Code also on Github at:

https://github.com/kkintnermeyer6/AMATH584HW

1 Introduction

Using the MNIST database, we attempted to create a trained model to classify images of written digits. The database consisted of 60,000 training images and 10,000 test images.

2 Full Database Training

To create a trained model, we first have to think about the dimensions of the problem. Each training image was reshaped into a 784x1 column vector, and all of the column vectors were stacked side-by-side to create a matrix, X, which was therefore 784x60,000. Each image had a corresponding label in the form of a 10x1 vector that was all zeros , except for a 1 in the row corresponding to what digit the image contained (for example, a 5 would have a label vector that was $[0,0,0,0,1,0,0,0,0,0]^T$). The digit "0" was captured by the 10th row of the label vector (i.e. a 0 would have a label vector that was $[0,0,0,0,0,0,0,0,0,1]^T$). All of the label vectors were stacked side-by-side as well to create a matrix, B, which was 10x60,000. Thus, using the AX=B framework, we were searching for an A matrix that is 10x784 and gives us the best success rate of classifying the images in the test set. Two main approaches were used. The first approach involved solving for the best least squares fit A matrix. This can be done by solving A = B*pinv(X) where pinv() is the pseudoinverse function. The second approach involves solving for a sparse regression A using the lasso() function, generating a sparser solution for A. Within these two separate solution regimes, we tried a few different heursitics to further characterize the solutions.

2.1 Pseudoinverse approach

The pseduoinverse approach was fairly straightforward. The trained A matrix was calculated by executing A = B*pinv(X) in matlab. The resulting A was then used in the calculation $A * X_{test} = B_{test}$. X_{test} contained the test images reshaped into column vectors, and B_{test} returned what each test image was classified as. The column vectors in B_{test} generally did not have a structure where all entries were zero with one entry equal to one, but instead had mostly small entries, with one positive term dominating the others in magnitude. The row of this entry was taken to be the digit that the test image was classified as. The classifications from the calculation were then compared to the test labels vector to see how many digits were correctly classified and how many were incorrectly classified. Using this barebones approach, the pseudoinverse-calculated A classified 85.34% of the test images correctly.

We now take this time to take a closer look at what the A matrix represents. A is a matrix with 10 rows and 784 columns, and we see that a given column of A represents how a specific pixel in the 28x28 image maps to the 10x1 label-space. Since we rearranged the 28x28 image into a vector, we can also easily tell which pixel a given column of A corresponds to. With this in mind, we looked at the columns of A to see if there was any insights to be gained. Some columns of A had entries that were all order 10^{-17} to order 10^{-15}, while other columns had entries that were all order 10^{-4} to 10^{-2}. Clearly, some of the columns played a much more dominant role in contributing to the label space than others. To further explore this concept, we tried the following heuristic. We found the cartesian norm (2-norm) of each column vector in A. We then specified a cutoff value and zeroed out any columns of A that had a 2-norm that was below the cutoff value. With the zeroing out applied, we then redid the $A * X_{test} = B_{test}$ calculation to see how the classification success rate changed. We found that using a cutoff value of 10^{-3} resulted in a classification success rate of 14.81% but using

a cutoff value of 10^{-4} resulted in a classification success rate of 84.86% with further decrements to the cutoff value order resulting in diminishing returns to the classification success rate. By the time we get to a cutoff value of 10^{-6}, we have a success rate of 85.34% again. We took the norms of each column vector of the original A matrix (no cutoff applied) and reshaped it into a 28x28 image that shows which pixel each norm corresponds to. By plotting the norm values in log10-scale for each pixel, we can infer from the image which pixels play the largest role in classifying digits.

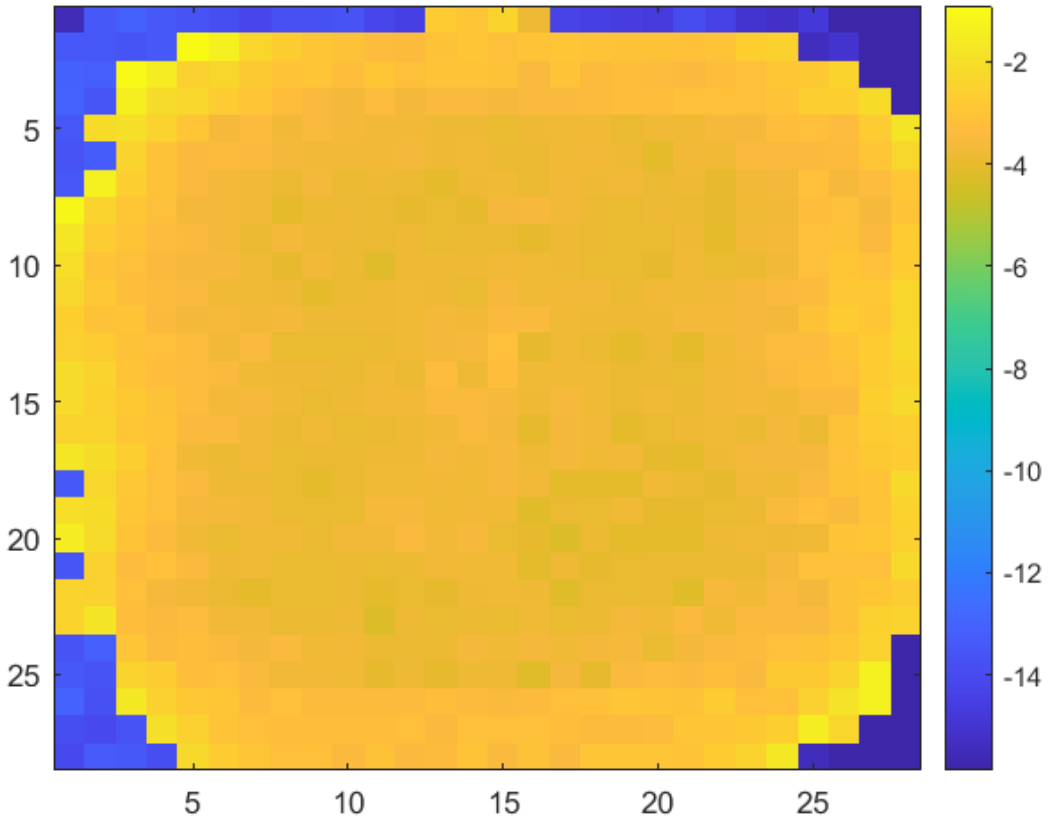

Figure 1: Dominant Pixels for classification - Pseduoinverse method

2.2 Lasso approach

We also used the lasso function in matlab to do a sparse regression on the training dataset. However, to do so, we had to make a few modifications to our approach. The lasso function, as stated by the documentation, solves the regression problem X*w = y, with X and y as inputs into the lasso function, and w as the output. Further, the input, y, into the function must be a column vector. Thus, we cannot put our entire B matrix into the lasso function and expect it to work. Instead, let us reconsider our original AX=B problem, where X is 784x60,000 and contains columns that are reshaped images, and B which is 10x60,000 and contains the training labels. We can rewrite this problem by transposing everything such that $X^T * A^T = B^T$. We now consider this transposed problem by looking at the columns of A^T, which correspond to the rows of A. We can solve the

