ylabel = L"\textrm{Percentage of total energy}",
    label = false)

for k in eachindex(r_thres)
    vline!( [r_thres[k]],
        label="$(Int(percent_thres[k]*100))%",
        style=:dash)
end

plot!()
savefig("../figures/hw-2-cropped-percent-energy-thresholds.pdf")

# Approximating this face at threshold ranks
rankr_approx = []
for k in eachindex(r_thres)
    # Project face into U space with rank r
    proj = project_face(F, X_test[:, test_face], r_thres[k])
    proj_face = vec_to_face(proj, m, n)
```

```
    # Adding to approximation to plot
    push!(rankr_approx, heatmap(proj_face,
                                color = :greys,
                                yflip=true,
                                axis = nothing,
                                legend=false,
                                title="$(Int(percent_thres[k]*100))% energy threshold"))
end

plot(real_plot, rankr_approx..., layout = 1 + length(r_thres), size = (800, 700))
savefig("../figures/hw-2-cropped-percent-energy-approximation.pdf")

# Use readdir to get file names
UncroppedDataDir = "../data/yalefaces_uncropped/"
YaleFolders = readdir(UncroppedDataDir)[occursin.("yale", readdir(UncroppedDataDir))]

# Loop over folders and read images
u_img_array = []
for folder in YaleFolders
    FolderDir = UncroppedDataDir * folder
    FileNames = readdir(FolderDir)
    for file in FileNames
        if occursin("subject", file)
            push!(u_img_array, Gray.(load(UncroppedDataDir * folder * "/" * file)))
        end
    end
end

# Save image sizes for conversion to image later
m,n = size(u_img_array[1])

# Covert greyscale images to column vector
u_img_array = vec.(u_img_array);

# Assigning 80% of data to 'train' set for the SVD
train_length = trunc(Int,0.8*length(u_img_array))

# Combine column vectors of training set to matrix.
#Here, each column corresponds to an observation or individual face.
X_train = hcat(u_img_array[1:train_length]...);

# Combining remaining columnb vectors to test matrix
X_test = hcat(u_img_array[(train_length+1):end]...);
```

```
# Compute the SVD of the matrix
U, S, V = F = svd(X_train);

# Visualizing top 20 eigenfaces
r = 20

P = [] # List of plots
for i in 1:r
    Face = vec_to_face(U[:,i], m, n)
    push!(P, heatmap(Face, color = :greys, yflip=true, axis = nothing, legend=false))
end
plot(P[:]..., layout = (5,4), size = (800, 1000))
savefig("../figures/hw-2-uncropped-eigenfaces.pdf")

# Plot singular value fall off
scatter(S,
    xlabel = L"r",
    ylabel = L"\sigma_r",
    label = "Singular values",
    title = "Singular values for uncropped faces")
savefig("../figures/hw-2-uncropped-singular-values.pdf")

r = [5, 10, 50, 100, length(S)] # Vector of ranks used to approximate test face

# Extracting and plotting test face
test_face = 5 # Test index
real_face = vec_to_face(X_test[:, test_face], m, n)
real_plot = heatmap(real_face, color = :greys, yflip=true, axis = nothing, legend=false,

# Approximating this face at various ranks
rankr_approx = []
for rank in r
    # Project face into U space with rank r
    proj = project_face(F, X_test[:, test_face], rank)
    proj_face = vec_to_face(proj, m, n)

    # Adding to approximation to plot
    push!(rankr_approx, heatmap(proj_face, color = :greys, yflip=true, axis = nothing, l
end

plot(real_plot, rankr_approx..., layout = 1 + length(r), size = (800, 800))
savefig("../figures/hw-2-uncropped-rank-approximation.pdf")

# Percent energy thresholding on uncropped data
percent_energy = cumsum(S)/sum(S)
```

```
percent_thres = [0.50 0.75 0.90 0.95 0.99]

# Finds the minimal rank r for which total energy is less than threshold
r_thres = []
for thres in percent_thres
    push!(r_thres, findfirst( percent_energy .> thres) )
end

# Plotting location on previous plot
plot(percent_energy,
    linewidth = 2,
    xlabel = L"\textrm{Rank } r",
    ylabel = L"\textrm{Percentage of total energy}",
    label = false)

for k in eachindex(r_thres)
    vline!( [r_thres[k]],
        label="$(Int(percent_thres[k]*100))%",
        style=:dash)
end

plot!()
savefig("../figures/hw-2-uncropped-percent-energy-thresholds.pdf")

