UNIVERSITY OF VICTORIA

Department of Electrical and Computer Engineering

ECE 503 Optimization for Machine Learning

PROJECT REPORT

Project Topic Option: Option 1 (Classify MNIST dataset)

Title: PCA of HOG applied to MNIST dataset

Date of Experiment: August 6th, 2020

Report Submitted on: August 12th, 2020

To: Dr. Wu-Sheng Lu

Name: Jude Onyia (V00947095)

1. Objective

This project involves the development of software to achieve handwritten digit recognition (HWDR) through machine learning. Principal component analysis (PCA) is used as a multi-class classification technique applied to the popular MNIST database of handwritten digits. A pre-processing technique is used to compute the histogram of oriented gradients (HOG) of the dataset, the HOG consists of explicit features that are typically hidden within an image. This project applies the PCA technique on the HOG of the dataset and compares the results with that of applying PCA on the dataset directly.

2. Introduction

MNIST handwritten digits database includes 60,000 samples intended to be used as training data and 10,000 samples as testing data [1]. For this project, 16,000 samples of the training data are used, this consists of 1,600 samples for each digit. The project uses the entire testing data in the MNIST database. The dataset is associated with ten classes for the ten digits that are distinguished [1]. The structure of the dataset is shown in equation 1, where $\boldsymbol{x}_p$ is a sample, y_p is its corresponding label, and L is the number of classes (which is 10 for this dataset) [1]. P is the number of samples for the training data and N is number of features in each sample. The training data is arranged according to equation 2, where each group of contiguous 1,600 samples form a class with all samples containing the same digit, hence, having the same label. Class j contains samples of written digit $j-1$. Each sample $\boldsymbol{x}_p$ is a 784×1 vector constructed from a 28×28 pixels gray-scale digital image of a handwritten digit by stacking the 28 columns of the image to form a single column [1].

$$\{(\boldsymbol{x}_p, y_p), \quad p = 1,2,\dots,P\} \tag{1}$$

$$where\ \boldsymbol{x}_p = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \in Class_j\ \ and\ \ y_p = j \qquad\qquad for\ j = 1,2,\dots,L$$

$$\boldsymbol{X}_{tr} = \underset{digit\ "0"}{[\{\boldsymbol{x}_1\ \boldsymbol{x}_2\ \cdots\ \boldsymbol{x}_{1600}\}}\ \underset{digit\ "1"}{\{\boldsymbol{x}_{1601}\ \boldsymbol{x}_{1602}\ \cdots\ \boldsymbol{x_{3200}}\}}\ \cdots\cdots\cdots\ \underset{digit\ "9"}{\{\boldsymbol{x}_{14401}\ \boldsymbol{x}_{14402}\ \cdots\ \boldsymbol{x_{16000}}\}]} \tag{2}$$

2.1. Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) is a feature descriptor with robust properties [2]. The HOG of an image requires dividing the image into overlapping blocks, this project uses 50% overlap between blocks. The overlap ensures that all pixels, except the ones near or at a corner of the image, are present in multiple blocks [2]. This makes the HOG insensitive to slight image distortions [2]. For an image $D(x, y)$, its oriented gradient can be obtained by taking the partial difference along the *x* and *y* directions [2]. The partial differences can be computed by convoluting the image with kernels that implement the partial differences [2]. Examples of such kernels are defined in equation 3, and the resulting partial differences $\{\nabla_x D, \nabla_y D\}$ defines the gradient. The magnitude $\|\nabla D(x, y)\|$ and angle $\theta(x, y)$ of the oriented gradient is defined in equation 4 and 5, respectively.