problem columnwise $X^T * a_j = b_j$ where a_j is a column of A^T and b_j is a column of B^T. Thus, we use the lasso routine to determine each column of A^T and then stack all of the column solutions, a_j, side-to-side, and transpose the stacked a_j matrix to get out our trained A. Lasso also needs a parameter λ to determine how much to penalize the 1-norm of the solution vector. Larger values of λ generate sparser solution vectors. We redid the trial using four different values of λ: [0.01, 0.001, 0.0001, 0.00001]. We looked at how the classification success rate varied with values of λ. When λ is at its max value, 0.01, (the sparsest solution) we have a classification success rate of 78.35%. When λ is at its minimum value, 0.00001, (the least sparse solution) we have a classsification success rate of 81.17%. However, with our sparsest solution, we can get a much better idea of which pixels are playing a dominant role in classifying images of digits. Thus, we took our sparsest solution A and ran the same 2-norm analysis on the column vectors of it like we did for the pseduo-inverse case. This time there was no need to specify a cutoff value as the goal of sparsity is encapsulated within the sparse regression problem already. We again took the norms of each column vector of A and reshaped it into a 28x28 image, plotting the norms on a log10-scale.

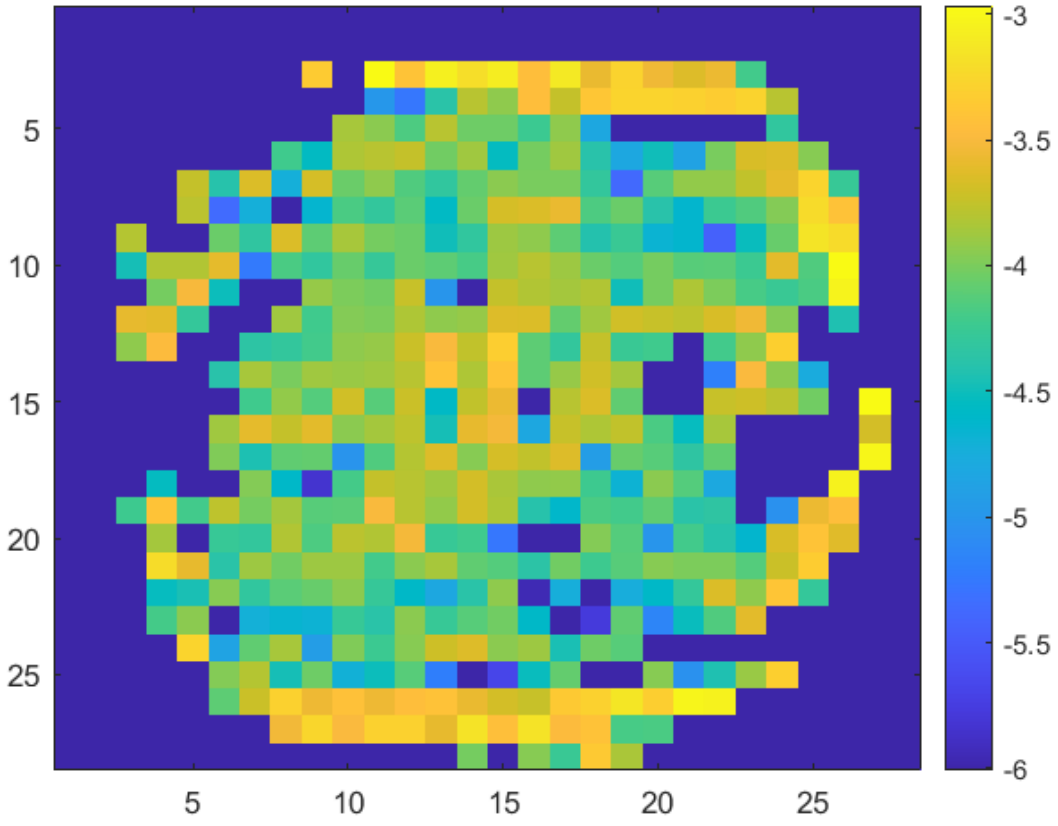

Figure 2: Dominant Pixels for classification - Lasso method

2.3 Discussion

We briefly note here the similarities and differences in Figure 1 and Figure 2. In both cases, the regression analysis correctly throws out the corner pixels as mostly irrelevant, and focuses on

a roughly circular shape of pixels centered around the center of the 28x28 pixel grid. In addition, pixels near the outside edge of the circular shape seem to be particular dominant for both cases. The differences between the two methods are also pretty apparent. In Figure 1, all of the pixels inside the circular shape seem to carry roughly equal weight. This is in line with what we expect, because the least-squares fit tends to try to give roughly equal weightings to all the components. In contrast, we see that in Figure 2, the sparse regression is far pickier in assigning weights to individual pixels. Some pixels inside the circular shape are designated as equally irrelevant to the corner pixels, and other pixels are designated as much less important to the regression than others very near to them. The result is that you get a far more checkered pattern of pixel weightings in the sparse regression case, and in the least-squares regression case, you get a relatively constant weighting across all of the pixels inside the circular shape. Further, we can look at which pixels tend to really dominate in the sparse regression case. The pixel weightings in the sparse regression case are informative to how much "variance" each pixel has. For example, with an 8, pixels at the bottom center of the image are filled in, whereas in a 9, the pixels at the bottom center of the image tend to be much less filled in. Keep in mind that pixel dominance is not a measure of how often a pixel is on. If a pixel is always on for every digit, then it is irrelevant in helping classify the image. Instead, pixel dominance should be interpreted as a measure of how much a pixel is on or off between different digits. If a pixel is always on for some digits, but not for other digits, then it plays a large role in determining the classification of a generic digit image and is thus a dominant pixel. If you look at the center-right of Figure 2, you see somewhat of a vertical line of dominant pixels. My assumption is that this is on the right side of the image and not the left side because digits tend to be right side biased. Most non-symmetric digits (with the exception of "2" and "6") have more pixels on their right hand side. For the aforementioned cases of "2" and "6", this left-sided dominance is roughly canceled out by the digits "5" and "9" respectively. This leaves "3", "7", and "4" as the remaining non-symmetric digits. All of which are right side biased.