# Approximating this face at threshold ranks
rankr_approx = []
for k in eachindex(r_thres)
    # Project face into U space with rank r
    proj = project_face(F, X_test[:, test_face], r_thres[k])
    proj_face = vec_to_face(proj, m, n)

    # Adding to approximation to plot
    push!(rankr_approx, heatmap(proj_face,
                                color = :greys,
                                yflip=true,
                                axis = nothing,
                                legend=false,
                                title="$(Int(percent_thres[k]*100))% energy threshold"))
end

plot(real_plot, rankr_approx..., layout = 1 + length(r_thres), size = (800, 700))
savefig("../figures/hw-2-uncropped-percent-energy-approximation.pdf")
```

Figure 3: The first 20 column vectors in $\mathbf{U}$ corresponding the highest singular values σ_j for the cropped faces dataset.

Figure 4: The first 20 column vectors in $\mathbf{U}$ corresponding the highest singular values σ_j for the uncropped faces dataset.

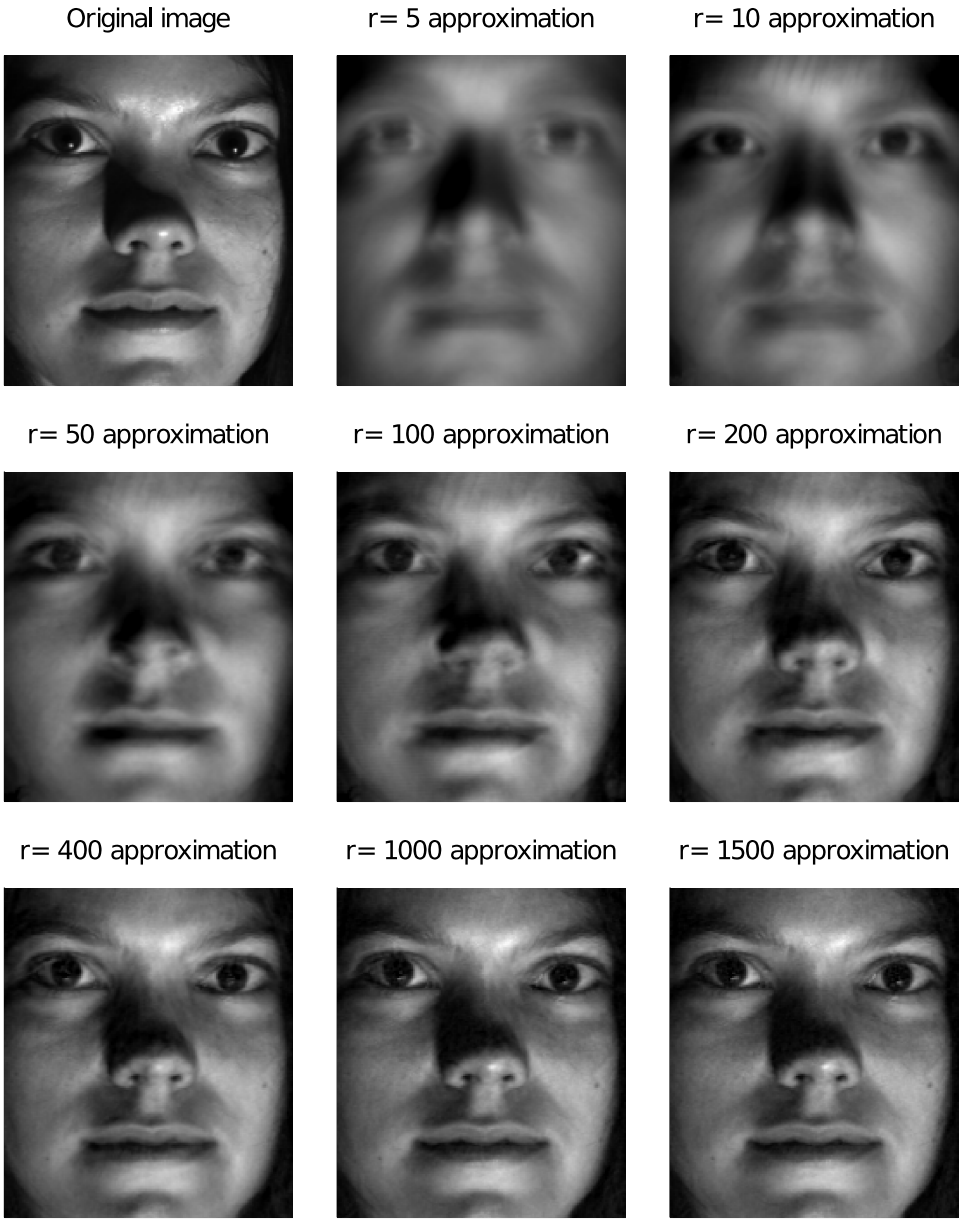

Figure 5: Various rank approximations for a test image on the cropped photos.

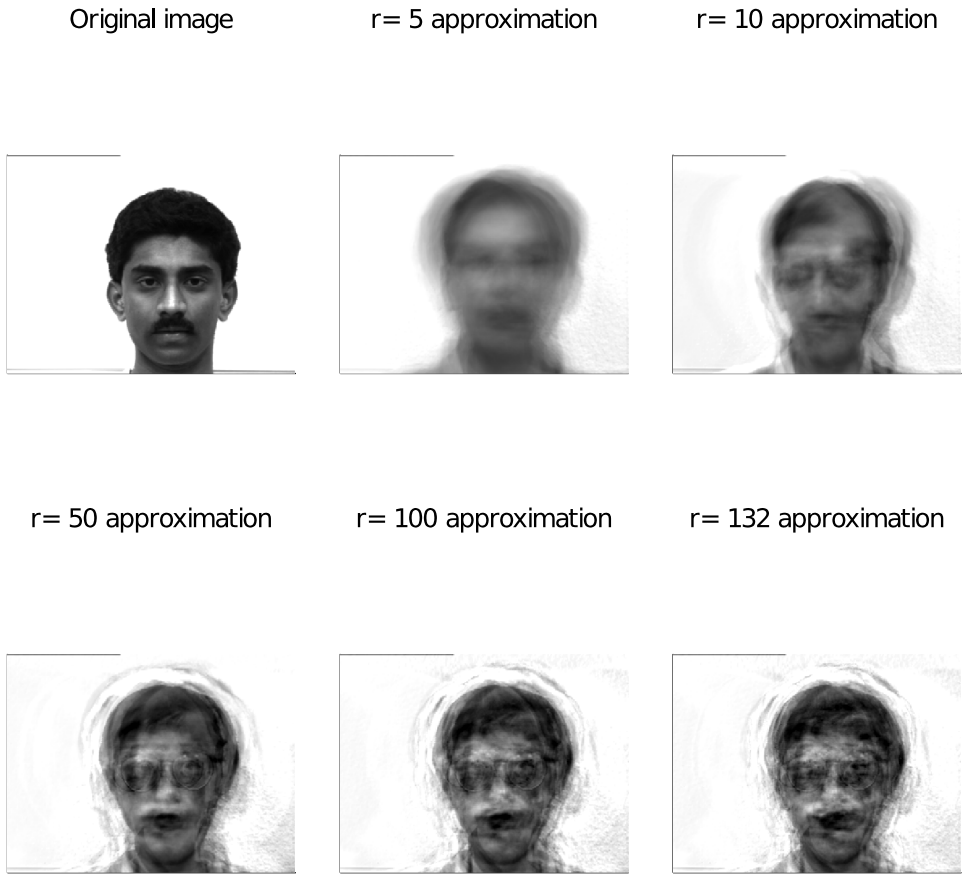

Figure 6: Various rank approximations for a test image on the uncropped photos.

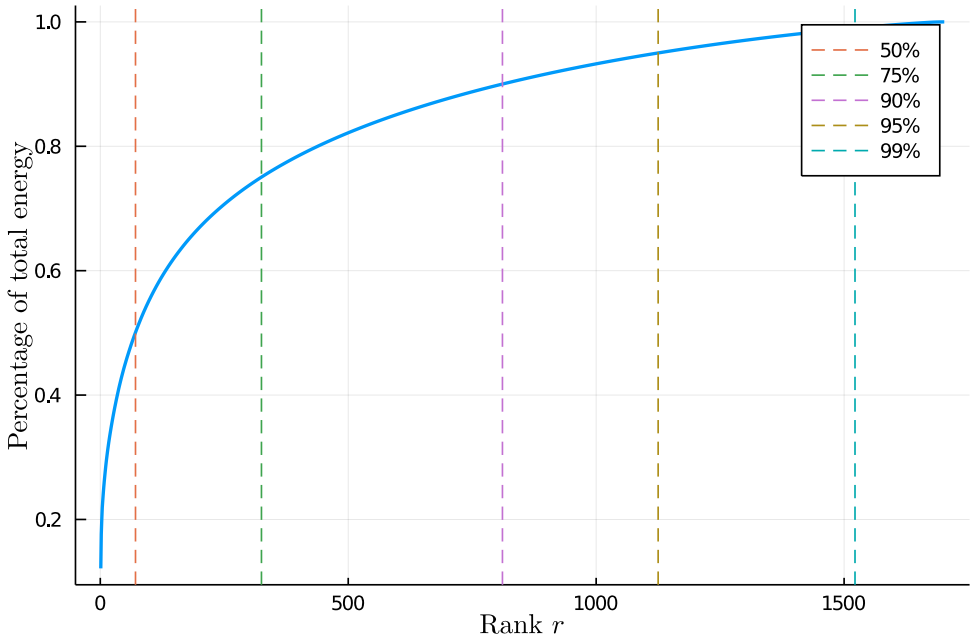

Figure 7: Percent energy threshold for cropped data at various cutoffs λ. Each vertical line is at the rank at which the threshold is met.

