# From Tiles to Text

## Scrabble Board Letter Recognition Using PCA and Edge Detection

Ryan Brooks and Emma Brookins

# Problem Statement

- Scrabble
  - Want to create the greatest number of interconnected words on a 15 by 15 grid board
- Scrabble Solver Applications
  - Assist players with gameplay
- Manual Input Creates Problems
  - Prone to user input mistakes
  - Time consuming



# Motivation/Application

- By utilizing SVD and PCA, we can process images to automatically extract the letters from a Scrabble board with up to 94% accuracy
  - Largely stems from the Eigenface problem
  - Recognize the most important features of each individual letter to create eigen letters
  - Compact representations can be compared to known letters for classification.



# Basic Background - Character Recognition

- 1998: Neil Muller and Ben Herbst extended the eigenface idea to character recognition
  - Found that “the ideas behind the eigenface technique can be easily extended to the OCR problem by using multiple image classes”
  - We won’t be using OCR, but this development was critical to expanding to letter recognition
- 2011: Tanja Van Hecke expanded to using PCA applied to A-Z letter data sets



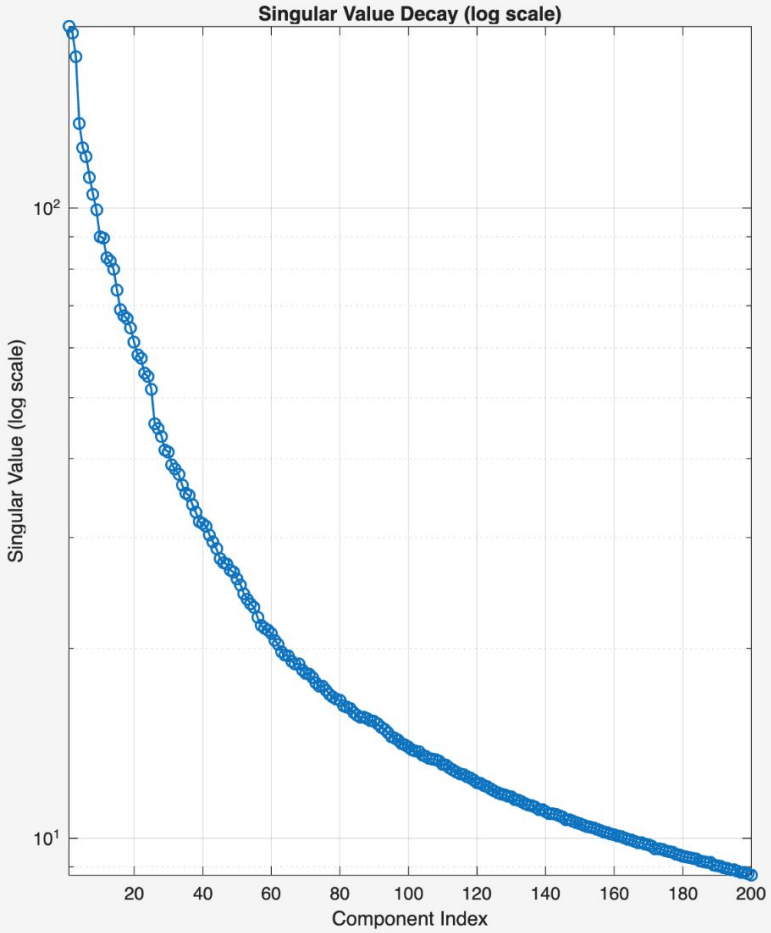
# Terms To Know

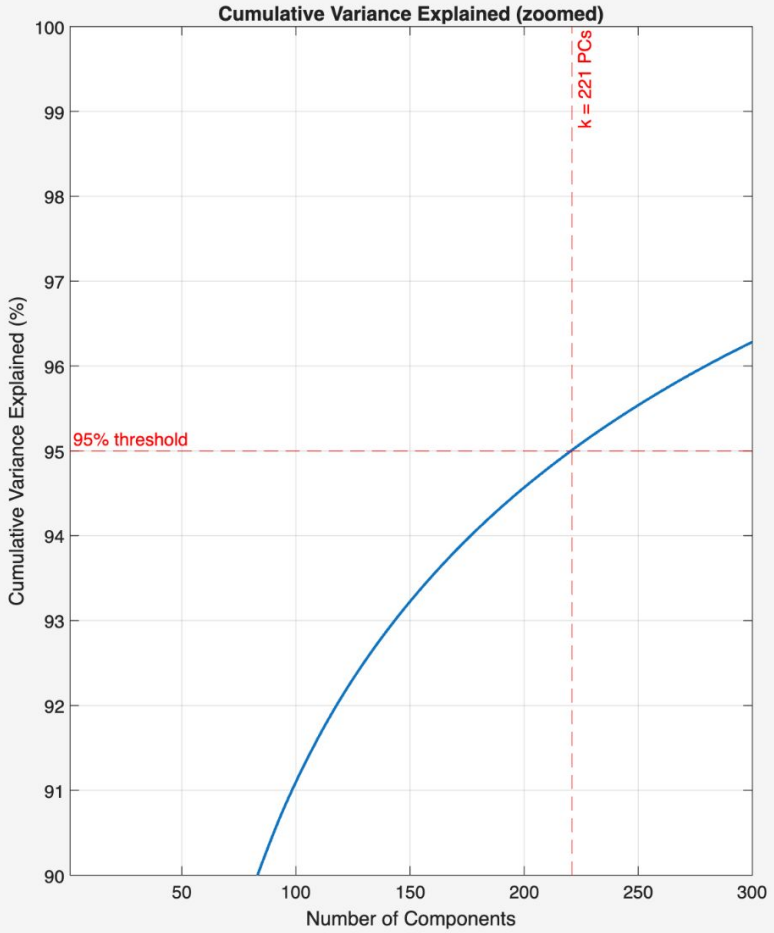
- PCA
- Covariance Matrix
- Eigenface Technique
- Eigenletters
- Image Preprocessing
- Edge Detection
- Letter Classification



