```
%==========================================================================
====
%LOADS IMAGE
%==========================================================================
====

pkg load image

directory_path = '/Users/charlesdavi/Desktop/MNIST/MNIST - JPG - training/';

counter = 1;

for k = 0 : 9

num_images = 250;

category_number = k;

directory = [directory_path int2str(k) '/'];

for i = 1 : num_images

I = imread([directory '1 (' int2str(i) ').jpg']);

MNIST_array{counter} = I;
MNIST_category{counter} = k; %stores the classifier associated with the image
counter = counter + 1;

endfor

endfor

%==========================================================================
====
%EXTRACTS SHAPE INFORMATION
%==========================================================================
====
tic;
[final_avg_matrix final_indexes] = partition_image_vectorized_gs(MNIST_array{1}); %this is to size the
partitions for the entire dataset

N = size(final_avg_matrix,1);

total_num_images = 10*num_images; %there are 10 categories of images corresponding to the digits
0 through 9

counter = 1; %index for the observations
dataset = [];

scramble = randperm(total_num_images,total_num_images); %this is to permute the dataset, which
would otherwise be sorted by digit

%iterates through entire dataset
```

```
while(counter <= total_num_images)

I = MNIST_array{scramble(counter)};

[avg_matrix] = calc_avg_color_vect(final_indexes, I, N); %this extracts shape information

input_vector = reshape(avg_matrix, [1 N^2]);

input_vector(N^2+1) = MNIST_category{scramble(counter)}; %this is the hidden classifier

dataset(counter,:) = input_vector;

counter = counter + 1;

endwhile
toc

%================================================================
%EXTRACTS BACKGROUND
%================================================================

s_vector = std(dataset(:,1:N^2)); %the standard deviation of every dimension
avg = mean(s_vector); %the average standard deviation

bin_vector = s_vector > avg; %binary vector of dimensions greater than the mean

dataset(:,1:N^2) = dataset(:,1:N^2).*bin_vector; %sets likely background entries to zero

%================================================================
%MAKES PREDICTIONS; TESTS ERROR
%================================================================

num_rows = size(dataset,1);

tic;nearest_neighbors = NN_fully_vectorized(dataset, N^2);toc

error_vector = dataset(nearest_neighbors, N^2+1) != dataset(:, N^2+1);

num_errors = sum(error_vector);

accuracy = 1 - num_errors / num_rows
```