DA2401 MACHINE LEARNING LAB
MOHAMMED ASEEL

ENDSEM REPORT
MULTI-CLASS CLASSIFIER

—

1. Summary of Models Used & System Architecture

The core objective of this project was to develop a robust multi-class image classifier using an ensemble approach. The system architecture employed a multi-stage process involving rigorous preprocessing, individual model training across various data transformations, and final aggregation via a voting ensemble.

System Architecture Overview:

1. **Data Preprocessing:** Raw image data underwent **Standard Scaling** followed by **Principal Component Analysis (PCA)** for dimensionality reduction.
2. **Model Training (Diversity Strategy):** To ensure maximum diversity and generalization, four distinct machine learning models—**Logistic Regression (SoftMax)**, **K-Nearest Neighbors (KNN)**, **XGBoost**, and Random Forest—were trained. Crucially, each model type was trained seperately on e times using different PCA component settings (e.g., PCA retaining 90%, 95%, and 98% variance). This produced a large pool of trained classifiers.
3. **Model Selection:** The top six performing individual models (selected based on validation set accuracy/F1-score) were chosen from the diverse pool generated in the training phase.
4. **Ensemble Prediction:** These six selected models were aggregated into a final **Voting Classifier (Hard Voting)** system, where the final prediction was determined by the majority vote among the constituent models.

Individual Models Employed:

Model Type	Core Algorithm	Rationale for Inclusion
Logistic Regression (SoftMax)	Linear Model	Provides a fast, interpretable linear baseline.
K-Nearest Neighbors (KNN)	Instance-Based Learning	Non-parametric model; captures local data structure.
XGBoost	Gradient Boosting	Highly effective, optimized boosting framework; handles non-linearities.
Random Forest	Tree-Based Ensemble	Robust ensemble method

2. Summary of Hyper-parameter Tuning and Individual Model Results

Standard Scalar

The Standard Scaler was implemented as the *first* preprocessing step to normalize image pixel intensities. Scaling is vital for ensuring each pixel contributes equally to the optimization process by setting the mean to 0 and the standard deviation to 1. This standardization is crucial for faster, smoother model convergence in Logistic Regression by preventing high-magnitude features from dominating the cost function, thereby enhancing generalization. This scaling was applied to the raw data using training set statistics, *before* the PCA transformation.

Principal Component Analysis (PCA) Hyper-parameter Tuning

The initial experimentation with Principal Component Analysis (PCA) focused on determining the optimal number of components required to retain a significant percentage of the dataset's variance.

Variance Retained	Number of Components
100%	728
98%	411
95%	289
90%	202

Logistic Regression (SoftMax) Hyper-parameter Tuning

I started with tuning the learning rate and number of epochs. Initial experimentation involved a grid search over these two critical parameters to establish a baseline performance. The learning rate was varied across a logarithmic scale (e.g., 0.1, 0.01, 0.001) while the epoch count was tested at conservative values (e.g., 50, 100,150,200). The most significant improvement was observed when using a lower learning rate (0.01) coupled with a moderate number of epochs (150), which helped prevent rapid convergence to a sub-optimal local minimum and allowed the model sufficient time to learn the class boundaries effectively.

PCA Components	Learning Rate (lr)	Epochs	Validation Accuracy (%)	Training Time (s)
289	0.01	200	93.76	5.47
411	0.01	200	93.72	5.85
289	0.1	100	93.72	2.78
411	0.01	100	93.60	2.97
289	0.01	100	93.52	2.69
411	0.1	100	93.28	2.95

K-Nearest Neighbors (KNN) Hyper-parameter Tuning

The optimal setting for the K-Nearest Neighbors (KNN) model involved a grid search focusing on the number of neighbors (k) and the impact of varying levels of dimensionality reduction via PCA. The value of k was explored across the range of 3 to 10. The results below highlight the trade-offs between model complexity (number of PCA components) and performance metrics (accuracy and training time).

