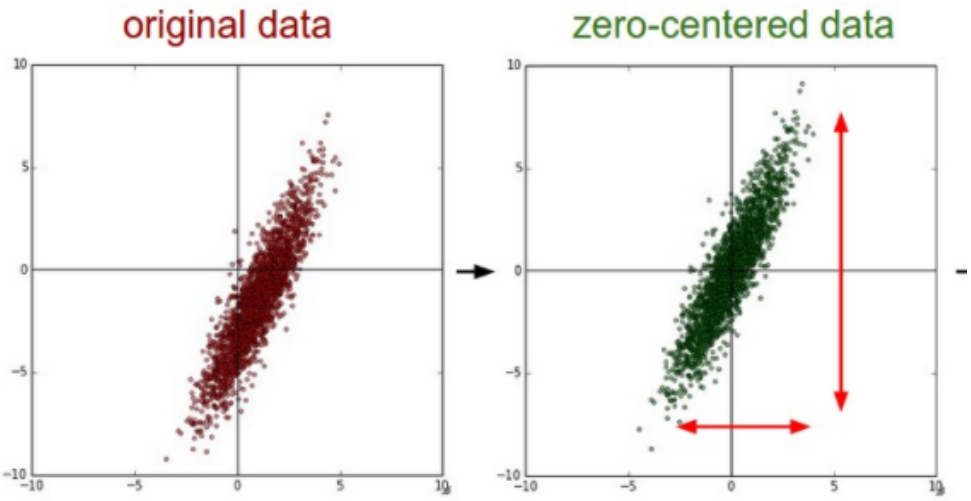
Computer Vision *Name:Abhay Doke UID:29552668*

Mini Project 04

1.Pixel features

For the pixel features I experimented with two methods where two sets of different normalizations were applied on the data.

Method 1: Making the data zero centered by mean-subtraction. Data is made zero centered by subtracting the mean across every individual feature in the data, and results in centering the cloud of data around the origin along every dimension.



Method 2: Applying the L2-normalization followed by the square-root scaling.

Following are the accuracy result scores obtained by these two methods on the validation set and the test set.

```
Model Parameters:
function model = trainModel(x, y)
param.lambda = 0.001;  % Regularization term
param.maxiter = 1000; % Number of iterations
param.eta = 0.001;     % Learning rate



Results:
Method 1
Normalization used: Zero Centered Data

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
```

```
--------------------------------------------------
dataset pixel
--------------------------------------------------
digits-normal.mat 85.60
digits-scaled.mat 82.80
digits-jitter.mat 21.20

Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset pixel
--------------------------------------------------
digits-normal.mat 83.00
digits-scaled.mat 79.40
digits-jitter.mat 19.00


Method 2
Normalization used: L2 normalization followed by square root scaling

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset pixel
--------------------------------------------------
digits-normal.mat 87.60
digits-scaled.mat 73.00
digits-jitter.mat 20.80

Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset pixel
--------------------------------------------------
digits-normal.mat 84.20
digits-scaled.mat 68.00
digits-jitter.mat 18.80
```

For the normal data, the L2-normalization followed by the square-root scaling performed the best with 87.6% accuracy on the validation set. For the scaled data and the jittered data, mean-subtraction(method 1) worked better on both the validation as well as test dataset.

2.HoG features

I used the 4 x 4 patches in the images to calculate the weighted histogram of oriented gradients. Image size is 28 x 28, so there are total 7 x 7 = 49 patches in each image. Orientation values ranged from 0 - 360 degree and these values are binned into 18 bins viz. 0-20, 21-40, 41-60,...., 341-360. So total number of features per image are 49*18 = 882.

