Weather Image Classification using ML

Nisarg Upadhyaya
19CS30031

Data Analytics Project [DA-17]
Academic Year 2023-24

1. Introduction

Weather image classification involves predicting the weather given an image. The input is an image which can be considered a N*M*K matrix where N=height, M=width and K=channels (3 for RGB, 1 for grayscale). The output is a label which takes value from a fixed set of weather labels. This is a supervised classification task and several traditional ML approaches can help us tackle this problem. We will be using Bayesian Classifier, Decision Tree Classifier and SVM for this project.

2. Dataset

The dataset is obtained from Kaggle. It has a total of 6862 images from 11 different weather classes. However, 2 of them, namely `snow/1187.jpg` and `fogsmog/4514.jpg`, were animated images and hence not compatible. The distribution of the other 6860 images in the 11 different classes are in the table below. Each class has been assigned an index number which is used as its identifier during the classification instead of the label. As the images were of different sizes first they were all standardized to 200*200*3. The feature extraction and model fitting was done on this standardized dataset.

Class Label	Class Index	Images
lightning	0	377
snow	1	620
rain	2	526
fogsmog	3	850
rainbow	4	232
sandstorm	5	692
rime	6	1160
frost	7	475
hail	8	591
dew	9	698
glaze	10	639

3. Feature Extraction

If we directly flatten each image then it will give us a feature vector of length 200*200*3 = 120000. This is an enormous number of features and it is not possible to train the models on such a large feature vector and expect good results. One more reason the models won't perform well is the flattened feature vector is not a good representation of the underlying image. We need to extract the features systematically and obtain good representations of the underlying image while also reducing the dimension of the feature vector so that we can train the model easily. For this we use the following 2 techniques:

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) for image feature extraction involves transforming high-dimensional image data into a reduced set of uncorrelated components (principal components) capturing the most significant variations. It identifies patterns by projecting data onto new axes, ordered by variance. In images, PCA condenses pixel information into fewer components, preserving essential features while reducing dimensionality.

Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients (HOG) is an image processing technique that computes and represents the distribution of local gradient orientations in an image. It involves dividing the image into small cells, calculating gradients' magnitudes and directions within these cells, and creating histograms of gradient orientations. These histograms summarize the gradient information, emphasizing edges and texture patterns. By aggregating local gradients into histograms, HOG captures shape and texture details while being robust to changes in lighting and contrast.

We use the above two techniques and carry out feature extraction using the following four methods which we call M1-M4 respectively:

1. ***M1:*** Convert all images to grayscale. This brings down the dimension to 200*200. Next fit a PCA model using the training images with 65 components. Use this model to transform the training, validation and test images. This brings down the dimension to 65.

2. ***M2:*** With M1 we lose the color data. Hence, now we separate the RGB channels into to obtain 3 arrays each having an image with dimensions 200*200 corresponding to one of the three color channels. Next we fit 3 PCA models using the training images with 65 components in each of the 3 channels. Use this model to transform the training, validation and test images for each channel. This brings down the dimension to 65*3.
3. ***M3:*** This is nothing but ***M2*** with standardized input parameters, i.e., scaled to have mean zero and unit variance.
4. ***M4:*** This is ***M3*** along with an extra set of HOG features appended for each image. For this we find the HOG features for each image. Next we fit a PCA model using the HOG features of the training images with 100 components. Use this model to transform the HOG features for training, validation and test images. This brings the down the dimension to 65*3 (PCA for each color channel) + 100 (PCA for HOG features).

The reason behind choosing 65 PCA components for the color channels and 100 PCA components for the HOG features is that by selecting these many components we are roughly able to explain 80% of the variance.

4. Classifiers

Once we have extracted the relevant features, we train the model using the following approaches:

a. Naive Bayes Classifier
b. Support Vector Machines
c. Decision Trees

For each classifier we split the dataset into 80-10-10 train-validation-test data and use the train and validation splits to find the optimal set of hyperparameters (using `sklearn.model_selection.GridSearchCV`). Finally we measure the performance of the model using the test data. We also use different combinations of feature extraction techniques ***(M1-M4)*** and find the best one for each classifier.

5. Naive Bayes Classifier

Scikit-learn model: `sklearn.naive_bayes.GaussianNB`
Hyperparameters tuned:

1. `var_smoothing`: Portion of the largest variance of all features that is added to variances for calculation stability

Method M1

Best parameters: `{'var_smoothing': 0.0001}`
Accuracy: `0.30903790087463556`
AUC-ROC score: `0.7516748927246583`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.45	0.44	0.45	43
1	0.56	0.25	0.35	71
2	0.29	0.21	0.24	52
3	0.34	0.71	0.46	72
4	0.23	0.24	0.23	29
5	0.20	0.23	0.22	77
6	0.33	0.38	0.35	117
7	0.18	0.13	0.15	38
8	0.31	0.40	0.35	58
9	0.19	0.11	0.14	74
10	0.35	0.13	0.19	55
accuracy			0.31	686
macro avg	0.31	0.29	0.28	686
weighted avg	0.32	0.31	0.29	686

Confusion matrix:

```
[[19  0  2  7  3  6  2  0  1  3  0]
 [ 0 18  8  4  2  6 17  3 10  2  1]
 [ 3  4 11  1  3  4  8  2 11  3  2]
 [ 2  0  0 51  1  8  7  0  0  2  1]
 [ 0  0  1 11  7  4  3  0  2  1  0]
 [ 6  0  0 42  2 18  8  0  0  1  0]
 [ 1  6  8 20  6 17 45  1  6  5  2]
 [ 1  1  1  1  1  3  9  5  6  7  3]
 [ 3  2  2  0  3  5 12  1 23  6  1]
 [ 3  1  4 13  2  9 15  8  8  8  3]
 [ 4  0  1  0  1  9 12  8  8  5  7]]
```

Method M2

Best parameters: `{'var_smoothing': 1e-11}`

Accuracy: `0.34402332361516036`

AUC-ROC score: `0.7719840111570373`

Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.36	0.51	0.42	43
1	0.50	0.27	0.35	71
2	0.27	0.21	0.24	52
3	0.41	0.71	0.52	72
4	0.25	0.34	0.29	29
5	0.21	0.27	0.24	77
6	0.41	0.38	0.39	117
7	0.15	0.11	0.12	38
8	0.32	0.43	0.37	58
9	0.35	0.24	0.29	74
10	0.46	0.20	0.28	55
accuracy			0.34	686
macro avg	0.34	0.33	0.32	686
weighted avg	0.35	0.34	0.33	686

Confusion matrix:

```
[[22  0  3  5  3  6  2  0  0  2  0]
 [ 0 19  8  3  4  6 14  3 10  2  2]
 [ 4  6 11  1  1  5  6  2 12  3  1]
 [ 3  0  0 51  3  5  9  0  0  1  0]
 [ 2  0  1  6 10  5  2  0  2  1  0]
 [ 6  0  0 38  6 21  5  0  0  1  0]
 [ 2  6 12 11  6 24 44  1  5  4  2]
 [ 3  3  1  1  0  2  5  4  8  8  3]
 [ 6  1  1  0  2  3  9  2 25  8  1]
 [ 8  3  2  7  4 13  2  7  6 18  4]
 [ 5  0  2  0  1  8  9  7  9  3 11]]
```