3 Individual Digit Analysis

For the individual digit analyses, we pulled out all of the images from the training dataset that corresponded to a specific digit. We ran the same pseudoinverse and lasso regression analyses as we did for the case with the full dataset. We did not check classification accuracy of the resulting trained A matrices as this regression involved training on a very biased datset. The goal of this analysis was instead to take another look at the columns of A and determine how dominant each individual pixel is to classifying a single digit. Repeating this analysis for each digit allows us to look at which pixels dominate for certain digits but not for others.

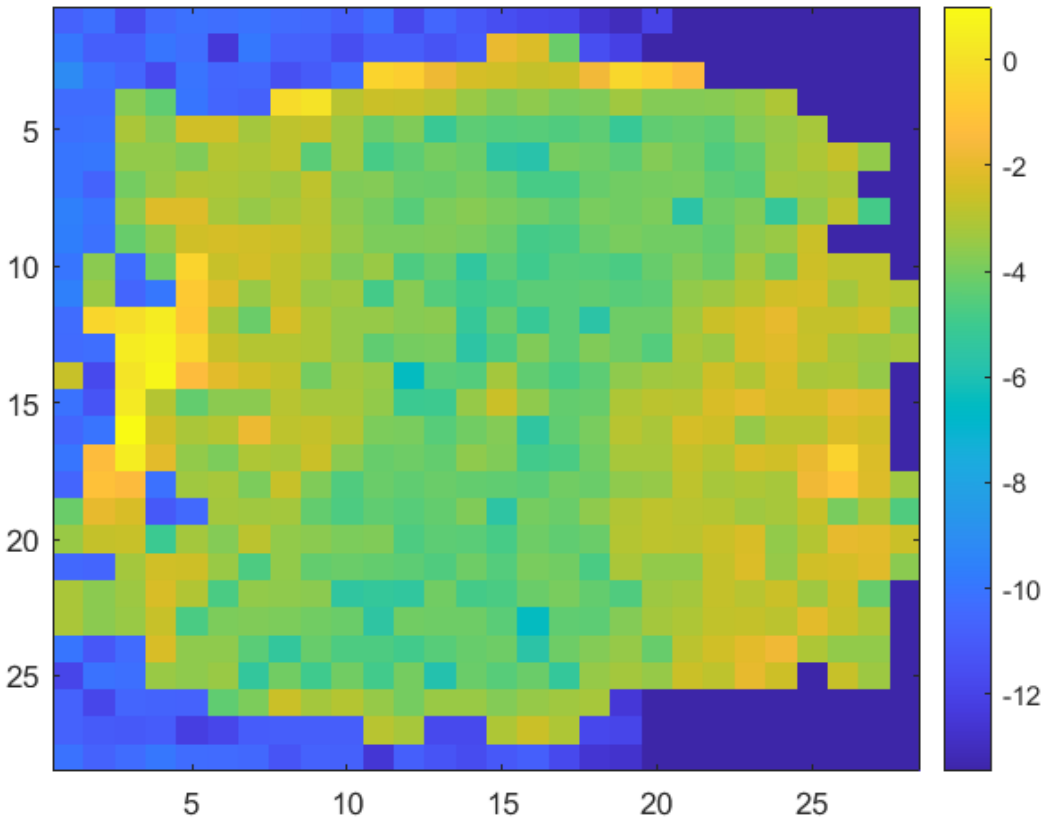

