Computer Vision

Programming Assignment 4 Report

Ulku Meteriz

# Part 1 – Fisher LDF

In this part, we are asked to implement the Fisher Linear Discriminant Function for CIFAR-10 dataset. For the classification computations, the below steps are followed:

1. The mean of each class is computed.
2. The covariance matrix of each class is computed.
3. The overall mean is computed.
4. The between class scatter matrix, B, is computed.
5. The sum of the class covariance matrices, A, is computed.
6. The eigenvalues and eigenvectors of A-1B is found.
7. The matrix H is created from the found eigenvectors.
8. The class means and the covariance matrices are transformed into the Fisher LDF space.
9. Each test instance is converted to Fisher LDF space.
10. The Mahalanobis distance between each sample and each class is computed.
11. The sample is assigned to the class where the Mahalanobis distance is the smallest.

## Results

### Confusion Matrix

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 7 | 4 | 11 | 2 | 18 | 14 | 22 | 6 | 6 | 13 |
| **1** | 8 | 4 | 9 | 3 | 15 | 14 | 13 | 4 | 2 | 17 |
| **2** | 15 | 4 | 8 | 2 | 15 | 14 | 13 | 3 | 3 | 23 |
| **3** | 14 | 3 | 5 | 2 | 18 | 13 | 24 | 2 | 4 | 18 |
| **4** | 9 | 4 | 15 | 2 | 18 | 7 | 10 | 5 | 7 | 13 |
| **5** | 13 | 9 | 4 | 0 | 19 | 6 | 15 | 4 | 5 | 11 |
| **6** | 12 | 7 | 8 | 2 | 24 | 9 | 15 | 10 | 9 | 16 |
| **7** | 15 | 3 | 15 | 4 | 16 | 12 | 16 | 10 | 5 | 6 |
| **8** | 8 | 3 | 18 | 6 | 14 | 21 | 19 | 6 | 1 | 10 |
| **9** | 13 | 7 | 11 | 2 | 22 | 12 | 15 | 7 | 3 | 17 |

### Accuracy

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0.797 | 0.871 | 0.812 | 0.876 | 0.767 | 0.804 | 0.756 | 0.861 | 0.851 | 0.781 |

### True positive rate

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0.0679 | 0.0449 | 0.0800 | 0.0194 | 0.2000 | 0.0697 | 0.1339 | 0.0980 | 0.0094 | 0.1559 |

### True negative rate

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0.8807 | 0.9517 | 0.8933 | 0.9743 | 0.8230 | 0.8730 | 0.8344 | 0.9476 | 0.9507 | 0.8574 |

### False positive rate

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0.1192 | 0.0482 | 0.1066 | 0.0256 | 0.1769 | 0.1269 | 0.1655 | 0.0523 | 0.0492 | 0.1425 |

### False negative rate

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0.9320 | 0.9550 | 0.9200 | 0.9805 | 0.8000 | 0.9302 | 0.8660 | 0.9019 | 0.9905 | 0.8440 |

# Part 2 – CNN

For this part, I used PyTorch and Torchvision libraries.