Method M3

Best parameters: `{'var_smoothing': 0.01}`

Accuracy: `0.34402332361516036`

AUC-ROC score: `0.7702483618254239`

Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.37	0.51	0.43	43
1	0.50	0.25	0.34	71
2	0.28	0.21	0.24	52
3	0.40	0.74	0.52	72
4	0.24	0.34	0.29	29
5	0.22	0.27	0.25	77
6	0.40	0.37	0.38	117
7	0.15	0.11	0.12	38
8	0.32	0.43	0.37	58
9	0.35	0.24	0.29	74
10	0.44	0.20	0.28	55
accuracy			0.34	686
macro avg	0.34	0.33	0.32	686
weighted avg	0.35	0.34	0.33	686

Confusion matrix:

```
[[22  0  3  5  3  6  2  0  0  2  0]
 [ 0 18  8  3  2  8 16  3 10  1  2]
 [ 3  6 11  1  2  5  6  2 12  3  1]
 [ 3  0  0 53  3  4  8  0  0  1  0]
 [ 2  0  1  8 10  3  2  0  2  1  0]
 [ 6  0  0 38  6 21  5  0  0  1  0]
 [ 1  5 11 15  7 21 43  1  5  5  3]
 [ 3  3  1  1  1  2  4  4  8  8  3]
 [ 6  1  1  0  2  3  9  2 25  8  1]
 [ 8  3  2  7  4 13  2  7  6 18  4]
 [ 5  0  1  0  1  8 10  7  9  3 11]]
```

Method M4

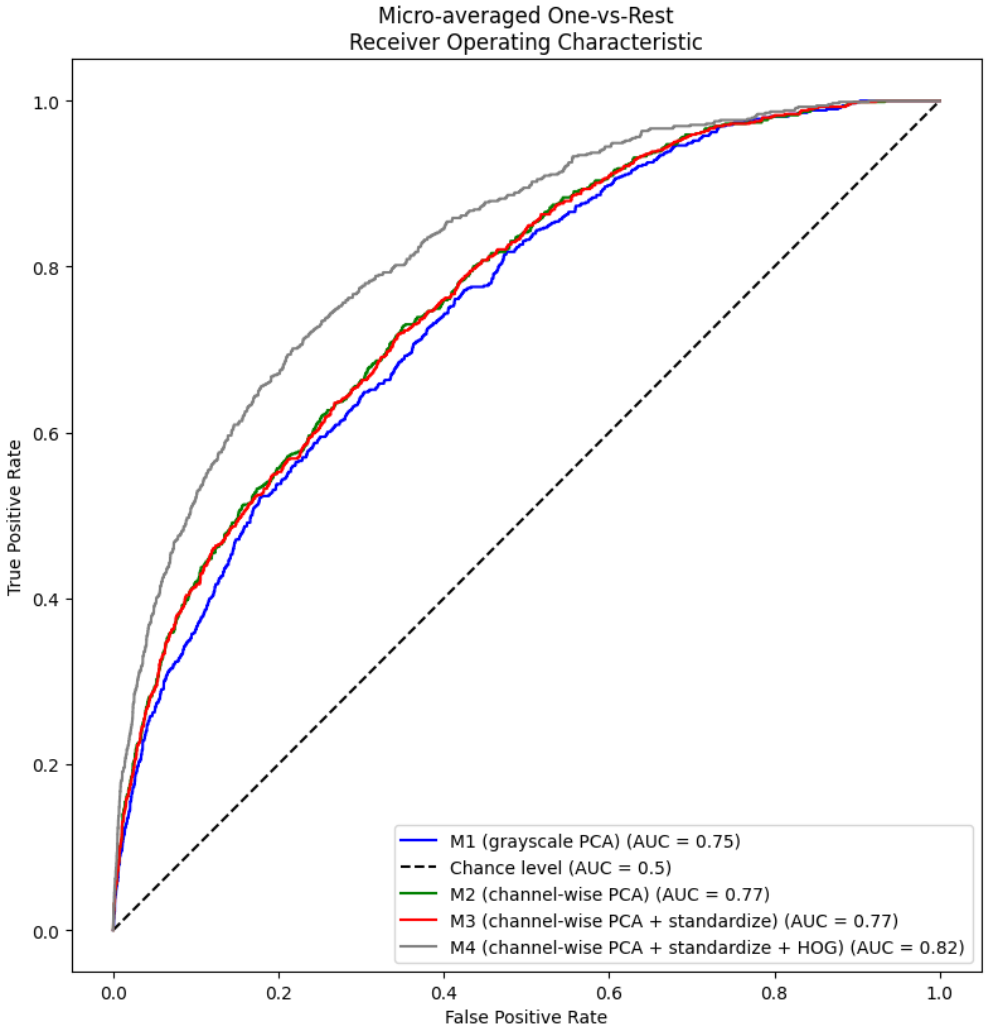
Best parameters: `{'var_smoothing': 1e-11}`
Accuracy: `0.4198250728862974`
AUC-ROC score: `0.825675152251803`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.65	0.56	0.60	43
1	0.51	0.28	0.36	71
2	0.51	0.52	0.51	52
3	0.43	0.71	0.54	72
4	0.48	0.52	0.50	29
5	0.22	0.30	0.25	77
6	0.40	0.48	0.44	117
7	0.30	0.42	0.35	38
8	0.42	0.47	0.44	58
9	0.56	0.24	0.34	74
10	0.69	0.20	0.31	55
accuracy			0.42	686
macro avg	0.47	0.43	0.42	686
weighted avg	0.46	0.42	0.41	686

Confusion matrix:

```
[[24  0  2  5  3  5  2  0  0  2  0]
 [ 0 20  9  3  1  6 17  5  9  0  1]
 [ 0  2 27  1  0  5  7  1  9  0  0]
 [ 3  0  0 51  2  9  3  0  0  4  0]
 [ 1  1  0  6 15  3  1  0  0  2  0]
 [ 4  1  0 36  7 23  6  0  0  0  0]
 [ 0  6  7 11  1 23 56  7  5  0  1]
 [ 0  2  2  1  0  2 10 16  5  0  0]
 [ 1  0  3  0  0  3 15  8 27  1  0]
 [ 4  5  2  4  2 19  7  6  4 18  3]
 [ 0  2  1  0  0  6 15 10  5  5 11]]
```

ROCs for Naive Bayes Classifier with different feature sets



Observations

1. Method M2 with color information in the features performs better than M1 which is grayscale PCA.
2. Method M2 and M3 perform nearly the same because Gaussian Naive Bayes is not affected due to standardization. For more information on this check the answer at this link: https://stats.stackexchange.com/questions/254723/standardisation-in-naive-bayes
3. Method M4 performs the best as it has a rich feature set representation with the HOG features also present.