# PCA: Principal Component Analysis

- Dimensionality reduction method (reduces large data sets)
- Transforms a data set into a set of orthogonal components
  - Principal Components
  - The first captures as much variance as possible
  - The second captures the most remaining variance and is orthogonal to the first
  - And so on with decreasing  amounts of variance captured
  - Preserves key data trends


Singular Value Decay (log scale)
Singular Value (log scale)
$10^2$
$10^1$
20
40
60
80
100
120
140
160
180
200
Component Index


Cumulative Variance Explained (zoomed)
Cumulative Variance Explained (%)
100
99
98
97
96
95
94
93
92
91
90
95% threshold
k = 221 PCs
50
100
150
200
250
300
Number of Components

# Covariance Matrix

- Square matrix that shows the covariances between multiple variables
  - Each element represents how much two variables vary together
- Joint Variance
- Positive Covariance
  - Both move up and down together (positively correlated)
- Negative Covariance
  - Move in opposite directions (negatively correlated)
- $$C = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\left(X_i - \bar{X}\right)^T$$

  $X_i$ is the i-th data point

  $\overline{X}$ is the mean vector of the dataset.



# Proof: PCA and SVD Yield the Same Principal Components

We find the eigenvectors of the covariance matrix using PCA

But, we compute the eigenletters using the svd() function in MATLAB...

To prove that our eigenletters are valid, we need to prove that SVD and PCA give us the same principal components.

# Proof: PCA and SVD Yield the Same Principal Components

SVD: $A = U\Sigma V^T$

- Columns of U are orthonormal ($U^TU=I$)
- Rows and columns of V are orthonormal ($V^TV=I$, $VV^T=I$)

Relationship to PCA:

- Let X = Design Matrix
  - $X=[X_1, X_2, X_3, ..., X_M]$
  - Each X represents a 1D vector of pixel intensities of each letter

# Proof: PCA and SVD Yield the Same Principal Components

$X = U\Sigma V^T$

$X^TX = V\Sigma^TU^TU\Sigma V^T$

- $U^TU = I$ because the columns of U are orthonormal

$X^TX = V\Sigma^T\Sigma V^T$

Let $D = \Sigma^T\Sigma$

- D is a diagonal matrix with the squares of the singular values

$X^TX = VDV^T$

If we multiply each side by V, we have

$(X^TX)V = VDV^TV$

$V^TV=I$, so we get

$(X^TX)V = VD$

- V is a matrix composed of the eigenvectors of $X^TX$
- D is the corresponding eigenvalues

Let C = the sample covariance matrix

$C= {}^1/_n\ (X^TX)$

So, $X^TX$ is just a scaled version of the sample covariance matrix.

# Proof: PCA and SVD Yield the Same Principal Components

Relationship to PCA:

To get the projection vectors, we calculate the eigenvalue eigenvector pairs for the sample covariance matrix

As we have just shown, we can get all of the eigenvector and eigenvalue pairs by doing the SVD of the sample covariance matrix

Therefore, PCA and SVD will give us the same principal components



# Eigenletters

- Represent the principal components of a dataset containing images of letters
- Patterns describe variations among the letter images
- All letter images composed of combinations of different eigenletters
- Ex.
  - Curved vs straight lines in letters, O vs K
  - Horizontal lines vs straight lines, E vs I



# Basic Approach

**Step 1:** Get a scrabble board image. Normalize, convert to grayscale, and further preprocessing steps.

**Step 2:** Divide scrabble board image into individual letter tiles (Edge Detection)

**Step 3:** Run PCA algorithm on our dataset of letters and use a classification algorithm on detected board tiles.

This eigenletter approach intends to show how eigendecomposition can be used for pattern recognition



# Image Preprocessing

Goal: Exaggerate features of the image to differentiate the playable board from the background.

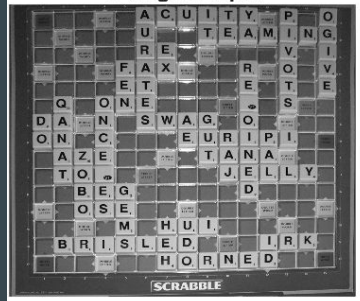
- Converts to grayscale
- Use of Gaussian blur to reduce overall noise
- Boost contrast using CLAHE
  - Smooths out histogram of pixel brightness intensities to cover the full distribution
- Median Filter to target “specks” while preserving edges
- Reduce glare and smooth boundaries with Morphological Opening
  - Slides a disk structuring element. Features smaller than the disk will be removed (erosion) and larger ones will be kept and restored using dilation.

# Image Preprocessing



# Image Preprocessing

1. Original Input



2. After Gaussian (σ=1.5)



3. After Contrast Boost



4. Gaussian (σ=1.5) Again



5. Median Filter [5x5]



6. Morphological Opening



7. Final for Edge Detection



# Grid Detection

Before applying PCA, we need to break up our board into individual letter tiles

Put into action in our MATLAB example

Step 1:

- Canny edge detector to compute gradients along edges where brightness changes rapidly and chooses its maxima.
- Threshold range determines if it's an edge
- Follows all connected pixels within or above that range and deletes the rest.
- Results in a binary image matrix
    - White = foreground, edge pixels and black = background, everything else

