

Final Project Report, Part 2

Prepared For:

Dr. Hichem Frigui

CSE 546, Introduction to Machine Learning

Department of Computer Science and Engineering

University of Louisville

Prepared By:

Jacob Sanders (Supervised Learning – MLP, Naive Bayes, Bagging, AdaBoost)

Cristian Levya (Supervised Learning – Random Forests)

Nicolas Kosanovic (Unsupervised Learning - SVM)

Date: 4 December 2024

## Abstract

Optimized machine learning toolboxes such as Python’s *Sci-Kit Learn* enable rapid prototyping and production of quality machine learning models for a variety of tasks. In this work, an image classifier that uses the outputs of a Convolutional Neural Network (CNN) trained on images from the FLIR Thermal Image Dataset is developed. By using classical estimators on CNN outputs, explainable insights are gained from otherwise incomprehensible neural network features. The Support Vector Machine, Naive-Bayes, Random Forest, and Multi-Layer Perceptron models were used in tandem with normalization and Principal Component Analysis to build a single optimal model. Furthermore, techniques like feature selection, grid-search hyperparameter tuning, and ensemble methods enabled the best developed pipeline (a 50 node, single-layer neural network using 16 principal components) to attain an average score (accuracy, AUC-ROC, F1-measure) on validation-set of 92.97% when classifying images from the FLIR Dataset. The individual metrics from this pipeline and model are accuracy of 89.93%, Area under the Receiver-Operating Characteristic Curve (AUC-ROC) of 98.72% and an F1-measure of 90.25%.

## Introduction

Object detection from images represents an increasingly important and tangible application of machine learning in the contemporary world. Convolutional Neural Networks (CNNs) have shown aptitude in this domain, as their ability to break down images into a plurality of patterns (edges, shapes, colors, etc.) enables them to grasp macroscopic features instead of individual patterns within pixels. Despite these capabilities, CNNs can behave like a *black box,* giving little insight as to which features are being used and for which purpose. By using a separate model pipeline, the individual features of the CNN are given distinct importance, potentially streamlining future classification efforts. Furthermore, techniques can be used to discard unimportant CNN features, hastening runtimes.

This pipeline, of taking CNN outputs and running a custom Machine Learning (ML) model on them, is used on the FLIR Thermal Image Dataset. The dataset contains 5000+ images of bikes, busses, people, cars, and signs taken with IR thermal cameras. The images are greyscaled and vary in height and width. In this work, multiple models are created using prevalent algorithms such as Support Vector Machines, Naive-Bayes, Random Forests, and Multi-Layer Perceptrons to classify CNN feature outputs as one of the five previously mentioned classes.

## Methodology

*Modularity and Consistency*

To ensure consistency between experiments and to avoid repetition of code, a *common* module was created to facilitate this. Within this module, a framework for conducting the experiments was defined. It contains functions for loading and displaying images in a standard way, similar to provided code. To ensure consistency in the initial data between experiments, functions were added to load and consolidate data to be learned from. To ensure consistency in the fitting, scoring, and cross-validation process, a function, *validation\_scores(),* was added to handle the execution of an experiment provided the components: pipeline, param grid(s), X and y training data. The function executes the experiments three times using three different scoring methods: accuracy, roc\_auc, and f1\_score, returning the complete results for further analysis. At every point, random states were set explicitly to, again, provide consistency throughout the experimental process and between researchers.

*Unsupervised Learning*

Prior to generating models that will map inputs to outputs, it is often useful to gain a deeper understanding of the dataset. Regrettably, visualizing raw data is impossible when a dataset contains more than three features. To this end, two techniques, namely Principal Component Analysis (PCA) and Feature Selection can be used to *compress* the dataset (either for visualization or accelerated training). The former analyzes covariance among the features to formulate a special transformation which retains significant amounts of variability among samples, while the latter seeks to capture an optimal variance to number-of-features ratio in the original dataset without explicit transformation. PCA is particularly useful for transforming the high-dimensional dataset to two or three principal components that can be visualized. In contrast, Feature Selection is accomplished via a Variance Thresholding with a threshold value of 0.2 or using SelectKBest to identify the 8 features with the most variance.