Figure 3: Dominant Pixels for Single Digit - 1

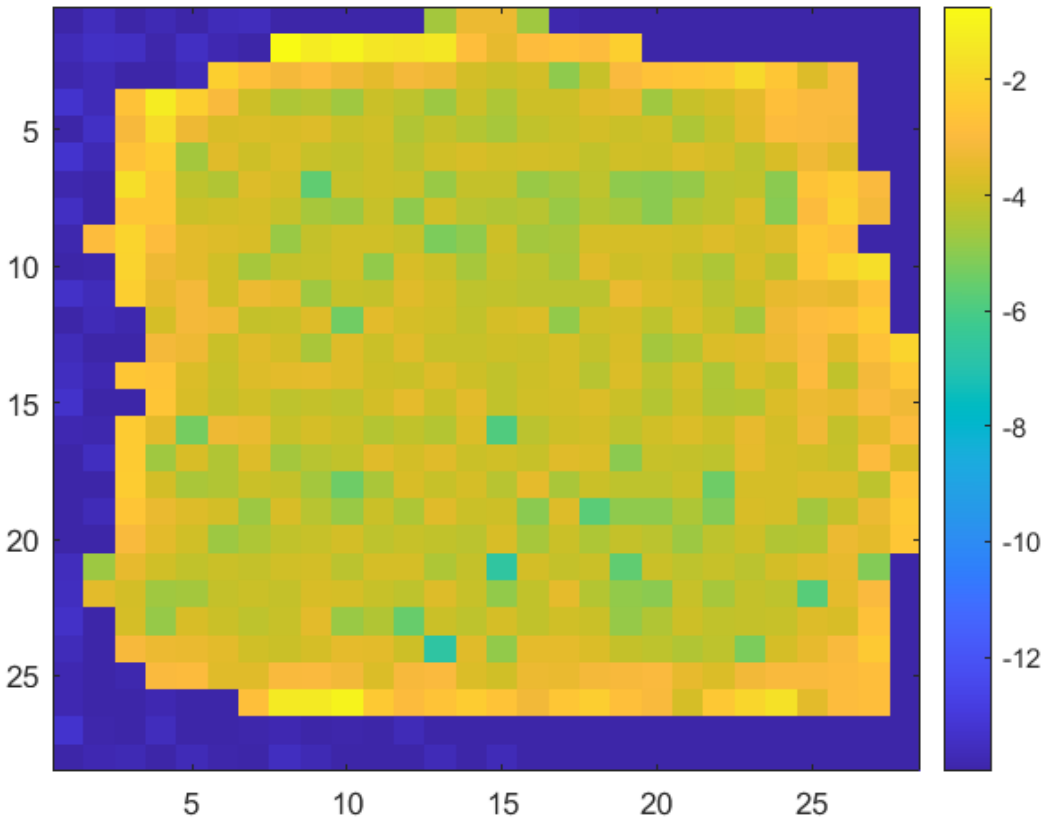

Figure 4: Dominant Pixels for Single Digit - 2

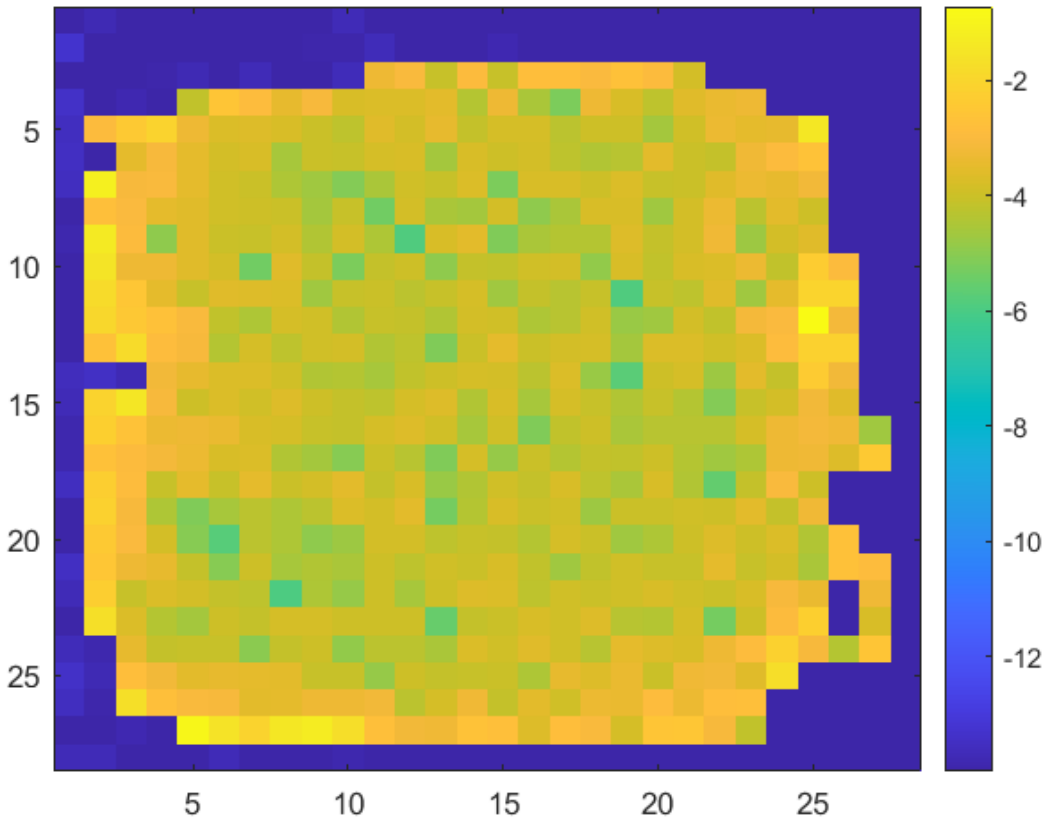

Figure 5: Dominant Pixels for Single Digit - 3

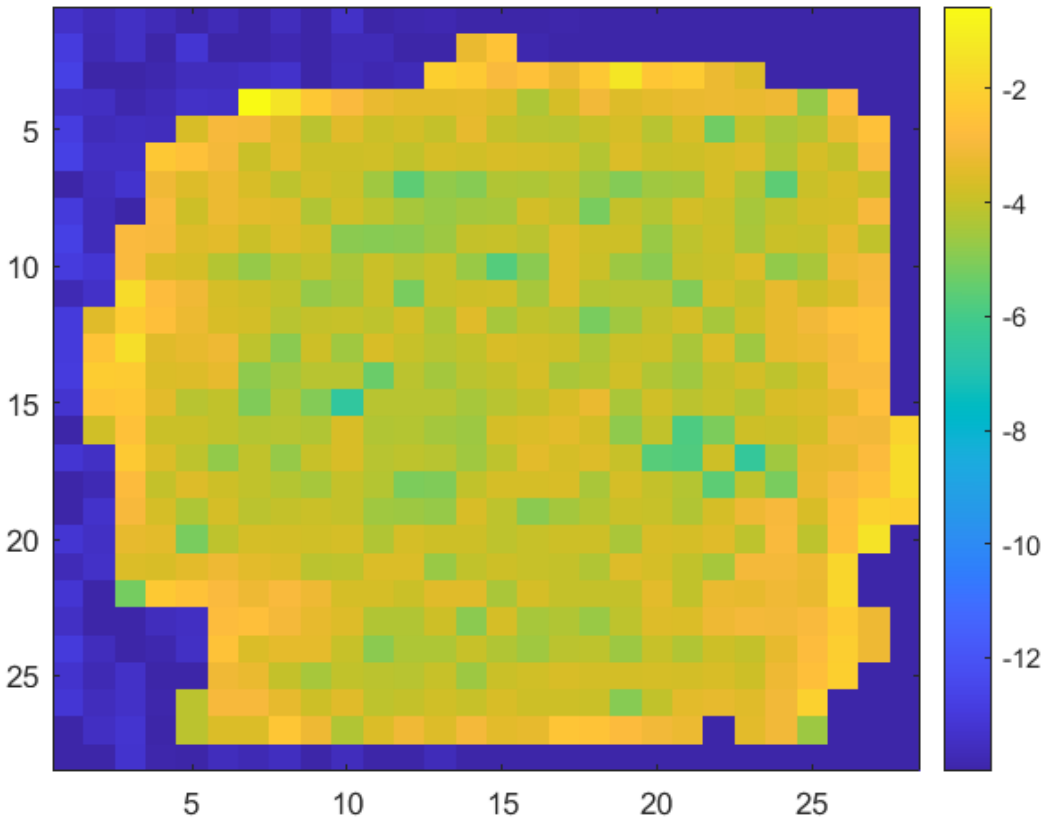

Figure 6: Dominant Pixels for Single Digit - 4

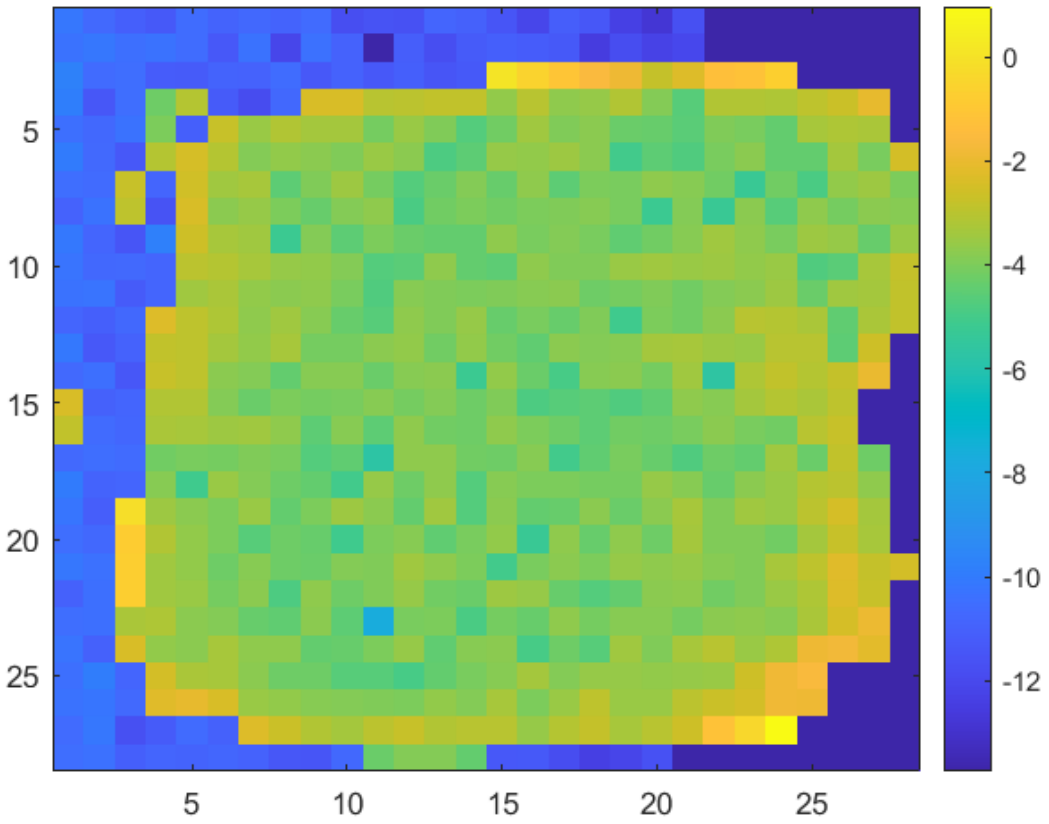

Figure 7: Dominant Pixels for Single Digit - 5