# Grid Detection

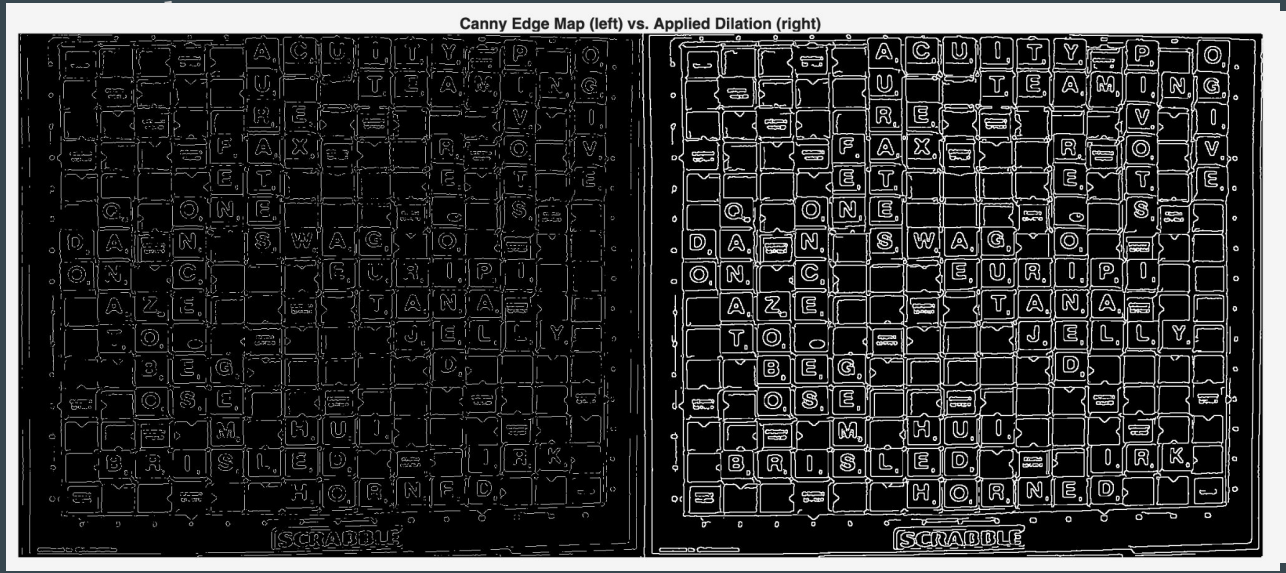
Step 2:

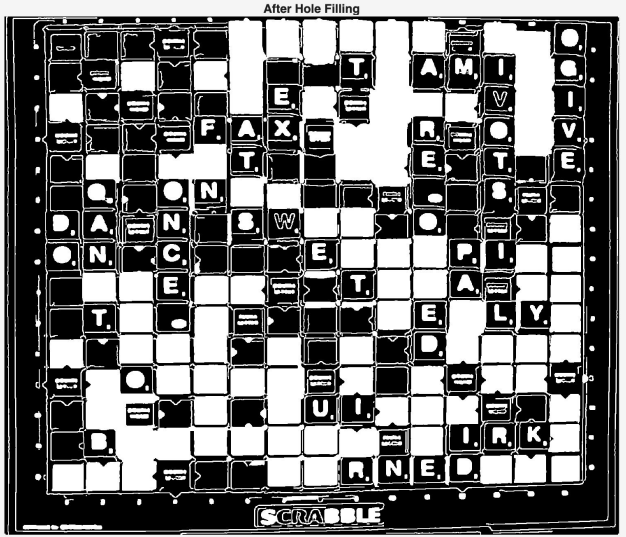
- Dilate edges using morphology
  - if a white pixel falls with the structuring element, it gets dilated to 3x3

Step 3:

- Fill-in holes that are background pixels surrounded by foreground ones with white pixels
  - Noisy areas won’t dominate the image with lots of contours

# Grid Detection



# Grid Detection

Step 4:

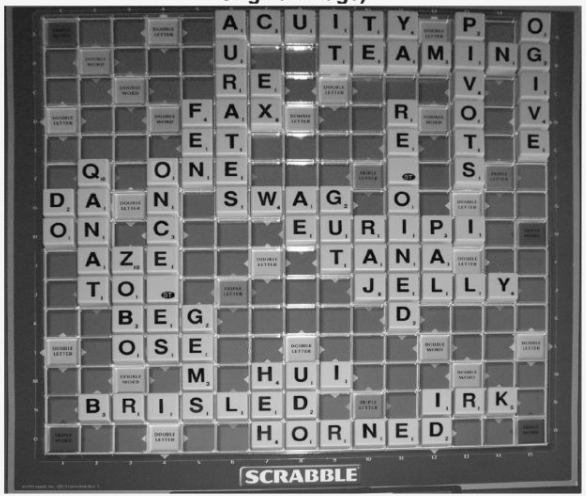
- Label each connected region in the matrix like 1, 2, ...
- Choose the largest based on number of connected pixels

Step 5:

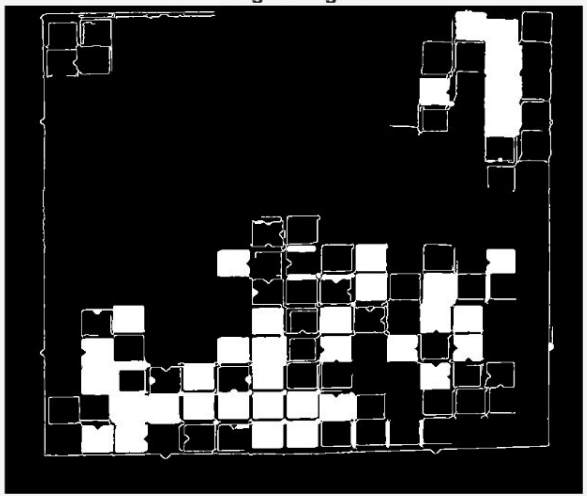
- Compute a convex hull of the largest region
  - Applies  neighbor tracking algorithm to keep only pixels that border the background
- Minimize to only the vertices that form a smooth outer-most set

# Grid Detection

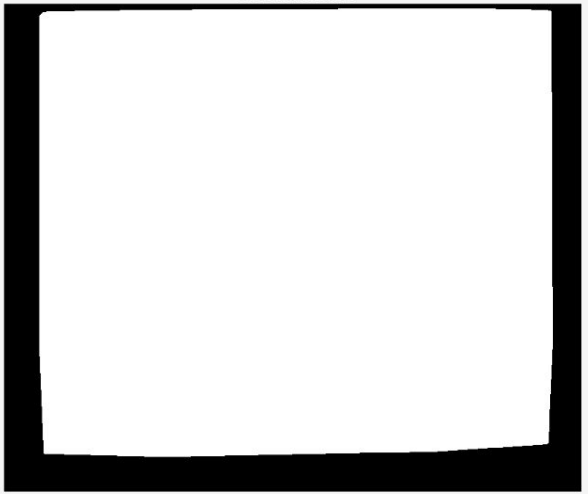
Original Image)