$$h_x = [-1 \quad 0 \quad 1], \qquad h_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{3}$$

$$\|\nabla D(x, y)\| = \sqrt{(\nabla_x D)^2 + \left(\nabla_y D\right)^2} \tag{4}$$

$$\theta(x, y) = tan^{-1}\left(\frac{\nabla_y D}{\nabla_x D}\right) \tag{5}$$

The HOG of an image is obtained by first dividing the angle range $(-\pi, \pi)$ evenly into b bins, where each bin is associated with an angle range of size $2\pi/b$. Then, each block is associated with a column vector $\boldsymbol{h}_{block\,i}$ initially set to zero with b components for the b number of bins. The kernels h_x and h_y can be used to scan each block. The scanning of each pixel within a block produces a pair of $\{\theta(x, y), \|\nabla D(x, y)\|\}$. Depending on which bin the angle $\theta(x, y)$ will fall into, the magnitude $\|\nabla D(x, y)\|$ is added to the corresponding bin or component of $\boldsymbol{h}_{block\,i}$. Each column vector $\boldsymbol{h}_{block\,i}$ now holds the HOG of its associated block. The HOG of the entire image $\boldsymbol{h}_{image}$ is obtained by stacking all $\boldsymbol{h}_{block\,i}$ vectors according to equation 6, where $i = 1,2, \ldots, S$ (S is the number of blocks in the image). The column vector $\boldsymbol{h}_{image}$ has $length = S \times b$.

$$\boldsymbol{h}_{image} = \begin{bmatrix} \boldsymbol{h}_{block\,1} \\ \boldsymbol{h}_{block\,2} \\ \vdots \\ \boldsymbol{h}_{block\,S} \end{bmatrix} \qquad where\ S\ is\ the\ number\ of\ blocks\ in\ the\ image \tag{6}$$

2.2. Principal Component Analysis

Principal component analysis (PCA) is typically used as a dimension reduction, feature extraction and data representation technique [2]. The technique requires computing the mean $\overline{\boldsymbol{x}}$ and covariance $\boldsymbol{C}$ of the training dataset as shown in equation 7 and 8, respectively.

$$\overline{\boldsymbol{x}} = \frac{1}{P}\sum_{p=1}^{P} \boldsymbol{x}_p \tag{7}$$

$$\boldsymbol{C} = \frac{1}{P-1}\sum_{p=1}^{P} (\boldsymbol{x}_p - \overline{\boldsymbol{x}})(\boldsymbol{x}_p - \overline{\boldsymbol{x}})^T \tag{8}$$

Eigen-decomposition can be performed on $\boldsymbol{C}$ as shown in equation 9. Where $\boldsymbol{T}$ is an orthogonal matrix with its column vectors $\boldsymbol{t}_i$ being the eigenvectors associated with eigenvalues λ_i for $i = 1,2,..,N$. An L_2-optimal rank-K approximation of the covariance matrix $\boldsymbol{C}$ can be obtained by keeping only the first K largest eigenvalues and their associated eigenvectors [2]. This is shown in equation 10.

$$\boldsymbol{C} = \boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^T \tag{9}$$

$$where \quad \boldsymbol{T} = [\boldsymbol{t}_1 \quad \boldsymbol{t}_2 \quad \dots \quad \boldsymbol{t}_N]$$

$$and\ \boldsymbol{\Lambda} = diag\{\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_N\} \qquad where\ \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$$

$$\boldsymbol{C} \approx \boldsymbol{T}_K\boldsymbol{\Lambda}_K\boldsymbol{T}_K^T \qquad with\ K < N \tag{10}$$

$$where \quad \boldsymbol{T}_K = [\boldsymbol{t}_1 \quad \boldsymbol{t}_2 \quad \dots \quad \boldsymbol{t}_K]$$

$$and\ \boldsymbol{\Lambda}_K = diag\{\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_K\} \qquad where\ \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_K \geq 0$$

The eigenvectors in $\boldsymbol{T}_K$ are referred to as the principal components of the dataset because they contain the most significant variations between individual mean-centered samples $(\boldsymbol{x}_p - \overline{\boldsymbol{x}})$ [2]. Therefore, equation 11 shows that $(\boldsymbol{x}_p - \overline{\boldsymbol{x}})$ can be represented by its projection $\boldsymbol{z}_p$ on the K-dimensional subspace spanned by the eigenvectors in $\boldsymbol{T}_K$. It then follows that $\boldsymbol{x}_p$ can be approximately recovered from $\boldsymbol{z}_p$ according to equation 12.

