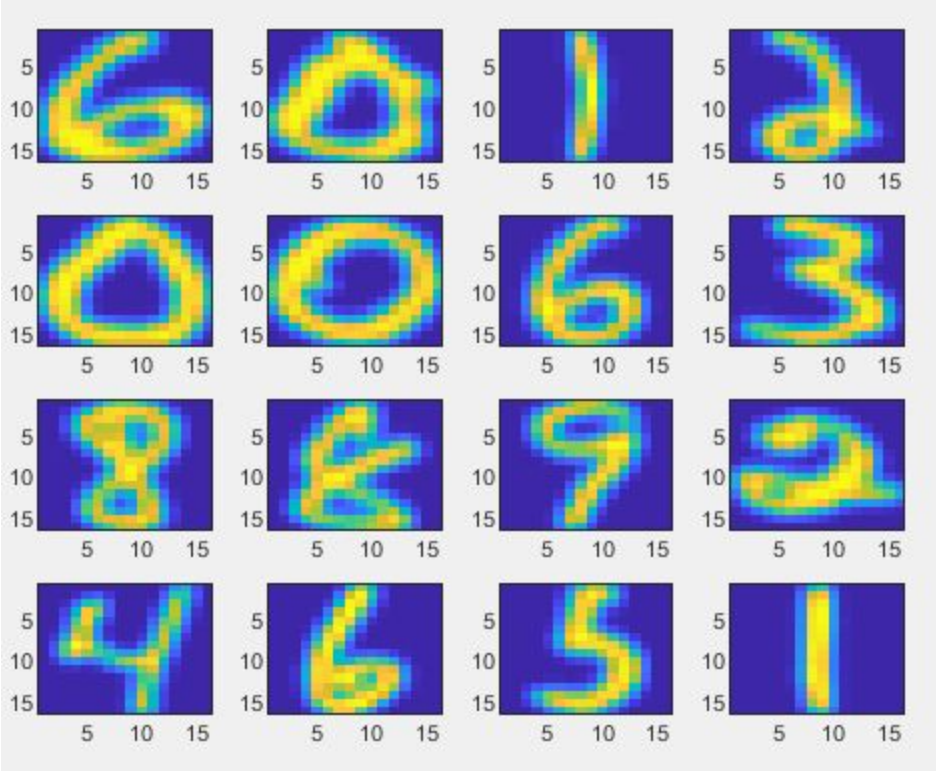
(a)

The images of the digits are stored in matrices. Each matrix, train_patterns and test_patterns, are 256x4649, representing 256 rows of “pixels” and 4649 columns of digits. The images below represent the first 16 digits in train_patterns. In order to achieve the images below, each digit’s pixels were stored into its own 256x1 matrix, reshaped and transposed to form a 16x16 matrix, with each value representing a pixel. They were then plotted, with the minimum value -1 being represented as deep blue and maximum value 1 being represented as yellow/orange.