6. Support Vector Machine

Scikit-learn model: `sklearn.svm.SVC`
Hyperparameters tuned:

1. `C`: Regularization parameter. The strength of the regularization is inversely proportional to C
2. `gamma`: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
3. `kernel`: Specifies the kernel type to be used in the algorithm

Method M1

Best parameters: `{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}`
Accuracy: `0.49271137026239065`
AUC-ROC score: `0.8621139086786171`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.64	0.65	0.64	43
1	0.50	0.45	0.47	71
2	0.40	0.44	0.42	52
3	0.45	0.75	0.57	72
4	0.55	0.38	0.45	29
5	0.52	0.38	0.44	77
6	0.51	0.59	0.55	117
7	0.31	0.29	0.30	38
8	0.54	0.43	0.48	58
9	0.49	0.32	0.39	74
10	0.52	0.58	0.55	55
accuracy			0.49	686
macro avg	0.49	0.48	0.48	686
weighted avg	0.50	0.49	0.49	686

Confusion matrix:

```
[[28  0  3  0  0  3  0  0  2  6  1]
 [ 2 32  7  3  0  1 18  0  4  2  2]
 [ 2  8 23  2  0  1  5  5  4  1  1]
 [ 1  0  1 54  0  9  1  0  1  3  2]
 [ 0  2  2  4 11  3  6  0  1  0  0]
 [ 2  2  2 33  1 29  5  1  0  0  2]
 [ 1  7  6  7  6  5 69  3  3  3  7]
 [ 1  3  2  4  0  0  7 11  2  5  3]
 [ 2  1  4  0  1  1  9  5 25  3  7]
 [ 4  8  3  7  1  1  9  9  4 24  4]
 [ 1  1  4  5  0  3  5  2  0  2 32]]
```

Method M2

Best parameters: `{'C': 0.1, 'gamma': 1, 'kernel': 'poly'}`
Accuracy: `0.5495626822157434`
AUC-ROC score: `0.8459120798468723`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.58	0.70	0.63	43
1	0.47	0.39	0.43	71
2	0.48	0.40	0.44	52
3	0.46	0.75	0.57	72
4	0.41	0.52	0.45	29
5	0.67	0.75	0.71	77
6	0.63	0.57	0.60	117
7	0.18	0.21	0.19	38
8	0.56	0.38	0.45	58
9	0.83	0.68	0.75	74
10	0.59	0.44	0.50	55
accuracy			0.55	686
macro avg	0.53	0.53	0.52	686
weighted avg	0.56	0.55	0.55	686

Confusion matrix:

```
[[30  1  0  2  3  0  2  4  0  0  1]
 [ 6 28  7  7  0  1 15  3  2  0  2]
 [ 1  6 21  4  2  3  5  4  2  2  2]
 [ 1  3  0 54  0  4  4  3  2  0  1]
 [ 3  0  0  6 15  1  1  1  1  0  1]
 [ 1  0  1 10  0 58  1  2  2  1  1]
 [ 5 11  3 10 11  2 67  0  2  0  6]
 [ 1  3  2  9  0  8  2  8  3  2  0]
 [ 3  2  3  6  2  3  5  6 22  4  2]
 [ 0  2  2  3  2  5  0  6  3 50  1]
 [ 1  3  5  6  2  1  4  8  0  1 24]]
```

Method M3

Best parameters: `{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}`
Accuracy: `0.5889212827988338`
AUC-ROC score: `0.8999414228086121`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.73	0.77	0.75	43
1	0.51	0.49	0.50	71
2	0.44	0.44	0.44	52
3	0.55	0.82	0.66	72
4	0.85	0.38	0.52	29
5	0.85	0.66	0.74	77
6	0.61	0.73	0.66	117
7	0.24	0.18	0.21	38
8	0.53	0.47	0.50	58
9	0.83	0.59	0.69	74
10	0.43	0.53	0.48	55
accuracy			0.59	686
macro avg	0.60	0.55	0.56	686
weighted avg	0.61	0.59	0.59	686

Confusion matrix:

```
[[33  0  2  1  1  0  2  1  0  3  0]
 [ 1 35  4  3  0  0 18  1  6  0  3]
 [ 0 11 23  3  0  0  4  3  3  0  5]
 [ 0  1  0 59  0  1  6  0  2  0  3]
 [ 0  0  1  7 11  2  6  1  0  0  1]
 [ 1  0  0 18  0 51  2  3  0  0  2]
 [ 1  6  6  8  0  1 85  3  0  1  6]
 [ 1  4  3  2  0  2  3  7  5  4  7]
 [ 6  5  4  1  1  0  6  2 27  0  6]
 [ 1  3  5  2  0  2  3  3  6 44  5]
 [ 1  3  4  4  0  1  5  5  2  1 29]]
```

Method M4

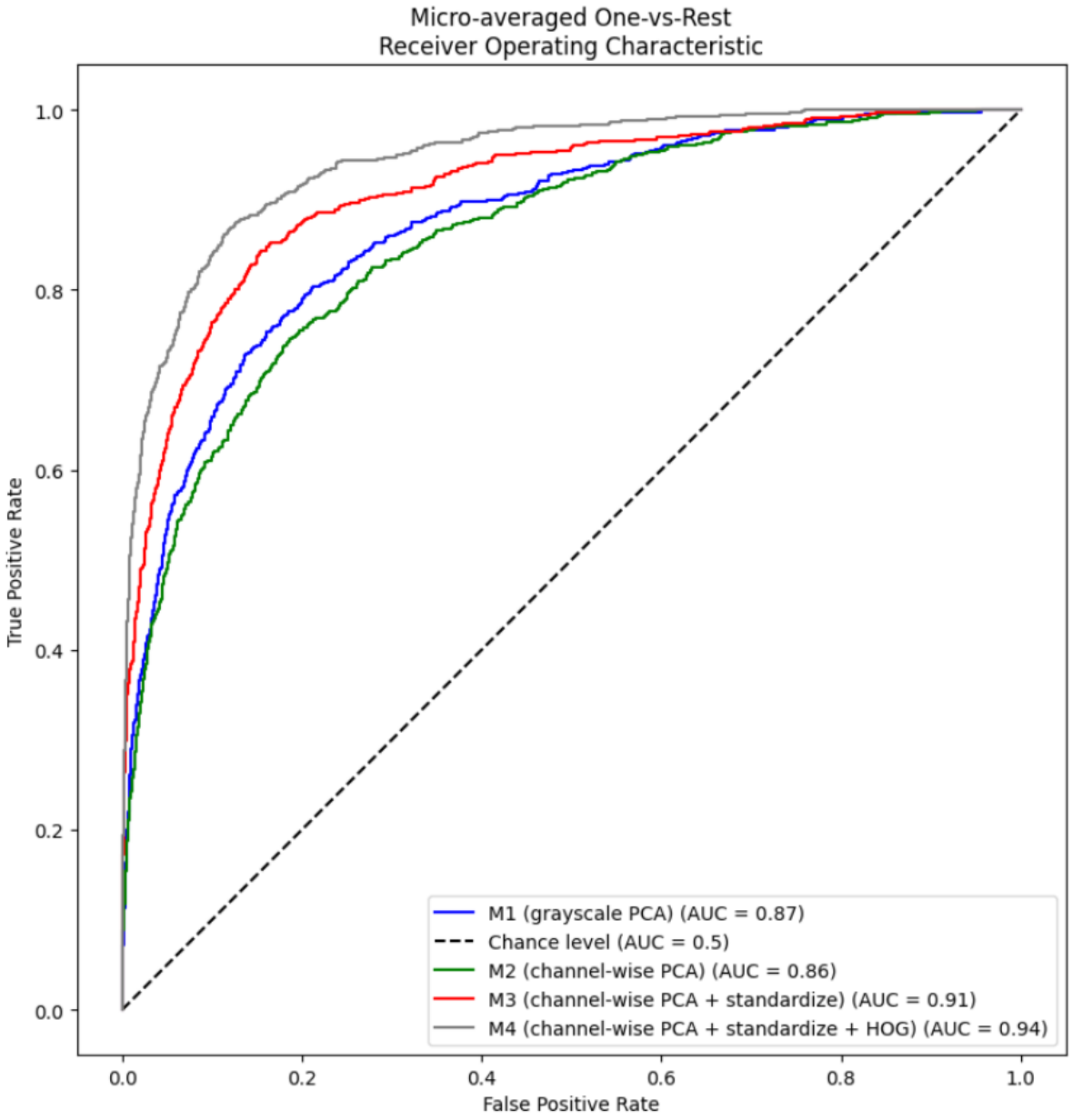
Best parameters: `{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}`
Accuracy: `0.6559766763848397`
AUC-ROC score: `0.9347909210925238`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.86	0.86	0.86	43
1	0.43	0.39	0.41	71
2	0.56	0.54	0.55	52
3	0.70	0.88	0.78	72
4	0.69	0.69	0.69	29
5	0.90	0.81	0.85	77
6	0.72	0.69	0.71	117
7	0.33	0.37	0.35	38
8	0.60	0.57	0.58	58
9	0.79	0.66	0.72	74
10	0.51	0.64	0.56	55
accuracy			0.66	686
macro avg	0.64	0.64	0.64	686
weighted avg	0.66	0.66	0.66	686