Method 1: No normalization

Method 2: L2-normalization followed by the square-root scaling. Following results shows the validation set and the test set accuracies.

```
HoG features are calculated over 4 x 4 patches in the images.
Image size is 28 x 28, so there are total 7 x 7 = 49 patches in each image.
Orientation values ranged from 0 - 360 degree and these values are binned into
18 bins viz. 0-20, 21-40, 41-60,...., 341-360.
So total number of features per image are 49*18 = 882

Method 1: No normalization

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset hog
--------------------------------------------------
digits-normal.mat 91.20
digits-scaled.mat 91.20
digits-jitter.mat 39.00

Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset hog
--------------------------------------------------
digits-normal.mat 91.20
digits-scaled.mat 90.40
digits-jitter.mat 33.80

Method 2: L2-normalization followed by the square-root scaling

+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset hog
--------------------------------------------------
digits-normal.mat 93.60
digits-scaled.mat 93.60
digits-jitter.mat 46.40

+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset hog
--------------------------------------------------
digits-normal.mat 95.80
digits-scaled.mat 96.00
digits-jitter.mat 39.80
```

```
Hyperarameters:

numOri = 0 - 360
histogramSize(binSize) = 18

param.lambda = 0.001;  % Regularization term
param.maxiter = 1000; % Number of iterations
param.eta = 0.001;     % Learning rate
```

HoG features work very well for the normal data and the scaled data. L2-normalization followed by the square-root scaling worked better on all of the datasets as compared to the no normalization strategy.

3.LBP features

For the LBP features I used two methods for calculating the LBP features. The main observation of the LBP feature experiment was the exclusion of LBP feature with value 255 from the feature vector really improved the accuracies for all three datasets. LBP feature with value 255 represents either all surrounding pixels are same or greater than the current pixel. Since most part of the images are either black or white, this feature is not representing any significant information. Also the scaling the pixel values in the images do not alter the LBP accuracies i.e. the accuracies on the normal data and the scaled data with LBP features are almost the same.

Method 1: Calculating a single histogram of LBP features for the whole image. The resulting histogram size was 255 and hence the feature vector had 255 features.

Method 2: Each image was divided in two 4 parts and LBP feature histogram were calculated independently for all 4 patches. All of the four histograms were concatenated to obtain the final feature vector. The resulting feature vector had 255*4 = 1020 features.

```
Method 1: Calculating LBP features on the whole image.
****** Hyperparameters for the Method 1 ***********
param.lambda = 0.001;  % Regularization term
param.maxiter = 2000; % Number of iterations
param.eta = 0.01;     % Learning rate
**************************************************

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset lbp
--------------------------------------------------
digits-normal.mat 65.20
digits-scaled.mat 65.20
digits-jitter.mat 63.40

Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset lbp
--------------------------------------------------
digits-normal.mat 62.40
digits-scaled.mat 62.40
digits-jitter.mat 59.80


Method 2: Calculating LBP features by dividing an image into 4 parts,
calculating LBP histogram for all four parts and then concatenating all four
****** Hyperparameters for the Method 2 ***********
histograms to obtain a feature vector of size 1020.
param.lambda = 0.001;  % Regularization term
param.maxiter = 1000; % Number of iterations
param.eta = 0.001;     % Learning rate
**************************************************

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
```

```
dataset lbp
--------------------------------------------------
digits-normal.mat 91.20
digits-scaled.mat 91.40
digits-jitter.mat 55.40

Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset lbp
--------------------------------------------------
digits-normal.mat 90.80
digits-scaled.mat 91.60
digits-jitter.mat 52.80
```

LBP features are representing the local patterns in images. Also they are invariant to the rotation and scaling to some extent. Hence the classification accuracy for the jittered data is better than other two feature types.

Below is the table showing the best accuracies achieved for each of the three datasets.

```
BEST Validation set accuracies:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset             pixel    hog     lbp
--------------------------------------------------
digits-normal.mat   87.60   93.60   91.20
digits-scaled.mat   82.80   93.40   91.40
digits-jitter.mat   21.20   46.40   63.40


BEST Test set accuracies:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset             pixel    hog     lbp
--------------------------------------------------
digits-normal.mat   84.20   95.80   90.80
digits-scaled.mat   68.00   96.00   91.60
digits-jitter.mat   19.00   39.80   59.80
```

Which feature works the best on normal digits? Why? – HoG features works the best for the normal digits. HoG features captures the local gradient patterns over the many patches in the images. Digits have fixed gradient patterns over the different areas and if the data is not jittered like rotated or skewed these features will capture the most information from the data.