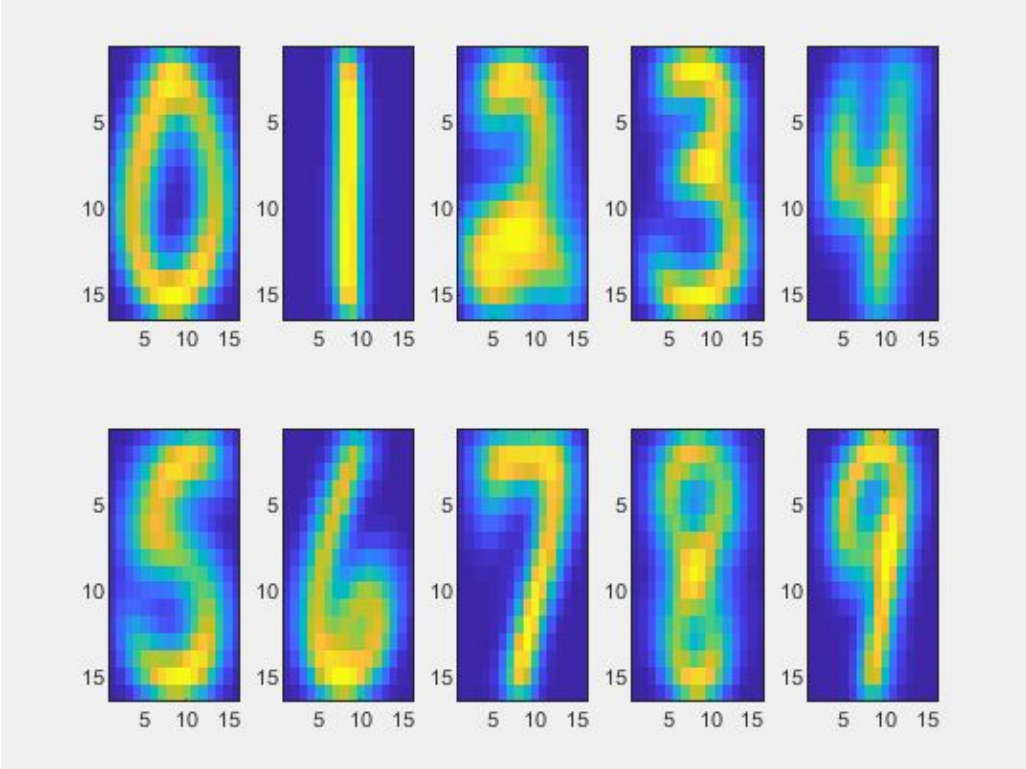


The images above are the first 16 images in train_patterns (part 1b).

Training data and test data are essentially the same. They are just used in different ways. The training data is used to teach a learning algorithm how to identify digits, whereas the test data is used to verify that the algorithm works properly. At the end of algorithm construction and testing, a confusion matrix is created, which compares the results of the algorithm with the actual values of the test data.

(b)

In step 2, I subsetted the training data, train_patterns, based on the label provided in train_labels. From there, I computed the average “pixel” value for each of the ten digits. This will be very important later, because I will use it to predict the test_labels, based on how far off each test image is from the average pixel values. Once I computed the average pixel value, I transposed and reshaped each 256x1 matrix into a 16x16 matrix that could be plotted in a similar fashion to above. The images below represent what the typical digit looks like.



The images above are the typical digits 0-9, based on the mean pixel value of each pool of digits (part 2).

(c), (d)

After predicting all 4649 test digits with both the mean pixel algorithm and the rank 17 SVD algorithm, I created a confusion matrix, which shows how each value was predicted compared to what it actually was, for both algorithms. The confusion matrices are below:

```
>> disp(tmp);
   656     1     3     4    10    19    73     2    17     1
     0   644     0     1     0     0     1     0     1     0
    14     4   362    13    25     5     4     9    18     0
     1     3     4   368     1    17     0     3    14     7
     3    16     6     0   363     1     8     1     5    40
    13     3     3    20    14   271     9     0    16     6
    23    11    13     0     9     3   354     0     1     0
     0     5     1     0     7     1     0   351     3    34
     9    19     5    12     6     6     0     1   253    20
     1    15     0     1    39     2     0    24     3   314
```