$$\boldsymbol{z}_p = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix} = \boldsymbol{T}_K^T(\boldsymbol{x}_p - \overline{\boldsymbol{x}}) \tag{11}$$

$$\boldsymbol{x}_p \approx \boldsymbol{T}_K\boldsymbol{z}_p + \overline{\boldsymbol{x}} \tag{12}$$

2.3. PCA solves an optimization problem

PCA requires solving the optimization problem in equation 13, considering that $\boldsymbol{x}_p$ is already mean-centered [3]. $\widetilde{\boldsymbol{T}}$ must be composed of orthonormal vectors defined in equation 14. The objective is to find a $\widetilde{\boldsymbol{T}}$ that minimizes $f(\widetilde{\boldsymbol{T}})$ [3].

$$f(\widetilde{\boldsymbol{T}}) = \frac{1}{P}\sum_{p=1}^{P}\left\|\widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}}^T\boldsymbol{x}_p - \boldsymbol{x}_p\right\|_2^2 \tag{13}$$

$$\widetilde{\boldsymbol{T}} = [\tilde{\boldsymbol{t}}_1 \quad \tilde{\boldsymbol{t}}_2 \quad \dots \quad \tilde{\boldsymbol{t}}_K] \qquad where\ \tilde{\boldsymbol{t}}_i^T\tilde{\boldsymbol{t}}_j = \begin{cases} 1 & if\ \ i = j \\ 0 & if\ \ i \neq j \end{cases} \quad for\ i,j = 1,2,\dots,K \tag{14}$$

Equation 13 can be written as equation 15, where $\boldsymbol{x}_p^T\widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}}^T\boldsymbol{x}_p$ reduces to a scalar, and $\frac{1}{P}\sum_{p=1}^{P}\left\|\boldsymbol{x}_p\right\|_2^2$ results in a constant. Using the properties of $trace$ in (16) and (17), equation 15 can be written as (18). A $trace$ of a matrix is the sum of all elements in the primary diagonal of the matrix [3].

$$f(\widetilde{\boldsymbol{T}}) \ = \ -\frac{1}{P}\sum_{p=1}^{P}\boldsymbol{x}_p^T\widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}}^T\boldsymbol{x}_p + \frac{1}{P}\sum_{p=1}^{P}\left\|\boldsymbol{x}_p\right\|_2^2 \tag{15}$$

$$trace(A) = A, \quad if\ A\ is\ a\ scalar \tag{16}$$

$$trace(AB) = trace(BA) \tag{17}$$

$$f(\widetilde{\boldsymbol{T}}) \ = \ -\frac{1}{P}\sum_{p=1}^{P} trace\left(\widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}}^T\boldsymbol{x}_p\boldsymbol{x}_p^T\right) + constant \tag{18}$$

Using the covariance equations in (8) and (9) shown earlier, equation 18 can be written as (19). From the property in (17), equation 19 can be written as (20). If $\boldsymbol{T}_K$ (obtained in equation 10) is used (where its column vectors $\boldsymbol{t}_i$ are the eigenvectors of the covariance of all $\boldsymbol{x}_p$ associated with eigenvalues λ_i for $i = 1,2,..,K$), then $f(\boldsymbol{T}_K)$ can be defined in equation 21.

$$f(\widetilde{\boldsymbol{T}}) \ = \ -trace\left(\widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}}^T\boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^T\right) + constant \tag{19}$$

$$f(\widetilde{\boldsymbol{T}}) \ = \ -trace\left(\widetilde{\boldsymbol{T}}^T\boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^T\widetilde{\boldsymbol{T}}\right) + constant \tag{20}$$

$$f(\boldsymbol{T}_K) \ = \ -trace(\boldsymbol{T}_K^T\boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^T\boldsymbol{T}_K) + constant \tag{21}$$

Since $\boldsymbol{T}$ is also composed of $\boldsymbol{t}_i$ but for $i = 1,2,..,N$ and $\boldsymbol{t}_i$ is orthonormal, equation 22 is used to express (21) as (23). This equates to (24), which can be expressed as (25). Since equation 25 involves the largest eigenvalues of the covariance of all $\boldsymbol{x}_p$, (25) shows the smallest possible combination of K numbers that reside in the function $f(\widetilde{\boldsymbol{T}})$ [3]. Therefore, $\boldsymbol{T}_K$ is a minimizer of $f(\widetilde{\boldsymbol{T}})$ [3].