Largest Region



Convex Hull



# Grid Detection

Step 6:

- Find corner points
- PCA
  - Center data
  - Compute covariance matrix
  - Eigendecomposition gives $CV = VD$
  - Principal components given the major and minor axis of the board
  - Project hull points onto new space and take the ones with the max dot product along these axis

# Grid Detection

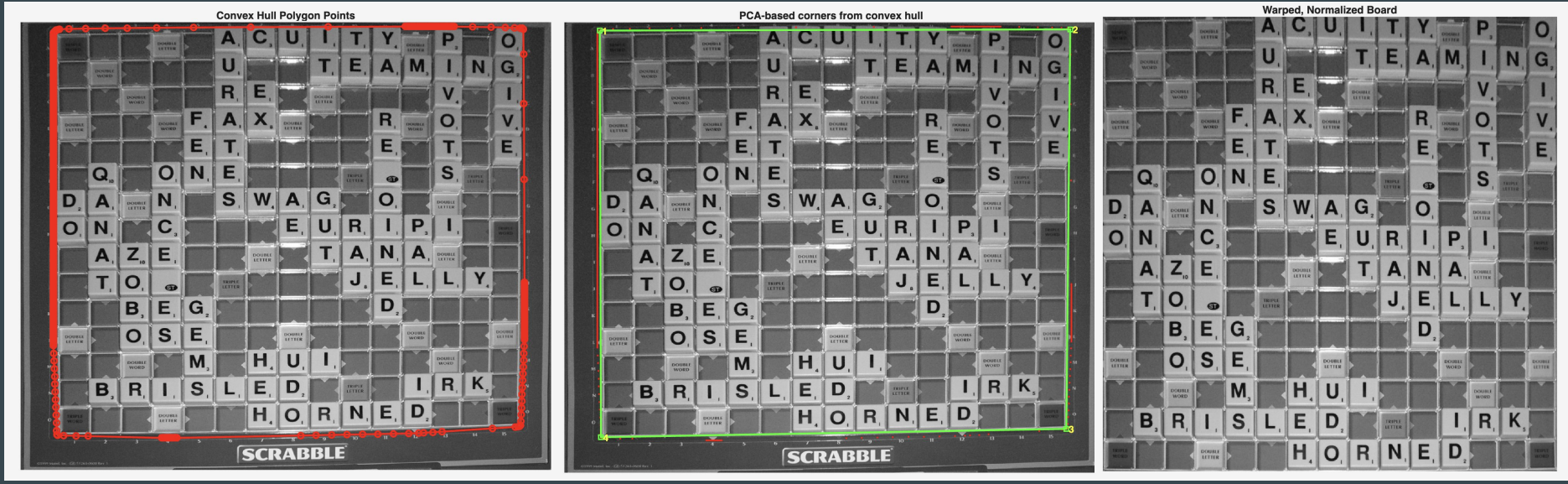
Step 7:

- Take the 4 corners, apply  projective transform
- Solves  homography matrix to map coordinates
- End up with a system Ah = 0 and solve 8 equations with 8 unknowns in h (SVD)
- Take the right singular vectors last column gives h, corresponding to the smallest singular value.
- Reshape h into a 3x3 H and apply the transformation

Step 8:

- Divide the image into an even 15x15 grid.
- Extract letters and vectorize them

# Grid Detection

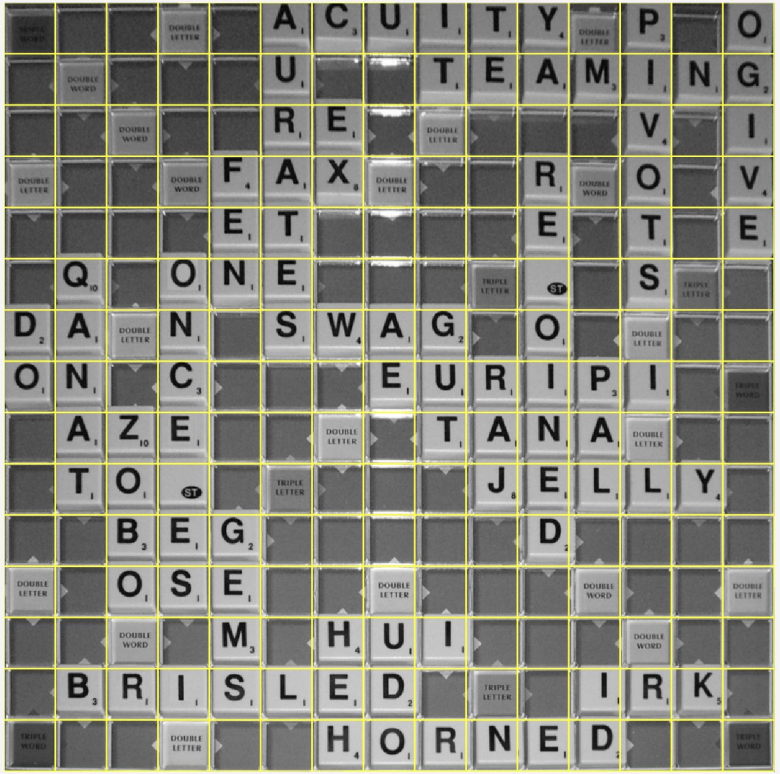


# Grid Detection

Original Board




Processed Board with 15×15 Grid



# Recap

~~**Step 1:** Get a scrabble board image. Normalize, convert to grayscale, and further preprocessing steps.~~

~~**Step 2:** Divide scrabble board image into individual letter tiles (Edge Detection)~~

**Step 3:** Run PCA algorithm on our dataset of letters and use a classification algorithm on detected board tiles.

We now have individual letter tiles from the scrabble board ready to be identified. We do this through the PCA method.

# Letter Processing

Tiles are extracted but still varies widely due to lighting, shadows, color differences, glare, and the fact that Scrabble tiles are 3-D objects.

Before PCA can be applied, each extracted tile must be standardized.