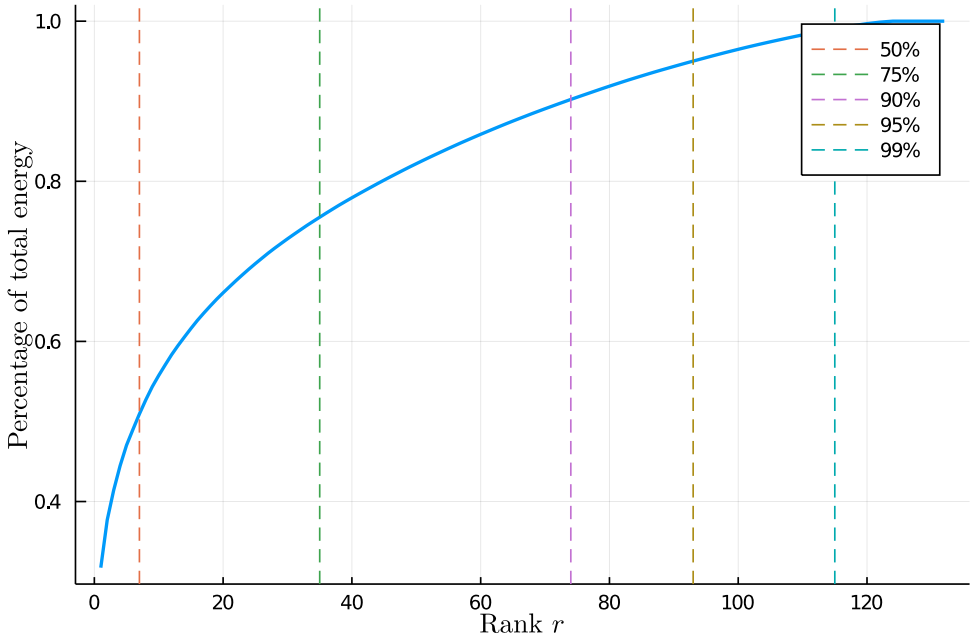

Figure 8: Percent energy threshold for uncropped data at various cutoffs λ. Each vertical line is at the rank at which the threshold is met.

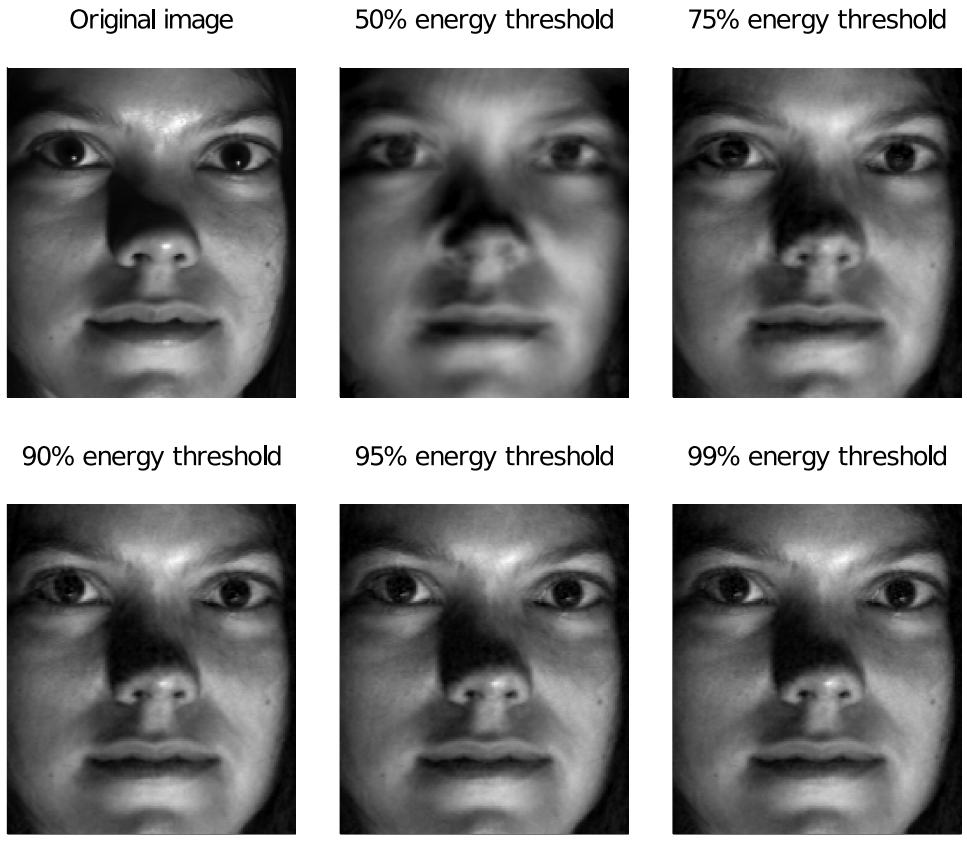

Figure 9: Approximation based on percent energy for cropped data set

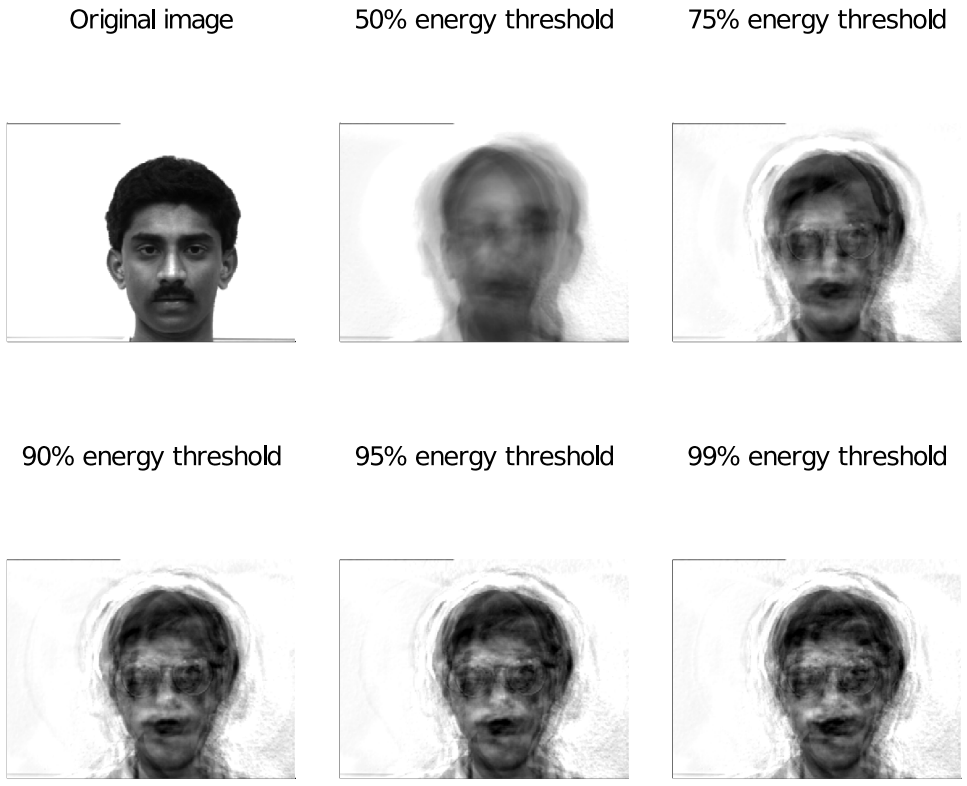

Figure 10: Approximation based on percent energy for uncropped data set