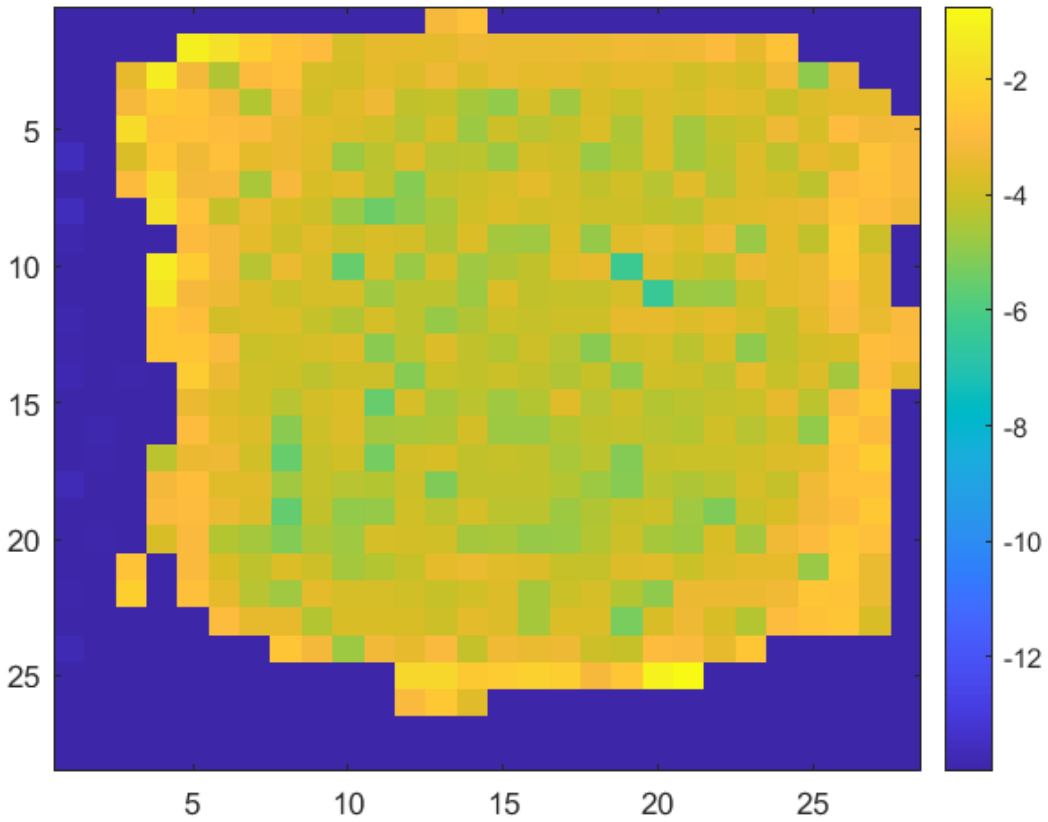

Figure 8: Dominant Pixels for Single Digit - 6

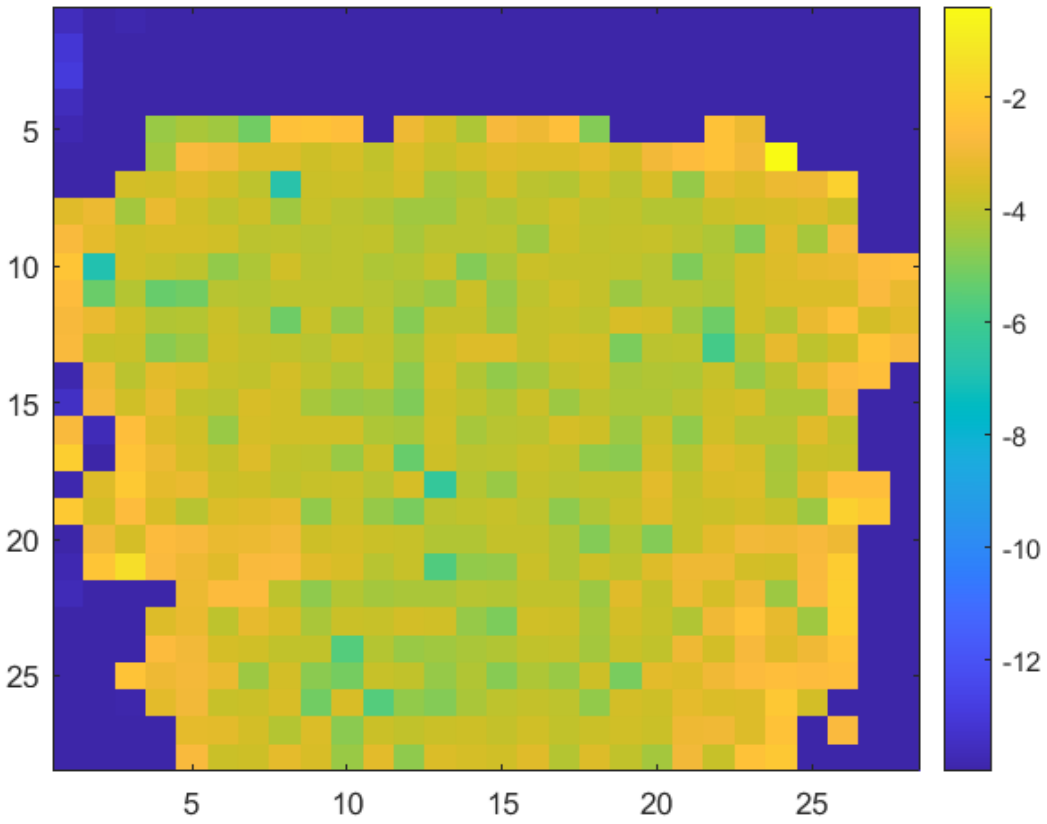

Figure 9: Dominant Pixels for Single Digit - 7

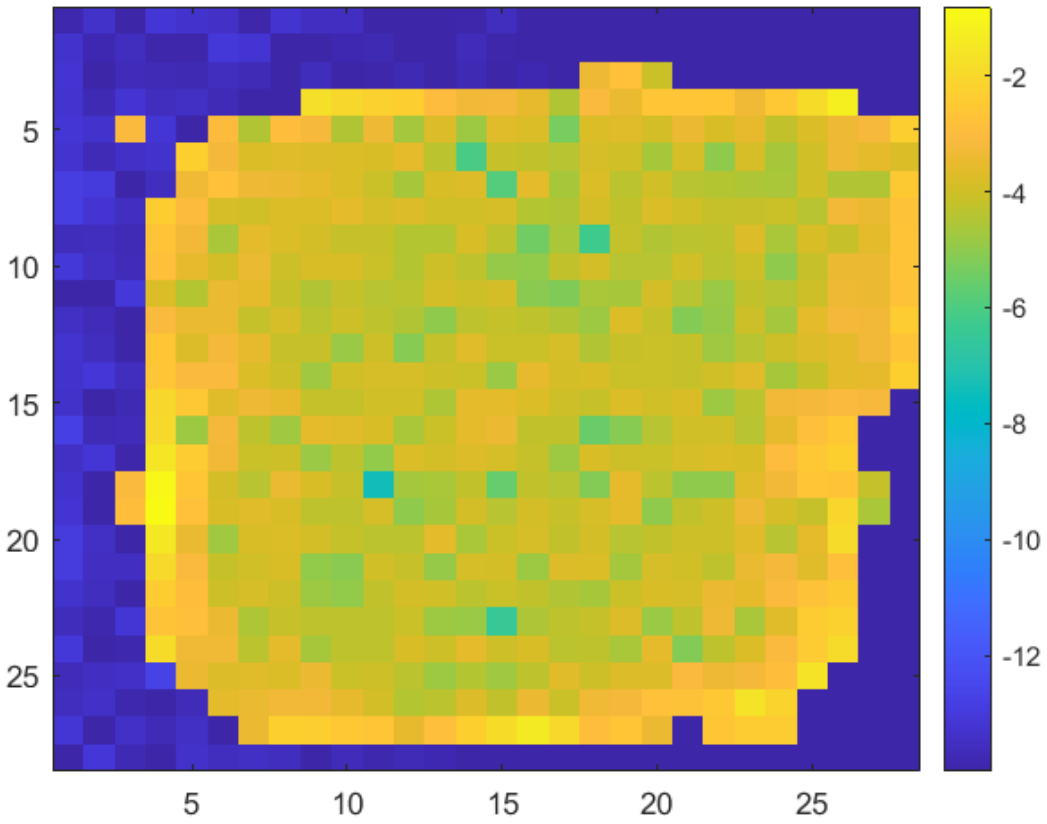

Figure 10: Dominant Pixels for Single Digit - 8

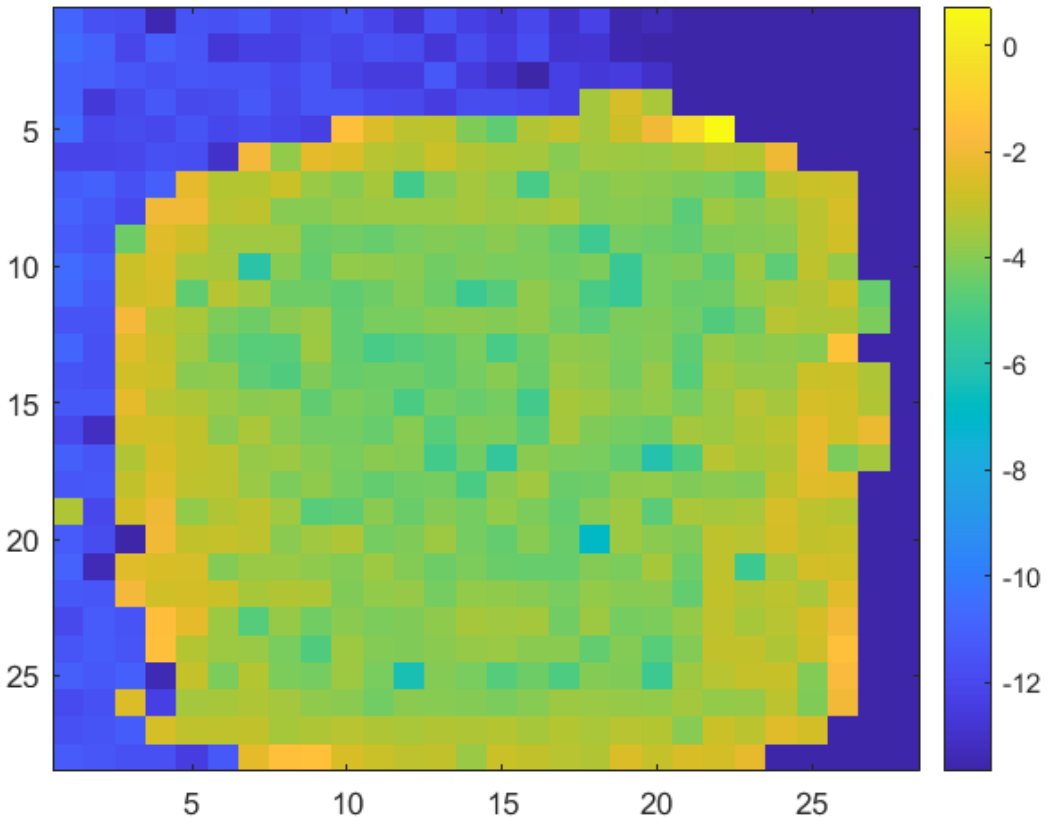

Figure 11: Dominant Pixels for Single Digit - 9

.