Process them just like we did the board image.

1. Boost contrast
2. Set a threshold and convert to black background and white letters
3. Center the letter (the largest central contour)
4. Attempt to filter out blank tiles and other non letters

| Raw | Gray | Contrast | Thresh | Final |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Letter Processing

Original



After Filter



Normalized

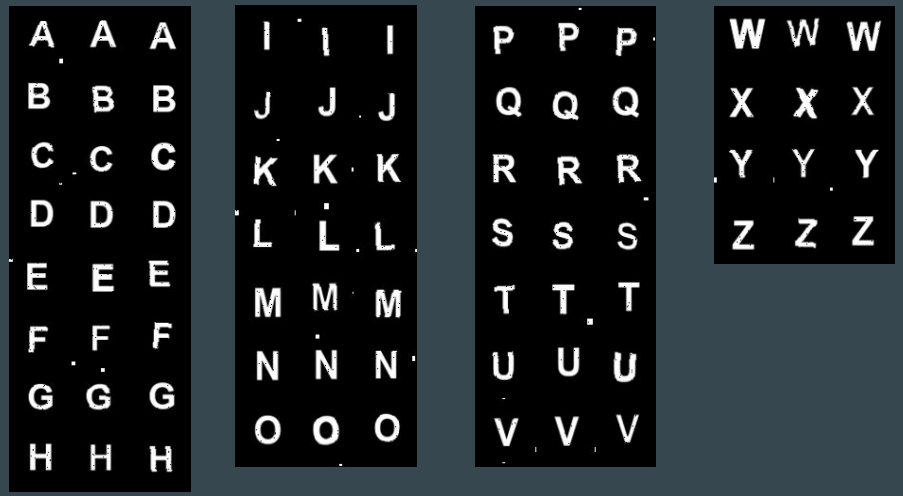


After Filter 2



# PCA and Eigenletters Method

1. Gather training images.
    a. Multiple images per letter, slight variations in noise, rotation, grain, etc
    b. Generated data using python
    c. Tuned sample size and parameters based on classification results



# PCA and Eigenletters Method

2. Vectorize each letter

    Collect all vectors into an PxN matrix X

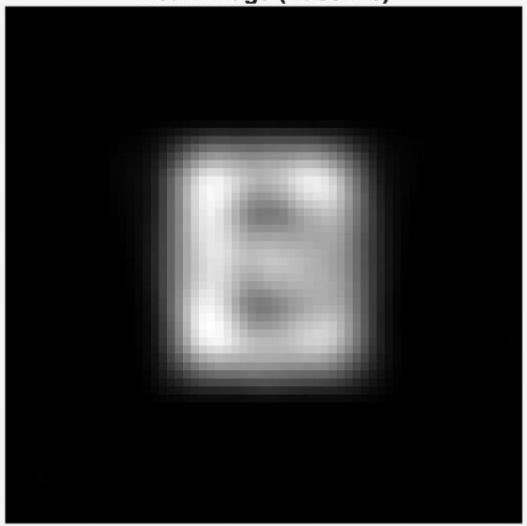
    1. P = size of the flattened letter
    2. N = 26 * k columns
    3. k = # of samples for each letter

3. Compute the average letter image and subtract it from each training vector

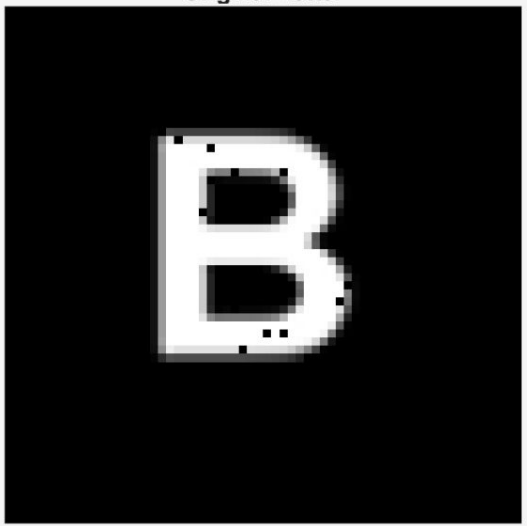
    a. $X_C = X - \overline{x}$
    b. Centers the data because PCA works on the variance from the mean

# PCA and Eigenletters Method

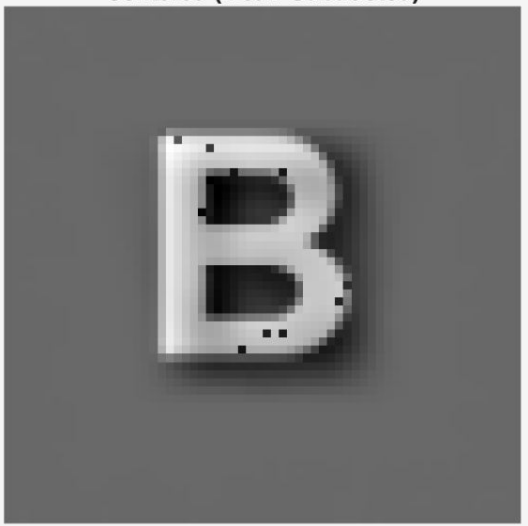
Mean Image (Baseline)



Original Letter



Centered (Mean-Subtracted)