Outside of dataset compression, Unsupervised Learning is also used for *clustering* to identify critical characteristics of the dataset, including patterns similar (or differing) from the rest of the data. With such knowledge, one can forecast which samples may be challenging for models, especially for those with probabilistic outputs. Thus, two clustering algorithms are employed: K-Means Clustering and TSNE. The former is commonly used but has the drawbacks of struggling with non-convex data distributions and requiring a user-specified number of clusters. In contrast, TSNE is fully autonomous, requiring no hyperparameter specification.

*Supervised Learning*

After setting up common code that we would use in all our models, we got to work setting up our pipelines for each model. We started with SVM and decision trees for this report. We decided upon these because of their benefits, namely SVM’s plentitude of regularization techniques that would improve our accuracy as well as working well with larger datasets. Decision trees were a good choice too for their ensemble methods of Random Forests and Gradient Boosting, as well as how quickly they can be trained. Decision trees are also very clear to interpret.

*Supervised learning: Decision Trees*

Starting with decision trees, we set up two pipelines for now, both with Random Forest ensemble, but one with Standard Scaling and the other with MinMax Scaling. They would both perform a grid search on the parameter of max depth for the trees, ranging from 1 to 5.

Then we moved on to Gradient-Boosted ensemble method. We set up two pipelines with this model, one with Standard Scaling and the other with MinMax Scaling. This set of pipelines would only perform grid search on depths of 1 to 3 as the ensemble method works best by correcting the mistakes of low depth trees. It also helps to reduce the fitting times of an already time consuming model.

*Supervised Learning: Multilayer Perceptron (MLP)*

The next experiment considered the Multilayer Perceptron (MLP) classifier, attempting to identify suitable preprocessing configurations and model parameters. First, the pipeline was defined to ensure a consistent and reasonable order of operations for each of grid-based fits to follow. The following order was chosen for preprocessing steps, which was then followed with model definition for fitting:

1. Data Normalization
2. Dimensionality Reduction
3. Feature Selection

Once the pipeline was defined, grid search was constructed with reasonable initial preprocessing methods and parameters, as well as model parameters. At this stage the values were not expected to be optimal. The choices were primarily chosen to provide information to narrow the search area for optimal parameters later and to be explorable in a reasonable amount of time.

|  |  |  |
| --- | --- | --- |
| **Step** | **Option 1** | **Option 2** |
| 1. **Scaler** | StandardScaler() | MinMaxScaler() |
| 1. **PCA** | PCA(n\_components=8) | PCA(n\_components=16) |
| 1. **Feature Selection** | SelectKBest(k=8) | VarianceThreshold(threshold=0.2) |

Table 1: Preprocessing step options explored in grid searches for MLP Classifier

|  |  |  |
| --- | --- | --- |
| **MLP Parameter** | **Option 1** | **Option 2** |
| **MLP Hidden Layer Sizes** | (50,) | (50,50) |
| **MLP Activation** | relu | tanh |
| **MLP Solver** | adam | sgd |
| **MLP Alpha** | 0.0001 | 0.01 |

Table 2: MLP Classifier parameter options explored in grid searches for MLP Classifier

Experiments using combinations of the options detailed above in Table 1 and Table 2 were executed using the *validation\_scores()* method from the common module*.* This yielded 385 results for each combination of parameters and scorer method. These results were combined by averaging the scores and standard deviation of scores for each three tests for each combination. This resulted in 128 results which for brevity, only the top 10 are shown below in Table 6.

*Supervised Learning: Naive Bayes*

The Naive Bayes classifier was chosen for experimentation due to its explainability, simplicity, speed, and its difference in approach from the other methods. As there are no hyperparameters to tune for Naive Bayes, the experimentation focused on structured tuning of preprocessing pipeline parameters. Parameter tuning was conducted through grid search for many reasonable values (see Table 3). Many more than other models could be tested due to the rapid training and testing of Naive Bayes. The experimental setup also evaluated the impact of clustering to enhance the classification performance, confirming the utility of clustering in specific scenarios.