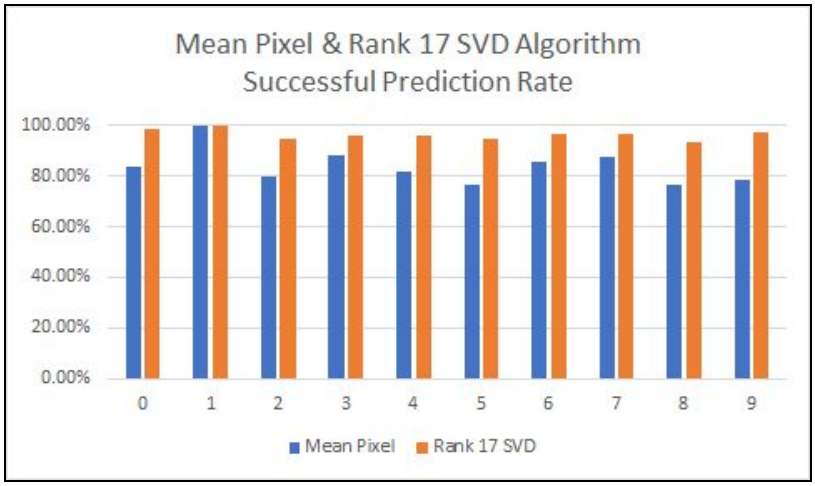
This confusion matrix above corresponds to the mean pixel algorithm (part 3c).

```
>> disp(test_svd17_confusion);
   772     2     1     3     1     1     2     1     3     0
     0   646     0     0     0     0     0     0     0     1
     3     6   431     6     0     3     1     2     2     0
     1     1     4   401     0     7     0     0     4     0
     2     8     1     0   424     1     1     5     0     1
     2     0     0     5     2   335     7     1     1     2
     6     4     0     0     2     3   399     0     0     0
     0     2     0     0     2     0     0   387     0    11
     2     9     1     5     1     1     0     0   309     3
     0     5     0     1     0     0     0     4     1   388
```

This confusion matrix above corresponds to the rank 17 SVD algorithm (part 4d).

While both algorithms are fairly accurate, the rank 17 SVD algorithm is far superior. The mean pixel algorithm correctly predicted the digits 3936/4649 times (84.66%), while the rank 17 SVD algorithm correctly predicted the digits 4492/4649 times (96.62%). A breakdown of how often each digit was correctly predicted is below:

Successful Prediction Rate		
Digit	Mean Pixel	Rank 17 SVD
0	83.46%	98.22%
1	99.54%	99.85%
2	79.74%	94.93%
3	88.04%	95.93%
4	81.94%	95.71%
5	76.34%	94.37%
6	85.51%	96.38%
7	87.31%	96.27%
8	76.44%	93.35%
9	78.70%	97.24%



The table and graph above show how often each algorithm successfully predicted a given test_label for a corresponding vector in test_patterns.

From the data above, it is clear that both algorithms struggled the most with correctly predicting 5s and 8s, although the rank 17 SVD algorithm performed exceptionally superior to the mean pixel algorithm for those digits. It is also clear that neither algorithm struggled trying to predict 1s. This makes sense given how straightforward and unique a simple vertical line for a 1 is compared to digits that twist and turn, such as 5 and 8.

The rank 17 SVD algorithm works by taking advantage of the fact that SVD's $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ is the best rank k approximation of **X** in the sense of least squares. In the algorithm, I pooled together all train_labels equal to *k* and created a matrix that contained each image of that digit *k-1*. From there, I used the svds function to produce $\mathbf{U\Sigma V}^T$, storing the left singular vectors, **U**, in train_u for each *k* = 1:10. Then, by multiplying train_u transposed by test_patterns (the data set that I would like to predict), I come up with the expansion coefficients for each digit basis, test_svd17. In the next step, the matrix svd_approx computes the pixel position of each digit in every 10 digit bases. For each digit, the *k*-th index of the *k*-basis with the minimized least squares error from taking (test_patterns - svd_approx)^2 is stored. This index corresponds to what the predicted digit is, with an index of *k* corresponding to the digit *k-1.* To test the accuracy of the algorithm, a confusion matrix is made, where predicted results stored in svd_res is compared to the actual digits, stored in test_labels.

The rank 17 SVD algorithm is superior to the mean pixel algorithm, because the mean pixel algorithm fails to take into consideration the variation within each class of digits. The SVD, on the other hand, models the variation within each digit class using orthogonal basis vectors.