Confusion matrix:

```
[[37  0  1  1  2  0  0  0  1  0  1]
 [ 0 28  9  4  2  1 16  4  2  1  4]
 [ 0 14 28  1  0  0  0  1  3  2  3]
 [ 0  1  1 63  0  4  0  0  1  2  0]
 [ 0  0  1  5 20  0  1  2  0  0  0]
 [ 2  0  0  8  3 62  1  0  0  1  0]
 [ 1  7  3  5  0  0 81  5  5  1  9]
 [ 0  5  0  0  1  0  3 14  5  4  6]
 [ 1  4  0  0  1  1  5  3 33  1  9]
 [ 1  4  5  1  0  1  2  5  4 49  2]
 [ 1  2  2  2  0  0  3  8  1  1 35]]
```

ROCs for Support Vector Machine with different feature sets



Observations

1. Method M2 with color information in the features performs better than M1 which is grayscale PCA in terms of accuracy but not on the AUC-ROC chart.
2. Method M3 with standardization in the features performs better than M2 and also we observe that the best kernel in that case is the radial basis function kernel which expects the data to be normalized and hence the improvement.
3. Method M4 performs the best with a rich feature set including HOG features.

7. Decision Tree

Scikit-learn model: `sklearn.tree.DecisionTreeClassifier`
Hyperparameters tuned:

1. `criterion`: The function to measure the quality of a split, e.g., gini or entropy
2. `max_depth`: The maximum depth of the tree
3. `max_features`: The number of features to consider when looking for the best split
4. `min_samples_leaf`: The minimum number of samples required to be a leaf node
5. `min_samples_split`: The minimum number of samples required to split an internal node

Method M1

Best parameters: `{'criterion': 'entropy', 'max_depth': 200, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}`
Accuracy: `0.30029154518950435`
AUC-ROC score: `0.6465009186049054`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.46	0.44	0.45	43
1	0.27	0.25	0.26	71
2	0.20	0.25	0.22	52
3	0.40	0.51	0.45	72
4	0.09	0.07	0.08	29
5	0.29	0.30	0.30	77
6	0.35	0.36	0.35	117
7	0.12	0.16	0.14	38
8	0.36	0.36	0.36	58
9	0.22	0.15	0.18	74
10	0.33	0.25	0.29	55
accuracy			0.30	686
macro avg	0.28	0.28	0.28	686
weighted avg	0.30	0.30	0.30	686

Confusion matrix:

```
[[19  1  4  1  0  1  3 10  1  2  1]
 [ 1 18  5  3  2  3 22  0  9  5  3]
 [ 1  4 13  1  3  4  7  6  7  4  2]
 [ 0  6  1 37  0 15  5  2  0  3  3]
 [ 0  3  0  6  2  4  9  0  3  1  1]
 [ 0  5  5 24  1 23  6  5  3  1  4]
 [ 3 15 17  9  4  9 42  5  3  6  4]
 [ 2  4  1  1  2  5  4  6  1  8  4]
 [ 3  3  5  0  2  2 12  5 21  4  1]
 [11  4  9  6  5  7  5  4  6 11  6]
 [ 1  3  5  4  2  5  6  5  4  6 14]]
```

Method M2

Best parameters: `{'criterion': 'entropy', 'max_depth': 30, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 2}`
Accuracy: `0.4198250728862974`
AUC-ROC score: `0.7118592408747023`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.45	0.47	0.46	43
1	0.34	0.32	0.33	71
2	0.32	0.37	0.34	52
3	0.49	0.58	0.54	72
4	0.17	0.17	0.17	29
5	0.51	0.45	0.48	77
6	0.53	0.49	0.51	117
7	0.14	0.13	0.14	38
8	0.31	0.29	0.30	58
9	0.64	0.58	0.61	74
10	0.32	0.40	0.36	55
accuracy			0.42	686
macro avg	0.39	0.39	0.39	686
weighted avg	0.42	0.42	0.42	686

Confusion matrix:

```
[[20  0  5  4  4  1  1  2  4  1  1]
 [ 1 23  3  2  3  2 19  4  8  1  5]
 [ 1  4 19  5  0  2  1  4 10  1  5]
 [ 1  7  0 42  6  4  8  1  0  1  2]
 [ 3  5  3  3  5  3  3  1  0  1  2]
 [ 1  1  1 17  1 35  0  6  1  7  7]
 [ 8 12  7  6  6  5 57  1  4  0 11]
 [ 1  2  6  0  1  6  5  5  4  5  3]
 [ 1  4 11  1  2  2  3  6 17  4  7]
 [ 5  4  3  3  1  4  3  1  4 43  3]
 [ 2  6  2  2  0  4  7  5  2  3 22]]
```