# PCA and Eigenletters Method

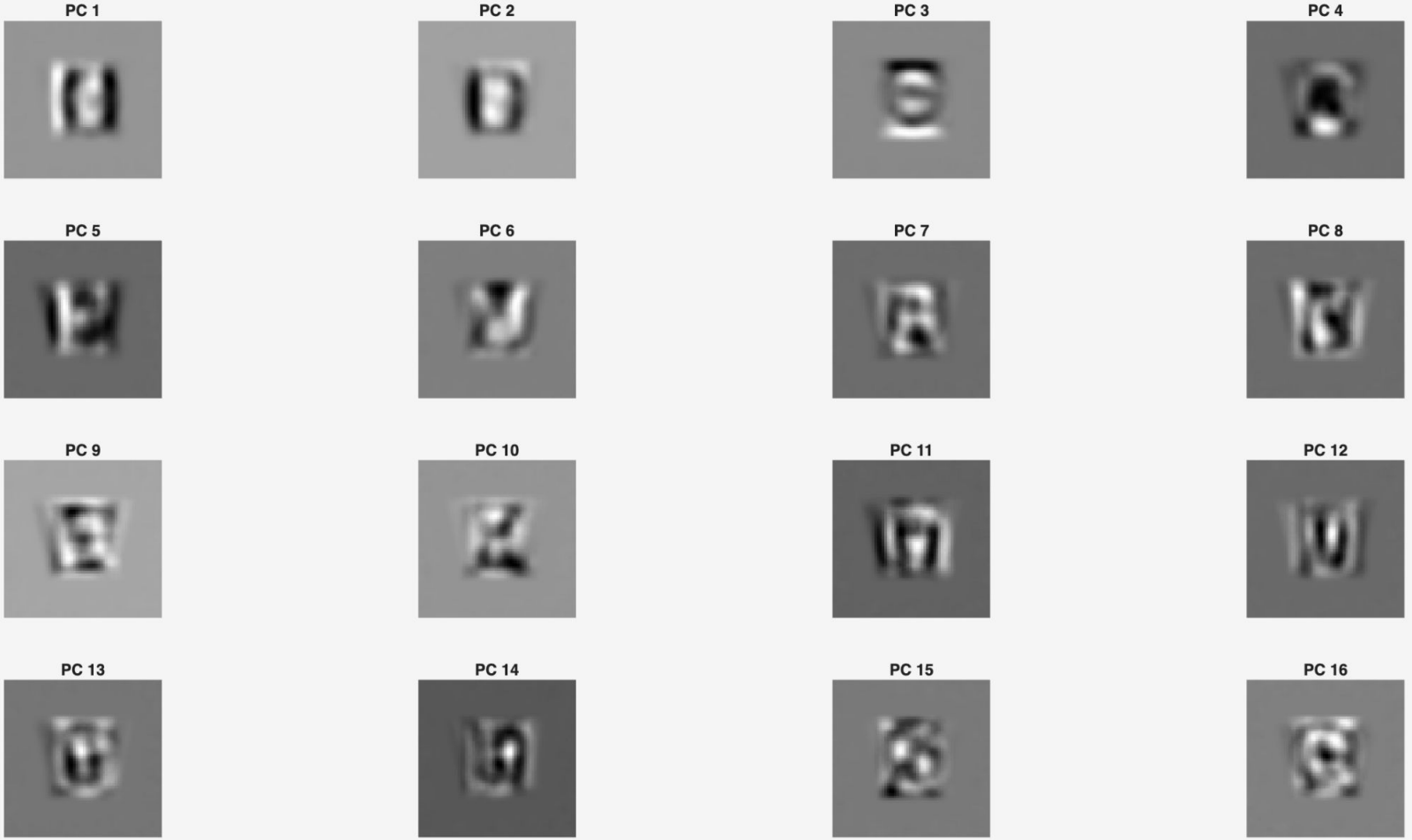
4. Compute eigenletters

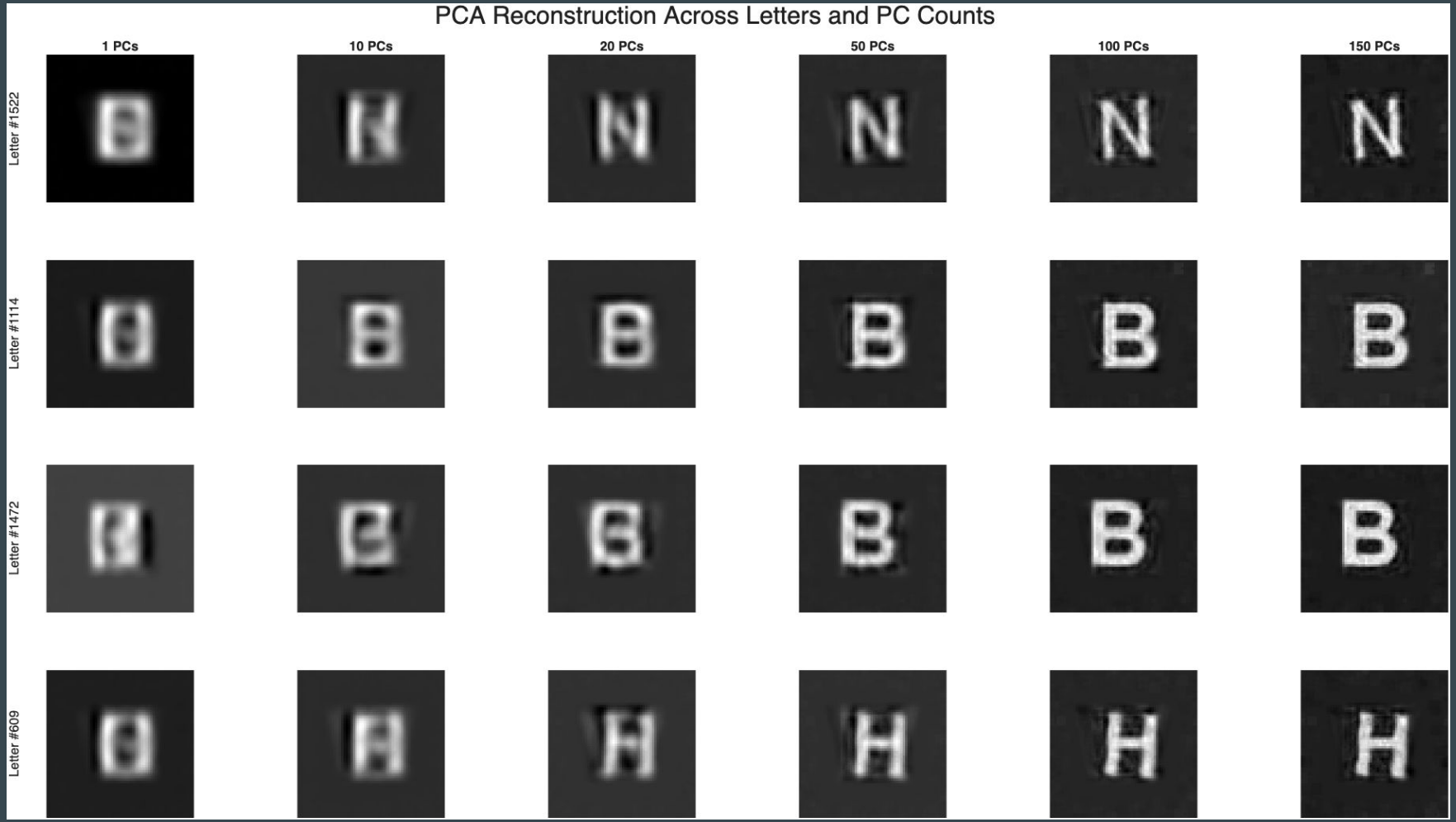
a. Find eigenvectors of the covariance matrix using SVD
b. $X_C = U\Sigma V^T$