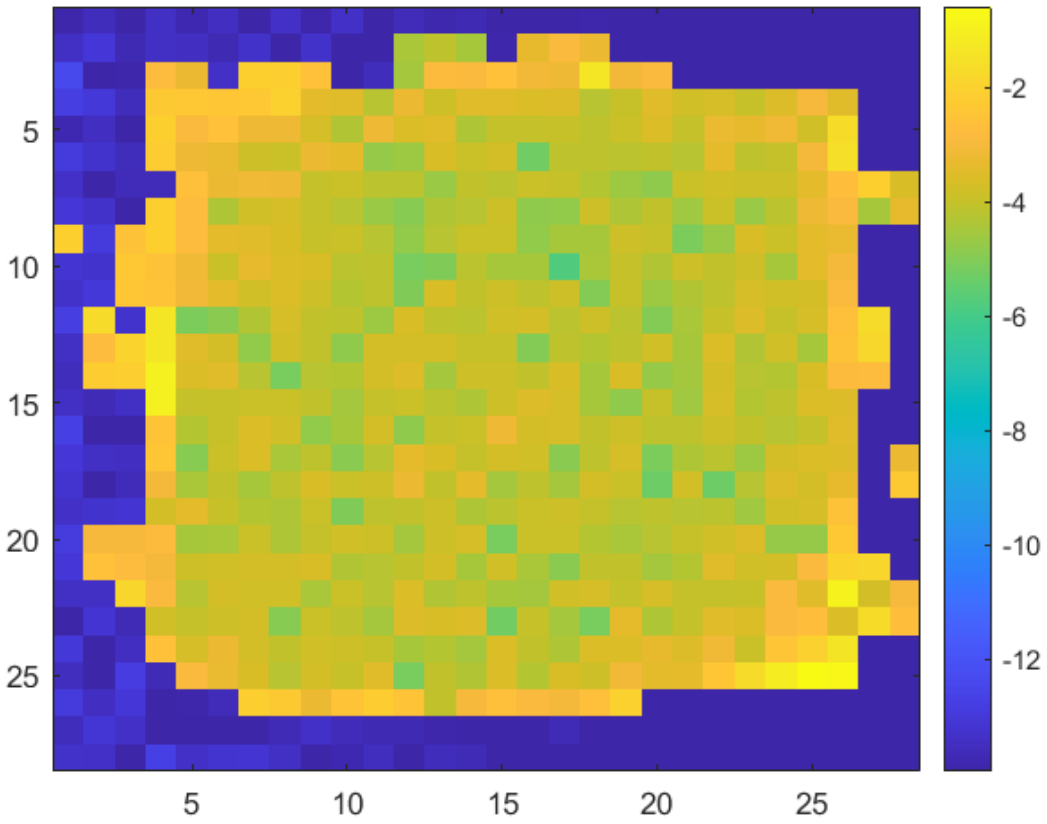

Figure 12: Dominant Pixels for Single Digit - 0

```
clear; close all; clc
%% Load training and test sets just once
% Read in Training Labels
TrainingLabels = fopen('train-labels.idx1-ubyte');
A = fread(TrainingLabels,inf,'uint8');
A_prime = A(9:end,:);
ST = fclose(TrainingLabels);

% Read in Training Images
TrainingImages = fopen('train-images.idx3-ubyte');
A2 = fread(TrainingImages,inf,'uint8');
A2_prime = A2(17:end,:);
ST2 = fclose(TrainingImages);

% Read in test labels
TestLabels = fopen('t10k-labels.idx1-ubyte');
A3 = fread(TestLabels,inf,'uint8');
A3_prime = A3(9:end,:);
ST3 = fclose(TestLabels);

% read in test images
TestImages = fopen('t10k-images.idx3-ubyte');
A4 = fread(TestImages,inf,'uint8');
A4_prime = A4(17:end,:);
ST4 = fclose(TestImages);

%% Test/debug Code
% Test Code, comment out when no longer needed ----------------------------
% % Testing reading in images and double checking against labels
% ImageIndexNum = 3410; % This input controls which image-label pair is output.
% offset = (ImageIndexNum-1)*(28*28);
%
% % Test method 1, looping through each entry in the training data vector
% % pic_test = zeros(28,28);
% % for k = 1:28
% %    for j = 1:28
% %        pic_test(k,j) = A2_prime(offset + 28*(k-1) + j,1);
% %    end
% % end
%
% % Test Method 2, using reshape command
% pic_test = A4_prime(offset+1:offset+28*28,1);
% pic_test = reshape(pic_test,28,28);
% pic_test = pic_test.';
%
% % print image and corresponding label from label vector
% % pcolor(pic_test); % pcolor is broken somehow, inverts/mirrors image
% imagesc(pic_test);
% colormap gray
```

```
% A3_prime((offset/(28*28))+1)
% End test code section ----------------------------------------------------


%% Create Matrices for Test and Train Data
% Create regression matrices
B_ForRegression = zeros(10,length(A_prime));
for j = 1:length(A_prime)
    if A_prime(j,1) == 0
        B_ForRegression(10,j) = 1;
    else
        B_ForRegression(A_prime(j,1),j) = 1;
    end
end

X_ForRegression = zeros(28*28,length(A_prime));
for k = 0:length(A_prime)-1
    pic_temp = A2_prime((k*28*28)+1:(k+1)*28*28,1);
    pic_temp = reshape(pic_temp,28,28);
    pic_temp = pic_temp.';
    pic_temp = reshape(pic_temp,28*28,1);
    X_ForRegression(:,k+1) = pic_temp;
end

% double check reshaping operation, comment when done ------------------
% index = 957;
% imagesc(reshape(A_ForRegression(:,index),28,28));
% colormap gray
% A_prime(index)
% end double checking ------------------------------------------------

% Create Testing Matrices
B_ForTesting = zeros(10,length(A3_prime));
for j = 1:length(A3_prime)
    if A3_prime(j,1) == 0
        B_ForTesting(10,j) = 1;
    else
        B_ForTesting(A3_prime(j,1),j) = 1;
    end
end

X_ForTesting = zeros(28*28,length(A3_prime));
for k = 0:length(A3_prime)-1
    pic_temp = A4_prime((k*28*28)+1:(k+1)*28*28,1);
    pic_temp = reshape(pic_temp,28,28);
    pic_temp = pic_temp.';
    pic_temp = reshape(pic_temp,28*28,1);
    X_ForTesting(:,k+1) = pic_temp;
end
```

```
%% Model Training
% pinv trained
A_Trained_Pinv = B_ForRegression*pinv(X_ForRegression);
B_output_TrainedPinvModel = A_Trained_Pinv*X_ForTesting;

[M,I] = max(B_output_TrainedPinvModel);
I = I.';
bool = I ~= 10;
I = I.*bool;
bool = I == A3_prime;
successes = sum(bool);
successrate = successes/10000;

%%
norms = 0*A_Trained_Pinv(1,:);
for j = 1:length(norms)
   norms(1,j) = norm(A_Trained_Pinv(:,j),2);
end

cutoff = 10^-14;
bool2 = norms >= cutoff;
% norms = norms.*bool2;
boolTemp = bool2;
for j = 1:9
   bool2 = [bool2;boolTemp];
end

A_Trained_Pinv_Sparse = bool2.*A_Trained_Pinv;
B_output_TrainedPinvModel_Sparse = A_Trained_Pinv_Sparse*X_ForTesting;
[M,I] = max(B_output_TrainedPinvModel_Sparse);
I = I.';
bool = I ~= 10;
I = I.*bool;