Method M3

Best parameters: `{'criterion': 'entropy', 'max_depth': 30, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 2}`
Accuracy: `0.4271137026239067`
AUC-ROC score: `0.7119106574352142`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.42	0.44	0.43	43
1	0.29	0.28	0.29	71
2	0.30	0.37	0.33	52
3	0.50	0.58	0.54	72
4	0.22	0.17	0.19	29
5	0.55	0.47	0.50	77
6	0.55	0.55	0.55	117
7	0.11	0.11	0.11	38
8	0.36	0.33	0.34	58
9	0.65	0.57	0.60	74
10	0.34	0.42	0.38	55
accuracy			0.43	686
macro avg	0.39	0.39	0.39	686
weighted avg	0.43	0.43	0.43	686

Confusion matrix:

```
[[19  0  5  4  4  1  1  3  4  1  1]
 [ 1 20  3  2  3  2 21  5  7  1  6]
 [ 1  5 19  4  0  2  1  4  9  1  6]
 [ 1  7  0 42  5  4  9  2  0  1  1]
 [ 2  5  3  3  5  3  4  1  0  1  2]
 [ 1  1  2 16  1 36  0  6  1  6  7]
 [11 13  7  6  1  4 64  0  3  1  7]
 [ 1  4  6  1  1  5  5  4  3  4  4]
 [ 1  4 12  1  2  2  2  5 19  4  6]
 [ 5  3  4  3  1  3  3  1  5 42  4]
 [ 2  6  2  2  0  4  7  4  2  3 23]]
```

Method M4

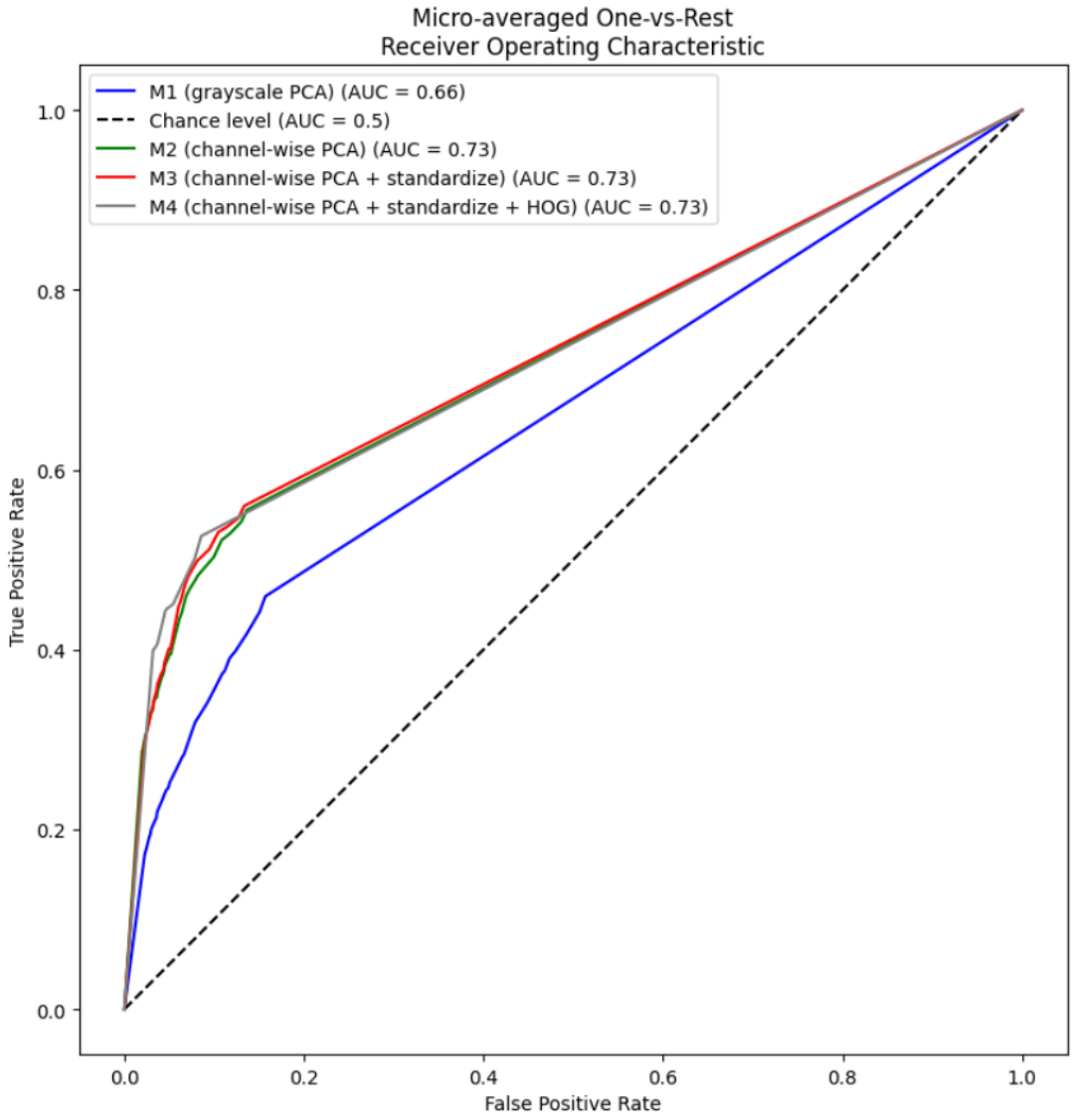
Best parameters: `{'criterion': 'entropy', 'max_depth': 30, 'max_features': None, 'min_samples_leaf': 3, 'min_samples_split': 2}`
Accuracy: `0.4518950437317784`
AUC-ROC score: `0.7132760745829757`
Precision, recall and f1-score for each of the 11 classes:

	precision	recall	f1-score	support
0	0.45	0.60	0.51	43
1	0.31	0.34	0.32	71
2	0.32	0.31	0.31	52
3	0.49	0.61	0.55	72
4	0.25	0.17	0.20	29
5	0.53	0.51	0.52	77
6	0.54	0.52	0.53	117
7	0.21	0.21	0.21	38
8	0.42	0.36	0.39	58
9	0.65	0.58	0.61	74
10	0.46	0.42	0.44	55
accuracy			0.45	686
macro avg	0.42	0.42	0.42	686
weighted avg	0.45	0.45	0.45	686

Confusion matrix:

```
[[26  0  2  5  1  1  1  1  4  0  2]
 [ 2 24  6  4  3  2 15  7  4  1  3]
 [ 1  8 16  4  0  5  6  4  4  4  0]
 [ 6  3  2 44  4 10  1  0  0  1  1]
 [ 4  2  3  6  5  2  4  1  0  0  2]
 [ 5  1  2 17  1 39  3  0  1  8  0]
 [ 3 22  6  3  2  1 61  5  3  1 10]
 [ 1  4  0  0  0  3  6  8  8  5  3]
 [ 3  6  4  2  1  1  8  6 21  2  4]
 [ 4  3  6  3  1  6  2  3  1 43  2]
 [ 3  4  3  1  2  4  7  3  4  1 23]]
```

ROCs for Decision Tree with different feature sets



Observations

1. In terms of accuracy each successive method performs better than the previous one, i.e., M4 > M3 > M2 > M1.
2. However, the AUC-ROC charts are almost similar for M2, M3, M4.