$$\boldsymbol{T}_K^T\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \vdots \\ \boldsymbol{t}_K \end{bmatrix} [\boldsymbol{t}_1 \quad \boldsymbol{t}_2 \quad \dots \quad \boldsymbol{t}_N] = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{bmatrix} = [\boldsymbol{I}_K \quad \boldsymbol{0}] \tag{22}$$

$$f(\boldsymbol{T}_K) = -trace\left([\boldsymbol{I}_K \quad \boldsymbol{0}][diag\{\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_N\}]\begin{bmatrix} \boldsymbol{I}_K \\ \boldsymbol{0} \end{bmatrix}\right) + constant \tag{23}$$

$$f(\boldsymbol{T}_K) = -trace(diag\{\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_K\}) + constant \tag{24}$$

$$f(\boldsymbol{T}_K) = -(\lambda_1 + \lambda_2 + \cdots + \lambda_K) + constant \tag{25}$$

2.4. PCA of HOG

According to [2], PCA can also be used as a multi-class classifier. First, the training samples are sorted according to their respective classes as shown in equation 26. Where $\boldsymbol{X}_j$ denotes all training sample belonging to $Class_j$, and P_j denotes the number of samples in $Class_j$. For the dataset in this project, P_j is 1,600 for each class. Then, HOG pre-processing is applied to each $\boldsymbol{X}_j$, this results in obtaining $\boldsymbol{H}_j$ defined in equation 27, where each $\boldsymbol{h}_i^{(j)}$ for $i = 1,2,..P_j$ is acquired with the same process used to get $\boldsymbol{h}_{image}$ in equation 6.

$$\boldsymbol{X}_j = \left[\boldsymbol{x}_1^{(j)} \quad \boldsymbol{x}_2^{(j)} \quad \dots \quad \boldsymbol{x}_{P_j}^{(j)}\right] \quad for\ j = 1,2,\dots,L \tag{26}$$

$$\boldsymbol{H}_j = \left[\boldsymbol{h}_1^{(j)} \quad \boldsymbol{h}_2^{(j)} \quad \dots \quad \boldsymbol{h}_{P_j}^{(j)}\right] \quad for\ j = 1,2,\dots,L \tag{27}$$

Using the HOG $\boldsymbol{H}_j$ of the samples, PCA is applied to each class. The mean $\overline{\boldsymbol{h}}_j$ is obtained according to equation 28, and the covariance $\boldsymbol{C}_j$ is obtained as shown in equation 29. $\boldsymbol{T}_K^{(j)}$ is obtained with a similar formula to equation 10 but acquired from $\boldsymbol{C}_j$.

$$\overline{\boldsymbol{h}}_j = \frac{1}{P_j}\sum_{i=1}^{P_j} \boldsymbol{h}_i^{(j)} \tag{28}$$

$$\boldsymbol{C}_j = \frac{1}{P_j - 1}\sum_{i=1}^{P_j} \left(\boldsymbol{h}_i^{(j)} - \overline{\boldsymbol{h}}_j\right)\left(\boldsymbol{h}_i^{(j)} - \overline{\boldsymbol{h}}_j\right)^T \tag{29}$$

To classify a test sample $\boldsymbol{x}$, its HOG $\boldsymbol{h}$ is first obtained, then (30), (31) and (32) are performed. The sample $\boldsymbol{x}$ can be classified to class j^* if E_{j^*} is the minimum value amongst all E_j for $j = 1,2,\dots,L$.

$$\boldsymbol{z}_j = \boldsymbol{T}_K^{(j)T}\left(\boldsymbol{h} - \overline{\boldsymbol{h}}_j\right) \tag{30}$$

$$\widehat{\boldsymbol{h}}_j = \boldsymbol{T}_K^{(j)}\boldsymbol{z}_j + \overline{\boldsymbol{h}}_j \tag{31}$$

$$E_j = \left\|\boldsymbol{h} - \widehat{\boldsymbol{h}}_j\right\|_2 \tag{32}$$

3. Implementation and Results

3.1. Implementation

The MATLAB implementation performed for this project involved the following steps (the MATLAB code can be found in the appendix):