bool = I == A3_prime;
successes = sum(bool);
successrate = successes/10000;


temp = norms.';
temp = log10(temp);
temp = reshape(temp,28,28);
imagesc(temp); colorbar

%%
% lasso trained
lambdaVals = [0.01,0.001,0.0001,0.00001];
lambdaVals = fliplr(lambdaVals);
A_Trained_Lasso = zeros(size(A_Trained_Pinv.',1),size(A_Trained_Pinv.',2),length ↙
```

```
(lambdaVals));

for k = 1:10
[A_vec_Trained_Lasso,~] = lasso(X_ForRegression.',B_ForRegression(k,:).', 'Lambda',↙
lambdaVals);
A_Trained_Lasso(:,k,:) = A_vec_Trained_Lasso;

end

%%
B_output_TrainedLassoModel = zeros(size(B_output_TrainedPinvModel,1),size↙
(B_output_TrainedPinvModel,2),4);
for k = 1:4
    B_output_TrainedLassoModel(:,:,k) = A_Trained_Lasso(:,:,k).'*X_ForTesting;

end

maxes_B_Lasso = zeros(4,size(B_output_TrainedLassoModel,2));
for k = 1:4
    [M,I] = max(B_output_TrainedLassoModel(:,:,k));
    maxes_B_Lasso(k,:) = I;
end
maxes_B_Lasso = maxes_B_Lasso.';
maxes_B_Lasso = maxes_B_Lasso.*(maxes_B_Lasso ~= 10);

A3_prime_expanded = [A3_prime,A3_prime,A3_prime,A3_prime];
bool = maxes_B_Lasso == A3_prime_expanded;
successrate_lasso = sum(bool);
successrate_lasso = successrate_lasso *(1/10000);

%%
A_temp = A_Trained_Lasso(:,:,4);
A_temp = A_temp.';
norms2 = zeros(1,size(A_temp,2));
for j = 1:length(norms2)
   norms2(1,j) = norm(A_temp(:,j),2);
end

norms2 = log10(norms2);
norms2_reshaped = reshape(norms2.',28,28);
figure;
imagesc(norms2_reshaped); colorbar;

%% Repeated analysis for each digit individually
A_Trained_Pinv_Short_All = zeros(10*size(A_Trained_Pinv,1),size(A_Trained_Pinv,2));
for q = 0:9
    bool_digit = A_prime == q;
    Q = find(bool_digit);
```

```
    % Pull out the training data corresponding to the indices in Q here
    X_Regression_Short = zeros(784,length(Q));
    for j = 1:length(Q)
        offset = Q(j)-1;
        tempVector = A2_prime((offset*784+1):(offset*784+784),1);
        tempVector = reshape(tempVector,28,28);
        tempVector = tempVector.';
        tempVector = reshape(tempVector,784,1);
        X_Regression_Short(:,j) = tempVector;
    end

    B_Regression_Short = zeros(10,1);
    B_Regression_Short(q+1,1) = 1;
    B_Regression_Short = [B_Regression_Short(2:10,1);B_Regression_Short(1,1)];
    tempVector = B_Regression_Short;
    for j = 1:length(Q)-1
        B_Regression_Short = [B_Regression_Short,tempVector];
    end

    % Once you have that, you can repeat the lasso regression.
    % comment out lasso approach ----------------------------------------
%     lambda = 0;
%     A_Trained_Lasso_Short = zeros(10*size(A_Trained_Pinv,1),size(A_Trained_Pinv, ↙
2));
%     for k = 1:10
%         [A_vec_Trained_Lasso_Short,~] = lasso(X_Regression_Short.', ↙
B_Regression_Short(k,:).','Lambda',lambda);
%         A_vec_Trained_Lasso_Short = A_vec_Trained_Lasso_Short.';
%         A_Trained_Lasso_Short(10*q+k,:) = A_vec_Trained_Lasso_Short;
%     end
    % ---------------------------------------------------------------------

    % lasso regression seems to not like a column vector that is all the
    % same number (in this case either 1 or 0) so try pinv() approach instead.
    A_Trained_Pinv_Short = B_Regression_Short*pinv(X_Regression_Short);

    A_Trained_Pinv_Short_All(10*q+1:10*q+10,:) = A_Trained_Pinv_Short;


end

A_Trained_Pinv_Short_All = [A_Trained_Pinv_Short_All(11:end,:); ↙
A_Trained_Pinv_Short_All(1:10,:)];

%%
cutoff_second = 10^-14;
bool4 = abs(A_Trained_Pinv_Short_All) >= cutoff_second;
A_Trained_Pinv_Short_All_Amended = bool4.*A_Trained_Pinv_Short_All;
A_Trained_Pinv_Short_All_Amended = A_Trained_Pinv_Short_All_Amended.';
```

```
A_Trained_Pinv_Short_All_Amended = abs(A_Trained_Pinv_Short_All_Amended);

%%
for j = 1:11:100
   temp = reshape(log10(A_Trained_Pinv_Short_All_Amended(:,j)),28,28);
   figure;
   imagesc(temp); colorbar;
end
```