8. Conclusion

The best performance, in terms of both accuracy and AUC-ROC score, was given by the method 4 (**M4**) of creating the feature set for each of the 3 classifiers which we tried. The respective accuracies and AUC-ROC scores have been summarized in the table below.

Classifier	Best Accuracy	Best AUC-ROC Score
Naive Bayes	0.42	0.83
Support Vector Machine	**0.66**	**0.93**
Decision Tree	0.45	0.71

The overall best accuracy is achieved by the Support Vector Machine classifier, surpassing the other two in both accuracy and AUC-ROC score metrics. We achieve an accuracy of **66%** and an AUC-ROC score of **0.93**.

Libraries Used

1. Scikit-learn for
    a. PCA
    b. Train Test Split
    c. GridSearchCV (hyperparameter tuning)
    d. StandardScaler (normalizing the feature vectors)
    e. Calculating metrics
    f. Gaussian Naive Bayes classifier
    g. SVM classifier
    h. Decision Tree classifier
2. Scikit-image for
    a. Image resizing
    b. RGB to grayscale transformation
    c. HOG feature extraction
    d. Reading and writing images
3. Numpy for data handling
4. Pickle for saving/loading trained models
5. Matplotlib for plotting ROC curves
6. TQDM for tracking progress

Appendix

The complete code and trained files are present at
https://github.com/nisarg1631/Weather-Image-Recognition

classifier_nb.py (naive bayes model)

```
from utils import *

def best_model(X_train, Y_train, X_val, Y_val, X_test, Y_test,
name):
   # scikit model name
   name = name + '_nb.pkl'

   # open file for writing results
   f = open('results_nb.txt', 'a')
   print(f'Results for {name}...', file=f)

   # find optimal var_smoothing parameter using grid search
   param_grid = {
       'var_smoothing': [1e-11,1e-10,1e-9, 1e-8, 1e-7, 1e-6,
1e-5,1e-4,1e-3,1e-2,1e-1,1,10,100]
   }

   # combine the training and validation set
   X_train_val = np.concatenate((X_train, X_val), axis=0)
   Y_train_val = np.concatenate((Y_train, Y_val), axis=0)

   # set up predefined splits for grid search
   test_fold = [-1] * len(X_train) + [0] * len(X_val)
   ps = PredefinedSplit(test_fold)

   # check if model has already been trained
   grid = load_obj(name)
   if grid is None:
```

```
        grid = GridSearchCV(GaussianNB(), param_grid, cv=ps,
verbose=3, refit=False, n_jobs=-1)
        grid.fit(X_train_val, Y_train_val)
        save_obj(grid, name)

    print(f'Best parameters: {grid.best_params_}', file=f)

    # refit using the best parameters
    grid = GaussianNB(**grid.best_params_)
    grid.fit(X_train, Y_train)

    # report accuracy, precision, recall, f1 score and confusion
matrix on test set
    predictions = grid.predict(X_test)
    accuracy = accuracy_score(Y_test, predictions)
    print(f'Accuracy: {accuracy}', file=f)
    print(classification_report(Y_test, predictions), file=f)
    print(confusion_matrix(Y_test, predictions), file=f)

    # report auc roc score and plot roc curve

    label_binarizer = LabelBinarizer().fit(Y_test)
    y_onehot_test = label_binarizer.transform(Y_test)
    y_score = grid.predict_proba(X_test)

    print(f'AUC ROC score: {roc_auc_score(Y_test, y_score,
multi_class="ovr")}', file=f)

    f.close()

    return y_onehot_test, y_score

if __name__ == '__main__':
    np.random.seed(42)
```

```
    dataset_path = 'resized/'

    X, Y, classes = get_images(dataset_path)
    print(f'Classes: {classes}')
    print(f'X shape: {X.shape}')
    print(f'Y shape: {Y.shape}')

    # shuffle the dataset
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]

    # train validation test split 80 - 10 - 10
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2)
    X_val, X_test, Y_val, Y_test = train_test_split(X_test,
Y_test, test_size=0.5)

    # train the Naive Bayes model

    # method M1
    X_train_temp, X_val_temp, X_test_temp =
grayscale_PCA(X_train, X_val, X_test,
standardize_features=False)
    y_onehot_test_m1, y_score_m1 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M1')

    # method M2
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=False, with_hog=False)
    y_onehot_test_m2, y_score_m2 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M2')
```

```
    # method M3
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=False)
    y_onehot_test_m3, y_score_m3 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M3')

    # method M4
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=True)
    y_onehot_test_m4, y_score_m4 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M4')

    RocCurveDisplay.from_predictions(
        y_onehot_test_m1.ravel(),
        y_score_m1.ravel(),
        name="M1 (grayscale PCA)",
        color="blue",
        plot_chance_level=True,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m2.ravel(),
        y_score_m2.ravel(),
        name="M2 (channel-wise PCA)",
        color="green",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
```

```
        y_onehot_test_m3.ravel(),
        y_score_m3.ravel(),
        name="M3 (channel-wise PCA + standardize)",
        color="red",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m4.ravel(),
        y_score_m4.ravel(),
        name="M4 (channel-wise PCA + standardize + HOG)",
        color="gray",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    plt.axis("square")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Micro-averaged One-vs-Rest\nReceiver Operating
Characteristic")
    plt.legend()
    plt.show()
```