Which feature works the best on scaled digits? Why? – Again data is scaled i.e. pixel values are scaled and as digits images are almost black and white, the local gradient patterns in the images still remains the same. Hence HoG features works the best followed by LBP features.

Which feature works the best on jittered digits? Why? – In the jittered data images are translated. As images are translated we need an image descriptor which is invariant to small translations and LBP are the best features to handle such data. LBP feature histogram remains the same if the image is translated. For example in an image number of pixels with value 10 are 100 and now if we translate the image, only pixel

positions are shifted but the local patterns remains the same and hence number of pixels with value 10 will still be around 10.

Code for extractDigitFeatures.m

```
function features = extractDigitFeatures(x, featureType)
% EXTRACTDIGITFEATURES extracts features from digit images
%   features = extractDigitFeatures(x, featureType) extracts FEATURES from images
%   images X of the provided FEATURETYPE. The images are assumed to the of
%   size [W H 1 N] where the first two dimensions are the width and height.
%   The output is of size [D N] where D is the size of each feature and N
%   is the number of images.

switch featureType
    case 'pixel'
        features = zeroFeatures(x);
        features = reshape(x, [784,2000]);
        %features = features - mean(features,2);
        %features = features./sum((features.^2),1);
        %features = features ./ std(features,0,2);
        %features = (features - min(features,[],2))./(max(features,[], 2) - min(features,[],2));

        %features = sqrt(features);
        features = features./sqrt(sum((features.^2),1));
        features = sqrt(features);
        disp(size(features));

    case 'hog'
        %features = zeroFeatures(x);
        [W,H,C,N] = size(x);
        blockSize = 4;
        overlap = 0;
        stepSize = int16(blockSize*(1-overlap));
        binAngle = 20;
        histogramSize = int16(360 / binAngle);

        features = [];
        for imageNumber = 1:N
            image = x(:,:,1,imageNumber);
            image = reshape(image,[W,H]);
            [h,w] = size(image);
            [gx, gy] = imgradientxy(image);
            [gMag, gDir] = imgradient(gx,gy);

            totalSteps = int16(h/stepSize);

            imageHistogram = [];

            for i = 1:totalSteps
                for j = 1:totalSteps
                    startRow = (i-1)*stepSize;
                    startColumn = (j-1)*stepSize;
                    tempMag = gMag(startRow+1:startRow+blockSize,startColumn+1:startColumn+blockSize);
                    tempDir = gDir(startRow+1:startRow+blockSize,startColumn+1:startColumn+blockSize);
                    negativeDirIndices = find(tempDir<0);
                    tempDir(negativeDirIndices) = tempDir(negativeDirIndices)+360;
                    tempHistogram = zeros(1,18);
```

```
                    for blockH = 1:blockSize
                        for blockW = 1:blockSize
                            %disp(tempDir(blockH,blockW));
                            binNumber = int16(floor(mod(tempDir(blockH,blockW),360)/binAngle)+1);
                            tempHistogram(binNumber) = tempHistogram(binNumber) + tempMag(blockH,blockW
                        end
                    end
                    imageHistogram = cat(2,imageHistogram,tempHistogram);
                end
            end
            features = cat(1,features,imageHistogram);
        end

        features = features';
        features = features./sqrt(sum((features.^2),1));
        features = sqrt(features);
        disp(size(features));


%     case 'lbp'
%         %features = zeroFeatures(x);
%         mask = [1,2,4;128,0,8;64,32,16];
%         [W,H,C,N] = size(x);
%         features = [];
%         x = reshape(x, [784,2000]);
%         %x = x - mean(x,2);
%         x = x./sum((x.^2),1);
%         %x = sqrt(x);
%         x = reshape(x,[28,28,1,2000]);
%         for imageNumber = 1:N
%             image = x(:,:,1,imageNumber);
%             image = reshape(image, [W,H]);
%             lbpVector = zeros(1,255);
%             for pixelH = 2:H-1
%                 for pixelW = 2:W-1
%                     pixelVal = image(pixelH, pixelW);