1. Acquire the training data $\boldsymbol{X}_{tr}$ and testing data $\boldsymbol{X}_{te}$.
2. Acquire the labels of the testing data $\boldsymbol{y}_{te}$.
3. Since $\boldsymbol{X}_{tr}$ is arranged according to equation 2, create its labels $\boldsymbol{y}_{tr}$ by taking the value of the digit represented by each sample and adding one.
4. Create $\boldsymbol{D}_{tr}$ by stacking $\boldsymbol{y}_{tr}$ underneath $\boldsymbol{X}_{tr}$.
5. Create $\boldsymbol{D}_{te}$ by stacking $\boldsymbol{y}_{te}$ underneath $\boldsymbol{X}_{te}$.
6. Set K=23, where K is the number of eigenvectors considered as the principal component. This value for K was found to achieve the best performance in the experiment conducted in [2].
7. Apply PCA on $\boldsymbol{D}_{tr}$ to acquire $\boldsymbol{T}_K^{(j)}$ for $j = 1,2,\dots,L$. Where L=10.
8. Use $\boldsymbol{T}_K^{(j)}$ to classify $\boldsymbol{D}_{te}$, and report the confusion matrix, classification accuracy and speed of recognition.
9. Apply the HOG feature extraction technique to $\boldsymbol{D}_{tr}$ and $\boldsymbol{D}_{te}$ with a stride of 50%, a block size of 4×4, and 9 bins associated to each block. This results in acquiring $\boldsymbol{D}_{ht}$ and $\boldsymbol{D}_{ht}$.
10. Apply PCA on $\boldsymbol{D}_{ht}$ to acquire $\boldsymbol{T}_K^{(j)}$ for $j = 1,2,\dots,L$. Where L=10.
11. Use $\boldsymbol{T}_K^{(j)}$ to classify $\boldsymbol{D}_{hte}$, and report the confusion matrix, classification accuracy and speed of recognition.

3.2. Results

PCA was first applied to the original dataset of handwritten digits from the MNIST database. The original training data $\boldsymbol{D}_{tr}$ of size 785×16,000 was used to acquire a set of $\boldsymbol{T}_K^{(j)}$ for $j = 1,2,\ldots,10$. To evaluate the performance of this HWDR machine learning model, the set of $\boldsymbol{T}_K^{(j)}$ was used to classify the original testing data $\boldsymbol{D}_{te}$ of size 785×10,000. A confusion matrix, shown in **Table I**, was formed by comparing the expected label $\boldsymbol{y}_{te}$ to the classification made by the model. **Table I** also shows the classification accuracy and the speed of recognition. The speed of recognition was calculated by measuring the time required to classify the testing data. This is important because if the model were to be deployed online for public use, the speed at which it produces a result is a crucial aspect of its performance.

Table I: Confusion matrix, accuracy and speed of recognition of PCA applied to original dataset

	Actual Label										
Predicted Label		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$	$j = 8$	$j = 9$	$j = 10$
	$j = 1$	969	0	12	4	1	5	4	2	7	7
	$j = 2$	1	1126	4	0	6	1	3	9	5	6
	$j = 3$	1	4	966	8	2	1	1	16	4	2
	$j = 4$	0	2	6	952	0	19	0	0	13	8
	$j = 5$	0	0	4	0	952	0	3	5	2	10
	$j = 6$	0	1	0	11	0	840	7	2	8	6
	$j = 7$	5	2	2	1	6	10	939	1	3	0
	$j = 8$	1	0	13	10	4	1	0	972	7	14
	$j = 9$	3	0	23	20	2	12	1	2	913	9
	$j = 10$	0	0	2	4	9	3	0	19	12	947
Classification accuracy = 95.76%											
Speed of recognition = 3350 digits per second											

The HOG feature extracting technique was then performed on the original training data to acquire $\boldsymbol{D}_{htr}$ of size 1,522×16,000. PCA was applied to $\boldsymbol{D}_{htr}$ to acquire a set of $\boldsymbol{T}_K^{(j)}$, and the set was used to classify the HOG of the testing data $\boldsymbol{D}_{hte}$ of size 1,522×10,000. Although, the HOG

of the training and testing data are much larger than their corresponding raw data, the performance gained is significant as shown in **Table II**. Using the HOG of the dataset improved the classification accuracy by 2.54%, which is significant in machine learning. However, there is a drawback, computing the HOG adds latency to each digit classification. Calculating the HOG for each sample reduced the speed of recognition by 86.09% of the speed obtained by the model using the original data.