classifier_svm.py (support vector machine model)

```
from utils import *

def best_model(X_train, Y_train, X_val, Y_val, X_test, Y_test,
name):
    # scikit model name
    name = name + '_svm.pkl'

    # open file for writing results
    f = open('results_svm.txt', 'a')
    print(f'Results for {name}...', file=f)

    # grid search for best parameters using validation set
    param_grid = {
        'C': [0.1, 1, 10, 100, 1000],
        'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
        'kernel': ['rbf', 'poly', 'sigmoid'],
        'probability': [True]
    }

    # combine the training and validation set
    X_train_val = np.concatenate((X_train, X_val), axis=0)
    Y_train_val = np.concatenate((Y_train, Y_val), axis=0)

    # set up predefined splits for grid search
    test_fold = [-1] * len(X_train) + [0] * len(X_val)
    ps = PredefinedSplit(test_fold)

    # check if model has already been trained
    grid = load_obj(name)
    if grid is None:
        grid = GridSearchCV(SVC(), param_grid, cv=ps, verbose=3,
refit=False, n_jobs=-1)
        grid.fit(X_train_val, Y_train_val)
```

```
        save_obj(grid, name)

    print(f'Best parameters: {grid.best_params_}', file=f)

    # refit using the best parameters
    grid = SVC(**grid.best_params_)
    grid.fit(X_train, Y_train)

    # report accuracy, precision, recall, f1 score and confusion
matrix on test set
    predictions = grid.predict(X_test)
    accuracy = accuracy_score(Y_test, predictions)
    print(f'Accuracy: {accuracy}', file=f)
    print(classification_report(Y_test, predictions), file=f)
    print(confusion_matrix(Y_test, predictions), file=f)

    # report auc roc score and plot roc curve

    label_binarizer = LabelBinarizer().fit(Y_test)
    y_onehot_test = label_binarizer.transform(Y_test)
    y_score = grid.predict_proba(X_test)

    print(f'AUC ROC score: {roc_auc_score(Y_test, y_score,
multi_class="ovr")}', file=f)

    f.close()

    return y_onehot_test, y_score

if __name__ == '__main__':
    np.random.seed(42)
    dataset_path = 'resized/'

    X, Y, classes = get_images(dataset_path)
```

```
    print(f'Classes: {classes}')
    print(f'X shape: {X.shape}')
    print(f'Y shape: {Y.shape}')

    # shuffle the dataset
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]

    # train validation test split 80 - 10 - 10
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2)
    X_val, X_test, Y_val, Y_test = train_test_split(X_test,
Y_test, test_size=0.5)

    # train the SVM model

    # method M1
    X_train_temp, X_val_temp, X_test_temp =
grayscale_PCA(X_train, X_val, X_test,
standardize_features=False)
    y_onehot_test_m1, y_score_m1 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M1')

    # method M2
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=False, with_hog=False)
    y_onehot_test_m2, y_score_m2 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M2')

    # method M3
```

```
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=False)
    y_onehot_test_m3, y_score_m3 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M3')

    # method M4
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=True)
    y_onehot_test_m4, y_score_m4 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M4')

    RocCurveDisplay.from_predictions(
        y_onehot_test_m1.ravel(),
        y_score_m1.ravel(),
        name="M1 (grayscale PCA)",
        color="blue",
        plot_chance_level=True,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m2.ravel(),
        y_score_m2.ravel(),
        name="M2 (channel-wise PCA)",
        color="green",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m3.ravel(),
        y_score_m3.ravel(),
```

```
        name="M3 (channel-wise PCA + standardize)",
        color="red",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m4.ravel(),
        y_score_m4.ravel(),
        name="M4 (channel-wise PCA + standardize + HOG)",
        color="gray",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    plt.axis("square")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Micro-averaged One-vs-Rest\nReceiver Operating
Characteristic")
    plt.legend()
    plt.show()
```

classifier_dt.py (decision tree model)

```
from utils import *

def best_model(X_train, Y_train, X_val, Y_val, X_test, Y_test,
name):
    # scikit model name
    name = name + '_dt.pkl'

    # open file for writing results
    f = open('results_dt.txt', 'a')
    print(f'Results for {name}...', file=f)

    # grid search for best parameters using validation set
    param_grid = {
        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20, 30, 40, 50, 100, 150, 200,
250, 300],
        'min_samples_split': [2, 5, 10, 15, 20, 25, 30, 35, 40],
        'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9],
        'max_features': ['sqrt', 'log2', None]
    }

    # combine the training and validation set
    X_train_val = np.concatenate((X_train, X_val), axis=0)
    Y_train_val = np.concatenate((Y_train, Y_val), axis=0)

    # set up predefined splits for grid search
    test_fold = [-1] * len(X_train) + [0] * len(X_val)
    ps = PredefinedSplit(test_fold)

    # check if model has already been trained
    grid = load_obj(name)
    if grid is None:
```

```
        grid = GridSearchCV(DecisionTreeClassifier(), param_grid,
cv=ps, verbose=3, refit=False, n_jobs=-1)
        grid.fit(X_train_val, Y_train_val)
        save_obj(grid, name)

    print(f'Best parameters: {grid.best_params_}', file=f)

    # refit using the best parameters
    grid = DecisionTreeClassifier(**grid.best_params_)
    grid.fit(X_train, Y_train)

    # report accuracy, precision, recall, f1 score and confusion
matrix on test set
    predictions = grid.predict(X_test)
    accuracy = accuracy_score(Y_test, predictions)
    print(f'Accuracy: {accuracy}', file=f)
    print(classification_report(Y_test, predictions), file=f)
    print(confusion_matrix(Y_test, predictions), file=f)

    # report auc roc score and plot roc curve

    label_binarizer = LabelBinarizer().fit(Y_test)
    y_onehot_test = label_binarizer.transform(Y_test)
    y_score = grid.predict_proba(X_test)

    print(f'AUC ROC score: {roc_auc_score(Y_test, y_score,
multi_class="ovr")}', file=f)

    f.close()

    return y_onehot_test, y_score

if __name__ == '__main__':
    np.random.seed(42)
```

```
    dataset_path = 'resized/'

    X, Y, classes = get_images(dataset_path)
    print(f'Classes: {classes}')
    print(f'X shape: {X.shape}')
    print(f'Y shape: {Y.shape}')

    # shuffle the dataset
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]

    # train validation test split 80 - 10 - 10
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2)
    X_val, X_test, Y_val, Y_test = train_test_split(X_test,
Y_test, test_size=0.5)

    # train the Decision Tree model

    # method M1
    X_train_temp, X_val_temp, X_test_temp =
grayscale_PCA(X_train, X_val, X_test,
standardize_features=False)
    y_onehot_test_m1, y_score_m1 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M1')

    # method M2
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=False, with_hog=False)
    y_onehot_test_m2, y_score_m2 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M2')
```

```
    # method M3
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=False)
    y_onehot_test_m3, y_score_m3 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M3')

    # method M4
    X_train_temp, X_val_temp, X_test_temp =
channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=True)
    y_onehot_test_m4, y_score_m4 = best_model(X_train_temp,
Y_train, X_val_temp, Y_val, X_test_temp, Y_test, name='M4')

    RocCurveDisplay.from_predictions(
        y_onehot_test_m1.ravel(),
        y_score_m1.ravel(),
        name="M1 (grayscale PCA)",
        color="blue",
        plot_chance_level=True,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m2.ravel(),
        y_score_m2.ravel(),
        name="M2 (channel-wise PCA)",
        color="green",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
```

```
        y_onehot_test_m3.ravel(),
        y_score_m3.ravel(),
        name="M3 (channel-wise PCA + standardize)",
        color="red",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    RocCurveDisplay.from_predictions(
        y_onehot_test_m4.ravel(),
        y_score_m4.ravel(),
        name="M4 (channel-wise PCA + standardize + HOG)",
        color="gray",
        plot_chance_level=False,
        ax=plt.gca(),
    )

    plt.axis("square")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Micro-averaged One-vs-Rest\nReceiver Operating
Characteristic")
    plt.legend()
    plt.show()
```