%                     patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
%                     %disp(patch)
%                     lessThanIndices = find(patch<pixelVal);
%                     greaterThanIndices = find(patch>=pixelVal);
%                     patch(lessThanIndices) = 0;
%                     patch(greaterThanIndices) = 1;
%                     %disp(patch);
%
%                     patch(2,2) = 0;
%                     patch = patch.*mask;
%
%                     LBP = sum(sum(patch));
%                     %disp(patch);
%                     %disp(LBP);
%                     if(LBP~=255)
%                         %lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
%                         lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
%                     end
```

```
%
%                 end
%             end
%             %disp(features);
%
%             features = cat(1,features,lbpVector);
%
%         end
%         %disp(features(1,:));
%         %disp(size(features));
%         %features = features - mean(features,2);
%         features = features';
%
%         %disp(size(features));
%     case 'lbp'
%         zero = [0];
%         one = [1     2     4     8    16    32    64   128];
%         two = [3     6    12    24    48    96   192   129];
%         three = [7    14    28    56   112   224   193   131];
%         four = [15    30    60   120   240   225   195   135];
%         five = [31    62   124   248   241   227   199   143];
%         six = [63   126   252   249   243   231   207   159];
%         seven = [127   254   253   251   247   239   223   191];
%         eight = [255];
%
%         mask = [1,2,4;128,0,8;64,32,16];
%         [W,H,C,N] = size(x);
%         features = zeros(N,9);
%         %x = reshape(x, [784,2000]);
%         %x = x - mean(x,2);
%         %x = x./sum((x.^2),1);
%         %x = reshape(x,[28,28,1,2000]);
%         for imageNumber = 1:N
%             image = x(:,:,1,imageNumber);
%             image = reshape(image, [W,H]);
%             lbpVector = zeros(1,255);
%             for pixelH = 2:H-1
%                 for pixelW = 2:W-1
%                     pixelVal = image(pixelH, pixelW);
%                     patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
%                     %disp(patch)
%                     lessThanIndices = find(patch<pixelVal);
%                     greaterThanIndices = find(patch>=pixelVal);
%                     patch(lessThanIndices) = 0;
%                     patch(greaterThanIndices) = 1;
%                     %disp(patch);
%
%                     patch(2,2) = 0;
%
%                     patch = patch.*mask;
%
%                     LBP = sum(sum(patch));
%                     %disp(patch);
%                     %disp(LBP);
```

```
%                     if LBP==0
%                         features(imageNumber,1) = features(imageNumber,1)+1;
%                     elseif ismember(LBP,one)
%                         features(imageNumber,2) = features(imageNumber,2)+1;
%                     elseif ismember(LBP,two)
%                         features(imageNumber,3) = features(imageNumber,3)+1;
%                     elseif ismember(LBP,three)
%                         features(imageNumber,4) = features(imageNumber,4)+1;
%                     elseif ismember(LBP,four)
%                         features(imageNumber,5) = features(imageNumber,5)+1;
%                     elseif ismember(LBP,five)
%                         features(imageNumber,6) = features(imageNumber,6)+1;
%                     elseif ismember(LBP,six)
%                         features(imageNumber,7) = features(imageNumber,7)+1;
%                     elseif ismember(LBP,seven)
%                         features(imageNumber,8) = features(imageNumber,8)+1;
%                     elseif LBP==255
%                      %   features(imageNumber,9) = features(imageNumber,9)+1;
%                     else
%                         features(imageNumber,9) = features(imageNumber,9)+1;
%
%                     end
%
%                 end
%             end
%             %disp(features);
%
%             %features = cat(1,features,lbpVector);
%
%         end
%         disp(features(1,:));
%         %disp(size(features));
%         %features = features - mean(features,2);
%         features = features';
%
%         %disp(size(features));
%
case 'lbp'
        %features = zeroFeatures(x);