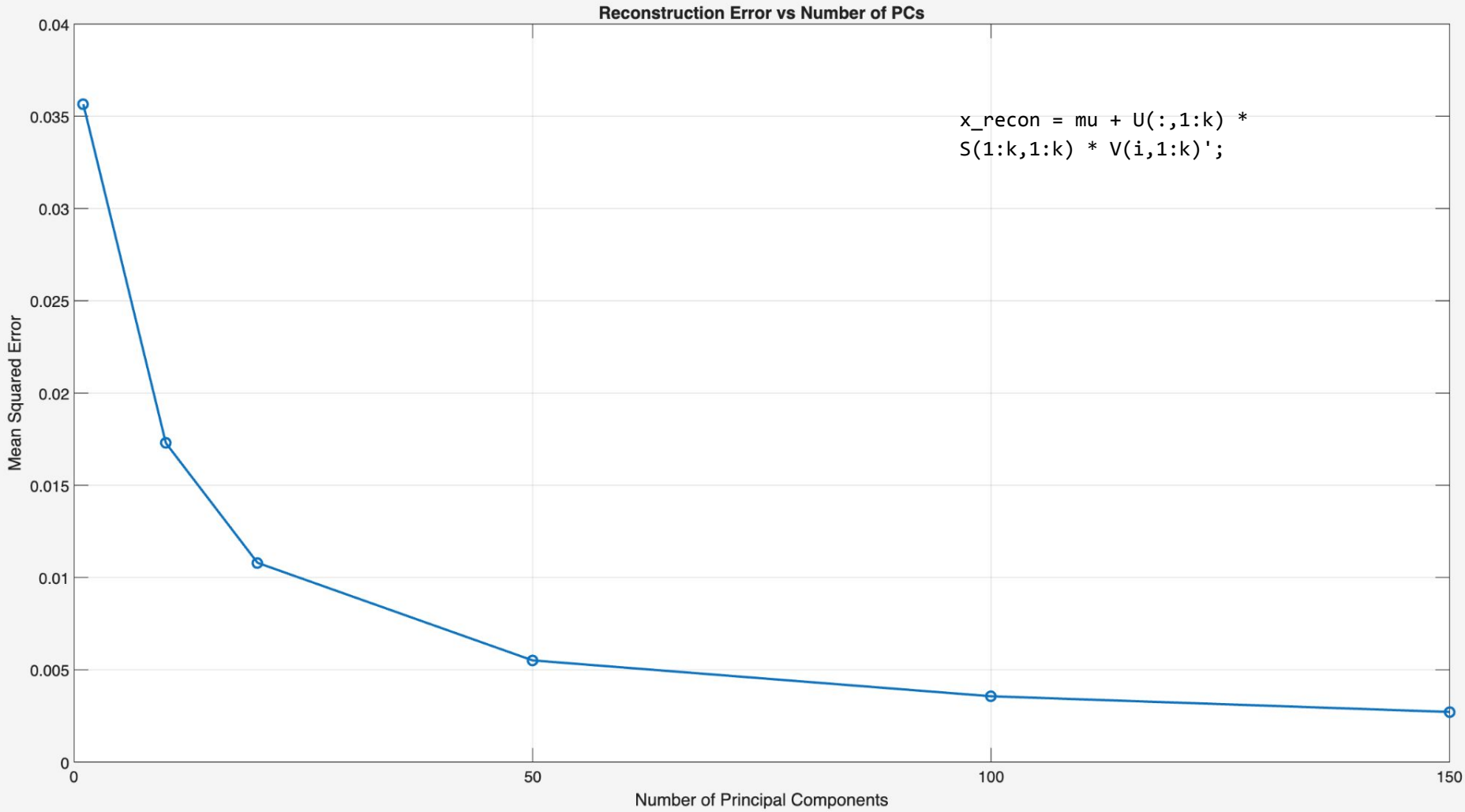
5. Each eigenvector is of length P and needs to be reshaped into its image to visualize it.

a. Top eigenletters correspond to the largest eigenvalues, capturing the most variance across letters.

First 16 Principal Components (Eigenletters)




PCA Reconstruction Across Letters and PC Counts
1 PCs
10 PCs
20 PCs
50 PCs
100 PCs
150 PCs
Letter #1522
Letter #1114
Letter #1472
Letter #609


Reconstruction Error vs Number of PCs
x_recon = mu + U(:,1:k) *
S(1:k,1:k) * V(i,1:k)';
Mean Squared Error
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
50
100
150
Number of Principal Components

# PCA and Eigenletters Method

6. Dimensionality reduction

a. d= the number of principal components we are keeping
b. Eigenletters $U_d = [u_1, ..., u_d]$ each is a d-dimensional vector of PCA coefficients $Z=[w_1, ..., w_d]$
c. Each $w_i$ is the projection of an image onto an eigenletter $u_i$ Represented by $w_i=U_i^T(X - \overline{x})$
    i. $\overline{x}$ = mean vector
    ii. Z = this letter in “letter space”
d. Do this for training and test set of letters to form $Z_{train}$, $Z_{test}$, matrices of letters in PCA space

# PCA and Eigenletters Method

7. To classify an unknown letter from the board, we project it into the same PCA space formed from the training data. Similar letters will be clustered together.

   a. Compare this to the feature vectors of known letters using k-NN clustering in the PCA space, use the Euclidean distance from $z_{test}$ and each known letter's feature vector $z_i$.
      i. Choose k based on odd values and 5-fold cross validation. k (1, 3, 5, ...) that gives the lowest classification error.
   b. Then for each z in $Z_{test.}$ the majority label among its nearest k neighbors is assigned letter class.