Table II: Confusion matrix, accuracy and speed of recognition of PCA applied to HOG of dataset

	Actual Label										
Predicted Label		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$	$j = 8$	$j = 9$	$j = 10$
	$j = 1$	975	0	4	0	0	2	5	0	1	5
	$j = 2$	0	1130	0	0	0	0	3	5	0	6
	$j = 3$	1	1	1017	1	0	1	0	9	4	1
	$j = 4$	0	2	0	996	0	7	0	0	6	6
	$j = 5$	0	1	0	0	963	0	1	0	1	3
	$j = 6$	0	0	0	1	0	875	5	0	1	3
	$j = 7$	2	0	0	0	6	3	943	0	2	0
	$j = 8$	1	1	7	6	0	0	0	1006	4	4
	$j = 9$	1	0	4	4	2	2	1	1	952	8
	$j = 10$	0	0	0	2	11	2	0	7	3	973
Classification accuracy = 98.30%											
Speed of recognition = 466 digits per second											

4. Discussion

Although computing the HOG for each sample is expensive, the speed of recognition of 466 digits per second achieved is decent according to the results obtained in [3]. The classification accuracy obtained from using PCA on HOG is significantly higher than that obtain from using PCA on the original dataset. This validates the benefit of computing the HOG.

The results obtained in this project also shows that PCA works well as a multi-class classifier. Though better accuracies could be achieved with other techniques, PCA provides a fast option that yields sufficient results.

5. Conclusion

Research for HWDR through machine learning is still an active area in need of improvement [1]. It provides many applications including postal mail sorting, mail routing, automatic address reading, and bank check processing [2]. Due to the wide variety of shapes, line width, and style found in writing a single digit, HWDR is challenging [2]. Also, the fact that certain styles make different digits appear similar adds to the complexity of distinguishing digits. This project examines a decent solution that utilizes PCA as a multi-class classifier. PCA is typically used as a dimension reduction, feature extraction and data representation technique [2]. The works performed in [2] shows a method of using PCA on each class independently to achieve a codebook $\boldsymbol{T}_K^{(j)}$ for each class. To classify a new sample, it is encoded and decoded using $\boldsymbol{T}_K^{(j)}$ from each class. Whichever class is associated with the $\boldsymbol{T}_K^{(j)}$ that most closely recovered the sample after decoding is the class that the sample belongs to. This technique was then applied to the HOG of the data set instead of the raw data. The results of the work performed in this project shows that using the HOG improves the classification accuracy but decreases the speed of recognition.

6. References

[1] Laboratory Manual for ECE 403/503 Optimization for Machine Learning. W.-S. Lu, University of Victoria Department of Electrical and Computer Engineering. May 2020.

[2] W.-S. Lu, “Handwritten digits recognition using PCA of histogram of oriented gradient,” Proc. PacRim Conf., Victoria, BC, Canada, August 2017.

[3] ECE 403/503 Optimization for Machine Learning. Course Notes. W.-S. Lu, University of Victoria Department of Electrical and Computer Engineering. May 2020.

7. Appendix

The MATLAB code required for this project consists of six scripts:

- The main script that loads the training and test data, calls the PCA training and classifying functions, calls the pre-processing function, and calls the function to evaluate performance.
- A function that performs PCA on each class in the training data independently and produces the set of $\boldsymbol{T}_K^{(j)}$ for each class.
- A function that classifies new sample using the set of $\boldsymbol{T}_K^{(j)}$.
- A pre-processing function that reshapes each 784×1 vector sample into a 28×28 matrix and calls the HOG function efficiently with histogram bins set to 9 and subblock length and width set to 4.
- A function that performs the HOG feature extraction on each sample.
- A function that computes confusion matrices and classification accuracies.