        mask = [1,2,4;128,0,8;64,32,16];
        [W,H,C,N] = size(x);
        features = [];
        %x = reshape(x, [784,2000]);
        %x = x - mean(x,2);
        %x = x./sqrt(sum((x.^2),1));
        %x = sqrt(x);
        %x = reshape(x,[28,28,1,2000]);
        for imageNumber = 1:N
            image = x(:,:,1,imageNumber);
            image = reshape(image, [W,H]);
            fullVector = [];
            lbpVector = zeros(1,255);
            for pixelH = 2:int16((H-1)/2)
                for pixelW = 2:int16((W-1)/2)
```

```
            pixelVal = image(pixelH, pixelW);
            patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
            %disp(patch)
            lessThanIndices = find(patch<pixelVal);
            greaterThanIndices = find(patch>=pixelVal);
            patch(lessThanIndices) = 0;
            patch(greaterThanIndices) = 1;
            %disp(patch);

            patch(2,2) = 0;
            patch = patch.*mask;

            LBP = sum(sum(patch));
            %disp(patch);
            %disp(LBP);
            if(LBP~=255)
                %lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
                lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
            end

        end
    end
    fullVector = cat(2,fullVector,lbpVector);
    lbpVector = zeros(1,255);
    for pixelH = 2:int16((H-1)/2)
        for pixelW = int16((W-1)/2):W-1
            pixelVal = image(pixelH, pixelW);
            patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
            %disp(patch)
            lessThanIndices = find(patch<pixelVal);
            greaterThanIndices = find(patch>=pixelVal);
            patch(lessThanIndices) = 0;
            patch(greaterThanIndices) = 1;
            %disp(patch);

            patch(2,2) = 0;
            patch = patch.*mask;

            LBP = sum(sum(patch));
            %disp(patch);
            %disp(LBP);
            if(LBP~=255)
                %lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
                lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
            end

        end
    end
    %disp(features);
    fullVector = cat(2,fullVector,lbpVector);

    lbpVector = zeros(1,255);
    for pixelH = int16((H-1)/2):H-1
        for pixelW = 2:int16((W-1)/2)
```

```
            pixelVal = image(pixelH, pixelW);
            patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
            %disp(patch)
            lessThanIndices = find(patch<pixelVal);
            greaterThanIndices = find(patch>=pixelVal);
            patch(lessThanIndices) = 0;
            patch(greaterThanIndices) = 1;
            %disp(patch);

            patch(2,2) = 0;
            patch = patch.*mask;

            LBP = sum(sum(patch));
            %disp(patch);
            %disp(LBP);
            if(LBP~=255)
                %lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
                lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
            end

        end
    end
    %disp(features);
    fullVector = cat(2,fullVector,lbpVector);

    lbpVector = zeros(1,255);
    for pixelH = int16((H-1)/2):H-1
        for pixelW = int16((W-1)/2):W-2
            pixelVal = image(pixelH, pixelW);
            patch = image(pixelH-1:pixelH+1,pixelW-1:pixelW+1);
            %disp(patch)
            lessThanIndices = find(patch<pixelVal);
            greaterThanIndices = find(patch>=pixelVal);
            patch(lessThanIndices) = 0;
            patch(greaterThanIndices) = 1;
            %disp(patch);

            patch(2,2) = 0;
            patch = patch.*mask;

            LBP = sum(sum(patch));
            %disp(patch);
            %disp(LBP);
            if(LBP~=255)
                %lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
                lbpVector(LBP+1) = lbpVector(LBP+1) + 1;
            end

        end
    end
    %disp(features);
    fullVector = cat(2,fullVector,lbpVector);
```

```
            features = cat(1,features,fullVector);

        end
        %disp(features(1,:));
        %disp(size(features));
        %features = features - mean(features,2);
        %features = sqrt(features);
        %features = features./sqrt(sum((features.^2),1));
        features = sqrt(features);
        features = features';


        disp(size(features));
end

function features = zeroFeatures(x)
features = zeros(10, size(x,4));
```