# Recap

1. Got the board tiles (testing data)
2. Normalized them
3. Generated training data
4. Performed PCA on the training data
5. Projected each board tile into PCA space
6. Assigned its predicted label using k-NN i.e. what letter it thinks it is

Processed Tiles (Board Layout)




Predicted Letters (Same Scrabble Layout)

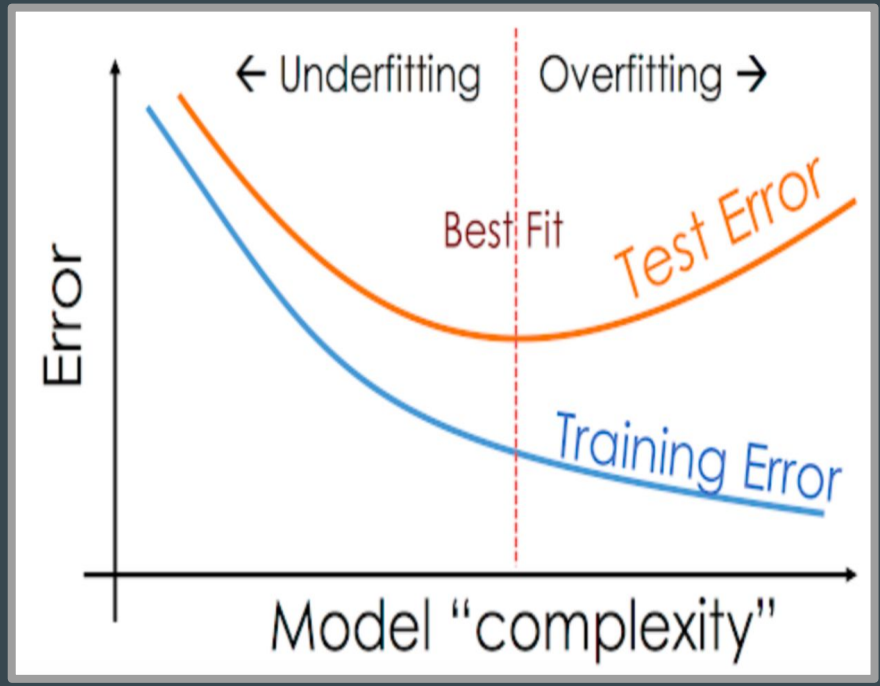


# Results

- With 2600 training samples: 80% training / 20% testing split
  - PCA dimensionality
  - $d$=60, k between 3-7
  - Typical accuracy: 90%–94%
- Misclassifications mainly include:
  - Q to O
  - P to R
  - E to F
  - S to B
- On real data, accuracy is lower. Room for improvement.
  - Filtering still misses some tiles and “DOUBLE LETTERS” that we do not account for in classification.
  - A couple letters need to be further tuned in the dataset to reduce overfitting.

# Possible Issues: Limited Training Data

- Each letter only shows up a few times on the scrabble board
- Issues arise (rotated, different lighting,different fonts, worn down tiles, etc)
- Differences can take over the principal components
    - Similar letters such as O and Q might start to be misclassified
- Can lead to overfitting



# Possible Issues: Board Detection

- Our image processing is extensive but does not generalize well to all situations.
- The type of board and lightning conditions can make contrasting features hard to extract (such as a board's bounding box versus playable area).
- If not all four corners of the board OR enough edges are in the picture, detection will not capture the entire board.

# Sources


1. The use of eigenpictures for optical character recognition
2. Teaching PCA through Letter Recognition
3. Scrabble Referee, Using Real Time OCR
4. Eigen Values  Matrix for Recognition English Letters
5. Scrabble Board Automatic Detector for Third Party Applications
6. PRINCIPAL COMPONENT ANALYSIS IN IMAGE PROCESSING
7. AUTOMATIC RECTIFICATION OF PERSPECTIVE DISTORTION FROM A SINGLE IMAGE USING PLANE HOMOGRAPHY
8. Matlab docs for image processing
9. https://en.wikipedia.org/wiki/Gaussian_blur
10. https://www.mathworks.com/help/images/apply-gaussian-smoothing-filters-to-images
11. https://en.wikipedia.org/wiki/Median_filter
12. https://www.mathworks.com/help/images/ref/medfilt2
13. https://en.wikipedia.org/wiki/Opening_%28morphology%29
14. https://www.mathworks.com/help/images/contrast-enhancement-techniques.html
15. https://en.wikipedia.org/wiki/Canny_edge_detector
16. https://www.mathworks.com/help/images/ref/edge
17. https://www.mathworks.com/help/images/edge-detection
18. https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html
19. https://www.mathworks.com/help/images/ref/bwlabel.html, https://www.mathworks.com/help/images/ref/regionprops.html, https://www.mathworks.com/help/images/ref/bwconvhull.html
20. https://www.mathworks.com/help/images/ref/bwboundaries.html
21. https://www.mathworks.com/help/matlab/ref/convhull.html
22. https://en.wikipedia.org/wiki/Quickhull
23. https://builtin.com/data-science/step-step-explanation-principal-component-analysis#:~:text=Principal%20component%20analysis%20(PCA)%20is%20a%20dimensionality,sets%20while%20preserving%20their%20most%20important%20structures.
24. https://statisticsbyjim.com/glossary/covariance-matrix/#:~:text=By%20Jim%20Frost,to%20rise%20and%20fall%20together.
25. https://epubs.siam.org/doi/pdf/10.1137/S0036144501387517
26. https://www.youtube.com/watch?v=DQ_BkPHIl-g