7.1. Main script

```
clear
clc
load('X1600.mat')
load('Lte28.mat')
load('Te28.mat')
u = ones(1,1600);
ytr = [u 2*u 3*u 4*u 5*u 6*u 7*u 8*u 9*u 10*u];
Dtr = [X1600; ytr]; % Original training samples
ytest = 1 + Lte28(:)'; % Turn MNIST labels to the labels
used in this esperiment
Dte = [Te28; ytest]; % Original testing samples
[N,P_tr] = size(X1600);
P_te = size(Te28,2);
L = 10;
K = 23;

% PCA on original dataset
[CK, meanK] = PCA_training(Dtr,N,P_tr,L,K); % Train
classifier using PCA and Original Dataset
tt = cputime; % Used to measure speed of recognition
ind_pre = PCA_classifying(Te28,CK,meanK,L,P_te); %
Classify Testing Dataset
tt = cputime - tt;
speed_of_recognition = floor(inv(tt/P_te)); % Speed of
recognition in samples per second
```

```
[C,accuracy] =
confusion_matrix_and_accuracy(ind_pre,ytest,L,P_te); %
Confusion matrix and accuracy

% PCA on HOG of dataset
Dhtr = pre_processing_HOG(Dtr,N,P_tr); % Obtaining the HOG
of training data
N_hog = size(Dhtr,1) - 1;
[CK_hog, meanK_hog] = PCA_training(Dhtr,N_hog,P_tr,L,K); %
Train classifier using PCA and HOG of Dataset
tt_hog = cputime; % Used to measure speed of recognition
Dhte = pre_processing_HOG(Dte,N,P_te); % Obtaining the HOG
of testing data
ind_pre_hog =
PCA_classifying(Dhte(1:N_hog,:),CK_hog,meanK_hog,L,P_te);
% Classify Testing Dataset
tt_hog = cputime - tt_hog;
speed_of_recognition_hog = floor(inv(tt_hog/P_te)); %
Speed of recognition in samples per second
[C_hog,accuracy_hog] =
confusion_matrix_and_accuracy(ind_pre_hog,ytest,L,P_te); %
Confusion matrix and accuracy
```

7.2. PCA Training function

```
% Using Principal Component Analysis for Classification
function [CK, meanK] = PCA_training(Dtr,N,P_tr,L,K)
P_tr_j = P_tr / L;
for j=1:L
    Xtr_j = Dtr(1:N,((j-1)*P_tr_j+1):(j*P_tr_j)); % Get
all samples belonging to each class
    mean_j = (mean(Xtr_j'))'; % Find the mean for all
samples of each dimension
    Xtr_j = Xtr_j - mean_j; % Make all samples mean-
centered
    CV_j = (Xtr_j*Xtr_j') / P_tr_j; % Covariance of mean-
centered samples of each dimension
    [CK_j,S] = eigs(CV_j,K); % Capture principal
components
    CK(:,:,j) = CK_j;
    meanK(:,j) = mean_j;
end
end
```

7.3. PCA Classifying function

```
 % Using the Principal Components acquired from
PCA_training function
% to classify a number of test samples
function ind_pre = PCA_classifying(Xte,CK,meanK,L,P_te)
E_jp = zeros(L,P_te);
for j=1:L
    CK_j = CK(:,:,j);
    meanK_j = meanK(:,j);
    Z_j = CK_j' * (Xte - meanK_j); % Compute the
projection
    Xte_est = CK_j*Z_j + meanK_j; % Approximation of test
data
    % Euclidean distance between actual and approximate
test data
    for p=1:P_te
        E_jp(j,p) = norm(Xte(:,p) - Xte_est(:,p));
    end
end

ind_pre = zeros(1,P_te);
for p=1:P_te
    ind_pre(p) = find( E_jp(:,p) == min(E_jp(:,p)) );
end
end
```