utils.py (feature extraction)

```
import os
import numpy as np
import pickle as pk
from skimage.transform import resize
from skimage.io import imread, imshow, show
from skimage.feature import hog
from skimage.color import rgb2gray
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split,
GridSearchCV, PredefinedSplit
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.metrics import RocCurveDisplay, accuracy_score,
roc_auc_score, classification_report, confusion_matrix
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from tqdm import tqdm

def get_images(dataset_path):
    X = []
    Y = []
    classes = []
    for subdir in os.listdir(dataset_path):
        classes.append(subdir)
        cnt = 0
        for file in os.listdir(dataset_path + subdir):
            img = imread(dataset_path + subdir + '/' + file)
            X.append(img)
            Y.append(classes.index(subdir))
            cnt += 1
        print(f'Class {subdir} loaded. Instances: {cnt}')
    return np.array(X), np.array(Y), classes
```

```
def grayscale_transform(X):
   return rgb2gray(X)

def PCA_fit(X, n_components=65):
   pca = PCA(n_components=n_components)
   X = X.reshape(len(X), -1)
   pca.fit(X)
   return pca

def save_obj(pca, filename):
   pk.dump(pca, open(filename, 'wb'))

def load_obj(filename):
   if os.path.exists(filename):
       pca = pk.load(open(filename, 'rb'))
       return pca

def PCA_transform(X, pca):
   X = X.reshape(len(X), -1)
   X = pca.transform(X)
   return X

def channel_transform(X):
   # separate the channels for each image
   X_r = X[:,:,:,0]
   X_g = X[:,:,:,1]
   X_b = X[:,:,:,2]
   return X_r, X_g, X_b

def channel_wise_PCA(X_train, X_val, X_test,
standardize_features=True, with_hog=False):
   # extract channels
   X_r, X_g, X_b = channel_transform(X_train)
```

```
    if with_hog:
        X_train_hog, X_val_hog, X_test_hog =
hog_transform(X_train, X_val, X_test,
standardize_features=standardize_features)

    # PCA on each channel
    pca_r_path = 'pca_r.npy'
    pca_g_path = 'pca_g.npy'
    pca_b_path = 'pca_b.npy'

    pca_r = load_obj(pca_r_path)
    if pca_r is None:
        pca_r = PCA_fit(X_r)
        save_obj(pca_r, pca_r_path)
    print(f'PCA red explained variance ratio:
{pca_r.explained_variance_ratio_}')

    pca_g = load_obj(pca_g_path)
    if pca_g is None:
        pca_g = PCA_fit(X_g)
        save_obj(pca_g, pca_g_path)
    print(f'PCA green explained variance ratio:
{pca_g.explained_variance_ratio_}')

    pca_b = load_obj(pca_b_path)
    if pca_b is None:
        pca_b = PCA_fit(X_b)
        save_obj(pca_b, pca_b_path)
    print(f'PCA blue explained variance ratio:
{pca_b.explained_variance_ratio_}')

    # transform each channel
    X_r = PCA_transform(X_r, pca_r)
```

```
    X_g = PCA_transform(X_g, pca_g)
    X_b = PCA_transform(X_b, pca_b)

    # concatenate the channels
    X_train = np.concatenate((X_r, X_g, X_b), axis=1)

    # transform validation set
    X_r, X_g, X_b = channel_transform(X_val)

    X_r = PCA_transform(X_r, pca_r)
    X_g = PCA_transform(X_g, pca_g)
    X_b = PCA_transform(X_b, pca_b)

    X_val = np.concatenate((X_r, X_g, X_b), axis=1)

    # transform test set
    X_r, X_g, X_b = channel_transform(X_test)

    X_r = PCA_transform(X_r, pca_r)
    X_g = PCA_transform(X_g, pca_g)
    X_b = PCA_transform(X_b, pca_b)

    X_test = np.concatenate((X_r, X_g, X_b), axis=1)

    # standardize the data
    if standardize_features:
        X_train, X_val, X_test = standardize(X_train, X_val,
X_test)

    if with_hog:
        # concatenate hog features
        X_train = np.concatenate((X_train, X_train_hog), axis=1)
        X_val = np.concatenate((X_val, X_val_hog), axis=1)
        X_test = np.concatenate((X_test, X_test_hog), axis=1)
```

```
    return X_train, X_val, X_test

def grayscale_PCA(X_train, X_val, X_test,
standardize_features=True):
    # transform training set
    X_train = grayscale_transform(X_train)

    pca_gs_path = 'pca_gs.npy'
    pca = load_obj(pca_gs_path)
    if pca is None:
        pca = PCA_fit(X_train)
        save_obj(pca, pca_gs_path)
    print(f'PCA grayscale explained variance ratio:
{pca.explained_variance_ratio_}')

    # transform training set
    X_train = PCA_transform(X_train, pca)

    # transform validation set
    X_val = grayscale_transform(X_val)
    X_val = PCA_transform(X_val, pca)

    # transform test set
    X_test = grayscale_transform(X_test)
    X_test = PCA_transform(X_test, pca)

    # standardize the data
    if standardize_features:
        X_train, X_val, X_test = standardize(X_train, X_val,
X_test)

    return X_train, X_val, X_test
```

```
# extract hog feature vectors from images
def hog_transform(X_train, X_val, X_test,
standardize_features=True):
    # transform training set
    X_train_path = 'X_train_hog.npy'
    X_train_temp = load_obj(X_train_path)
    if X_train_temp is None:
        X_train_temp = np.array([hog(img, channel_axis=-1) for
img in tqdm(X_train)])
        save_obj(X_train_temp, X_train_path)
    X_train = X_train_temp

    pca_hog_path = 'pca_hog.npy'
    pca = load_obj(pca_hog_path)
    if pca is None:
        pca = PCA_fit(X_train, n_components=100)
        save_obj(pca, pca_hog_path)
    print(f'PCA hog explained variance ratio:
{pca.explained_variance_ratio_}')

    # transform training set
    X_train = PCA_transform(X_train, pca)

    # transform validation set
    X_val = np.array([hog(img, channel_axis=-1) for img in
tqdm(X_val)])
    X_val = PCA_transform(X_val, pca)

    # transform test set
    X_test = np.array([hog(img, channel_axis=-1) for img in
tqdm(X_test)])
    X_test = PCA_transform(X_test, pca)

    # standardize the data
```

```
    if standardize_features:
        X_train, X_val, X_test = standardize(X_train, X_val,
X_test)

    return X_train, X_val, X_test

def standardize(X_train, X_val, X_test):
    # standardize training set
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)

    # standardize validation set
    X_val = scaler.transform(X_val)

    # standardize test set
    X_test = scaler.transform(X_test)

    return X_train, X_val, X_test
```

preprocess.py (image resizing)

```
import os
import numpy as np
from skimage.transform import resize
from skimage.io import imread, imsave

# open images from dataset directory and resize them to
200,200,3
dataset_path = 'dataset/'
resized_path = 'resized/'
classes = []

# create resized directory if it doesn't exist
if not os.path.exists(resized_path):
   os.mkdir(resized_path)

   # create subdirectories in resized directory for each class
   for subdir in os.listdir(dataset_path):
       if not os.path.exists(resized_path + subdir):
           os.mkdir(resized_path + subdir)

# subdirectories in dataset directory are classes
for subdir in os.listdir(dataset_path):
   for file in os.listdir(dataset_path + subdir):
       if not os.path.exists(resized_path + subdir + '/' +
file):
           try:
               img = imread(dataset_path + subdir + '/' + file)
               img = resize(img, (200, 200, 3))
               img = np.array(img * 255, dtype=np.uint8)
               imsave(resized_path + subdir + '/' + file, img)
           except Exception as e:
               print('Error in file: ' + file)
   print(f'Class {subdir} loaded.')
```