PCA Components	Optimal k	Training Accuracy (%)	Validation Accuracy (%)	Training Time (s)
411	6	95.08	92.60	70.03
289	4	96.86	93.28	45.51
202	6	95.45	93.28	23.66

XGBoost Hyper-parameter Tuning

The tuning process for the XGBoost classifier involved a grid search focusing on the ensemble's structural parameters: `n_estimators` (number of trees), `max_depth` (maximum depth of each tree), and `learning_rate` (step size shrinkage). To assess the trade-off between predictive power and computational cost, this tuning was performed across the different PCA dimensionality settings.

A consistent optimal set of parameters—`learning_rate`: 0.1, `max_depth`: 5, `n_estimators`: 50—was identified across all PCA settings. The results demonstrate how reducing dimensionality significantly decreased training time while maintaining competitive validation accuracy, with the 289-component model providing the best balance.

PCA Components	Best Parameters Found	Training Accuracy (%)	Validation Accuracy (%)	Training Time (s)
411	{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}	98.66	91.48	522.83
289	{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}	98.51	91.76	325.62
202	{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}	98.18	90.72	220.88

I ultimately abandoned the idea of incorporating XGBoost into the voting part because the training time proved to be excessive during my attempts.Also it has validation accuracy much lower than others.

Random Forest Hyper-parameter Tuning

The Random Forest model was incorporated for its robust performance and inherent resistance to overfitting, stemming from its ability to average multiple decorrelated decision trees. The tuning focused on two key parameters: `n_estimators` (number of trees) and `max_depth` (maximum depth of each tree). This process was critical to balancing model complexity with the available training time.

A key observation was that, similar to XGBoost, training time scaled significantly with the number of PCA components. However, unlike other model

However, I eventually had to drop it because the time constraints and validation errors were unfavorable.

PCA Components	Best Parameters Found	Training Accuracy (%)	Validation Accuracy (%)	Training Time (s)
411	{'max_depth': 10, 'n_estimators': 100}	91.12	89..92	615.40
289	{'max_depth': 15, 'n_estimators': 50}	91.88	84.00	548.75
202	{'max_depth': 10, 'n_estimators': 50}	89.25	83.88	429.11

3.1 Steps to Optimize System Performance and Limit Run Time

The primary strategies deployed to optimize system performance, limit training time, and maintain generalization were:

1. **Dimensionality Reduction via PCA:**
    - **Action:** Principal Component Analysis (PCA) was applied to reduce the feature space significantly (from 728 components to as low as 202, retaining 90% variance).
    - **Impact:** Training time for all subsequent models (especially distance-based models like KNN and iterative models like Logistic Regression) was drastically reduced without major sacrifices in validation accuracy. The smaller feature set required fewer calculations per iteration/comparison.
2. **Standard Scaling:**
    - **Action:** Standard Scaling was applied *before* PCA to normalize pixel intensities.
    - **Impact:** This preprocessing step accelerated the convergence of the gradient-descent-based Logistic Regression model.
3. **Strategic Hyper-parameter Tuning (Focus on Convergence and Speed):**
    - **Logistic Regression:** A controlled approach involving tuning the learning rate and epochs was used to achieve optimal performance (Validation Accuracy 93.7%) in a moderate time (Training Time 5.5 seconds) by preventing excessive epoch counts.
    - **K-Nearest Neighbors (KNN):** Experimentation favored lower PCA component settings (e.g., 202 components) which, even with a slightly less optimal k, resulted in a massive reduction in training time (from 70s down to 23s) while maintaining competitive validation accuracy (93.28%).
4. **Ensemble Method Selection (Hard Voting):**
    - **Action:** A final **Hard Voting** ensemble was chosen.

- **Impact:** Hard Voting, which simply counts the majority prediction, is computationally faster for prediction than Soft Voting (which requires averaging probabilities). This choice maintained high predictive power while minimizing latency during the final classification phase.

3.2 Evaluation Results

Ensembled Model Accuracy: 93.64%