7.4. Pre-processing function

```
 % HOG Training samples
function Dhtr = pre_processing_HOG(Dtr,N,P_tr)
Xtr = Dtr(1:N,:);
ytr = Dtr(N+1,:);
H = [];
for i = 1:P_tr
    xi = Xtr(:,i);
    mi = reshape(xi,28,28);
    hi = hog20(mi,4,9);
    H = [H hi];
end
Dhtr = [H; ytr];
```

7.5. HOG function

```
 % Image descriptor based on Histogram of Orientated
Gradients (HOG) for
% gray-level images. This code was developed for the work:
% O. Ludwig, D. Delgado, V. Goncalves, and U. Nunes,
"Trainable Classifier-Fusion
% Schemes: An Application To Pedestrian Detection," in
12th Int. IEEE Conf.
% Intelligent Transportation Systems, 2009, St. Louis,
2009. vol.1, pp. 432-437.
% In case of publication with this code, please cite the
paper above.
% Major revision was done by W.-S. Lu at University of
Victoria.
% Last modified: June 15, 2020.
% Input:
% Im: input image.
% d: d x d is the size of basic image block to perform
HOG.
% Typically d is in the range from 3 to 7.
% B: number of histogram bins. Typically B is set to 7 or
9.
% Output:
% h: HOG feature vector of input image.
function h = hog20(Im,d,B)
t = floor(d/2); % Stride t=7(no overlap) t=1 (large
overlap)
[N,M] = size(Im);
k1 = (M-d)/t;
c1 = ceil(k1);
k2 = (N-d)/t;
c2 = ceil(k2);
if c1 - k1 > 0
   M1 = d + t*c1;
   Im = [Im fliplr(Im(:,(2*M-M1+1):M))];
end
if c2 - k2 > 0
   N1 = d + t*c2;
   Im = [Im; flipud(Im((2*N-N1+1):N,:))];
end
[N,M] = size(Im);
nx1 = 1:t:M-d+1;
nx2 = d:t:M;
```

```
ny1 = 1:t:N-d+1;
ny2 = d:t:N;
Lx = length(nx1);
Ly = length(ny1);
hz = Lx*Ly*B;
H = zeros(hz,1);
Im = double(Im);
k = 1;
hx = [-1 0 1];
% hx = [-1 0 1; -2 0 2; -1 0 1]; % Sobel operator in
horizontal direction
hy = -hx';
grad_xr = imfilter(Im,hx);        % gradient in horizontal
direction
grad_yu = imfilter(Im,hy);        % gradient in vertical
direction
magnit = sqrt(((grad_yu.^2) + (grad_xr.^2)));
angles = atan2(grad_yu, grad_xr);
for m = 1:Lx
    for n = 1:Ly
        angles2 = angles(ny1(n):ny2(n),nx1(m):nx2(m));
        magnit2 = magnit(ny1(n):ny2(n),nx1(m):nx2(m));
        v_angles = angles2(:);
        v_magnit = magnit2(:);
        K = length(v_angles);
        % assembling the histogram with B bins (range of
360/B degrees per bin)
        bin = 1;
        h2 = zeros(B,1);
        for ang_lim = -pi+2*pi/B:2*pi/B:pi
            for i = 1:K
                if v_angles(i) < ang_lim
                    v_angles(i) = 100;
                    h2(bin) = h2(bin) + v_magnit(i);
                end
            end
            bin = bin + 1;
        end
        h2 = h2/(norm(h2)+0.01);
        h((k-1)*B+1:k*B,1) = h2;
        k = k + 1;
    end
end
```

7.6. Confusion Matrix and Classification Accuracy function

```
 % Compute confusion matrix and accuracy
function [C,accuracy] =
confusion_matrix_and_accuracy(ind_pre,ytest,L,P_te)
% Confusion matrix
C = zeros(L,L);
for j = 1:L
    ind_j = find(ytest == j);
    for i = 1:L
        ind_pre_i = find(ind_pre == i);
        C(i,j) = length(intersect(ind_j,ind_pre_i));
    end
end
% Accuracy
accuracy = 0;
for i = 1:L
    accuracy = accuracy + C(i,i);
end
accuracy = (accuracy / P_te) * 100;
end
```