For the LBP features, I tried 2-3 methods. One was from `http://www.springer.com/cda/content/document/cda_downloaddocument/9780857297471-c2.pdf?SGWID=0-0-45-1153741-p174122174` "Local Binary Patterns for Still Images". One which I tried was, Mappings of the LBP Labels: Uniform Patterns. This method generates LBP histogram with 9 bins only. Only LBP values representing uniform patterns such as Spot, Spot/Flat, Line End, Edge and Corner are kept as LBP features.

4. Extension – Random forest classifier

```
NumTrees = 100

Validation set accuracy:
+++ Accuracy Table [trainSet=1, testSet=2]
--------------------------------------------------
dataset pixel hog lbp
--------------------------------------------------
digits-normal.mat 84.60 90.00 79.00
digits-scaled.mat 80.60 90.00 79.40
digits-jitter.mat 25.00 35.40 40.60


Test set accuracy:
+++ Accuracy Table [trainSet=1, testSet=3]
--------------------------------------------------
dataset             pixel    hog     lbp
--------------------------------------------------
digits-normal.mat   78.20   84.80   77.00
digits-scaled.mat 76.00   85.80   76.20
digits-jitter.mat 24.20   36.00   40.80
```

Code of randomForest.m

```
% There are three versions of MNIST dataset
dataTypes = {'digits-normal.mat', 'digits-scaled.mat', 'digits-jitter.mat'};

% You have to implement three types of features
%featureTypes = {'pixel', 'hog', 'lbp'};
featureTypes = {'lbp'};

% Accuracy placeholder
accuracy = zeros(length(dataTypes), length(featureTypes));
trainSet = 1;
testSet = 3; % 2 = validation, 3 = test

for i = 1:length(dataTypes),
    dataType = dataTypes{i};
    % load data
    load(fullfile('..', 'data', dataType));
    fprintf('+++ Loading digits of dataType: %s\n', dataType);

    % Optionally montage the digits in the val set
    % montageDigts(data.x(:,:,:,data.set==2));

    for j = 1:length(featureTypes),
        featureType = featureTypes{j};
```

```
        % Extract features
        tic;
        features = extractDigitFeatures(data.x, featureType);
        fprintf(' %.2fs to extract %s features for %i images\n', toc, featureType, size(features,2));

        % Train model
        tic;
        model = TreeBagger(20,features(:, data.set==trainSet)', data.y(data.set==trainSet), 'Method', '

        fprintf(' %.2fs to train model\n', toc);

        % Test the model

        ypred = model.predict(features(:, data.set==testSet)');
        ypred = str2double(ypred);
        y = data.y(data.set==testSet);
        % Measure accuracy
        %disp(size(ypred));
        %disp(size(y));
        ypred = ypred';
        [acc, conf] = evaluateLabels(y, ypred, false);
        fprintf(' Accuracy [testSet=%i] %.2f%%\n\n', testSet, acc*100);
        accuracy(i,j) = acc;
    end
end

% Print the results in a table
fprintf('+++ Accuracy Table [trainSet=%i, testSet=%i]\n', trainSet, testSet);
fprintf('-------------------------------------------------\n');
fprintf('dataset\t\t\t');
for j = 1:length(featureTypes),
    fprintf('%s\t',featureTypes{j});
end
fprintf('\n');
fprintf('-------------------------------------------------\n');
for i = 1:length(dataTypes),
    fprintf('%s\t', dataTypes{i});
    for j = 1:length(featureTypes),
        fprintf('%.2f\t', accuracy(i,j)*100);
    end
    fprintf('\n');
end

% Once you have optimized the hyperparameters, you can report test accuracy
% by setting testSet=3. You should not optimize your hyperparameters on the
% test set. That would be cheating.
```