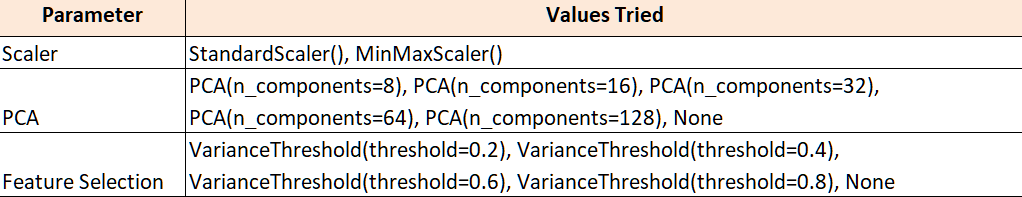


Table 3: Preprocessing parameters options explored in grid searches for Naive Bayes Classifier

*Supervised Learning: Support Vector Machine Classifier (SVM)*

A simpler model than an MLP is the Support Vector Machine (SVM). Such a model has easily interpretable hyperparameters, tools for regularization, and even nonlinearity via the kernel trick. Hence, SVMs can be especially powerful for learning nonlinear trends in data, especially among correlations between features. The same pipelining steps from the MLP classifier were used to build the SVM models, with the key difference being the hyperparameter Grid Search space. The SVM relies primarily on the inverse L2 regularization penalty *C*, the kernel type, and the kernel coefficient *Gamma*; the ranges of said parameters are listed in Table 4.

|  |  |
| --- | --- |
| **SVM Parameter** | **GridSearch Range** |
| **Regularization C** | 0.001, 0.01, 0.1, 1, 10, 100 |
| **Kernel Type** | RBF, Linear |
| **Kernel Gamma** | 0.001, 0.01, 0.1, 1, 10 |

Table 4: MLP Classifier parameter options explored in grid searches for MLP Classifier

*Supervised Learning: Ensemble Bagging and VotingClassifier*

After the previous experiments with preprocessing configurations and optimization of base models was completed, the results were used to build a Voting Classifier using bagging. A pipeline with reasonable preprocessing steps and transformations based on previous results was created for use with the ensemble. This included the parameters and clustering as can be seen described in the Table 5 below. The base models for ensemble learning include a **Gaussian Naive Bayes** classifier, a **Support Vector Machine (SVM)** with an RBF kernel, a **Random Forest**, and a **Multi-Layer Perceptron (MLP)** with specific hyperparameters optimized for each model. For bagging, individual classifiers are wrapped in the **BaggingClassifier**, while the **VotingClassifier** combines predictions from multiple models using the **soft voting** strategy. Performance metrics are computed on the test set to compare the effectiveness of the ensemble method.

A close-up of a table

Description automatically generated

Table 5: MLP Classifier parameter options explored in grid searches for MLP Classifier

*Supervised Learning: Ensemble AdaBoost and VotingClassifier*

Following the optimization of models and preprocessing configurations of base models from earlier experiments, an AdaBoost ensemble method was constructed making use of the optimal preprocessing and base models. A pipeline matching the one used for Bagging (except for *model)* was used (see Table 5 above). These preprocessing and clustering steps were used in combination with three of the four optimized models to create a VotingClassifier leveraging AdaBoost. The Multi-Layer Perceptron (MLP) was excluded due to incompatibility with AdaBoost’s requirement for base classifiers to support weighted training samples. Implementing this was beyond the scope of our experimentation, so unfortunately MLP had to be excluded from AdaBoost. Like Bagging, soft voting strategy was used. Evaluation metrics on the test set provide a comparison of AdaBoost-based ensemble again alternative methods.

## Results

*Unsupervised Learning*

Data compression and initial visualization is done via PCA. Figure 1 depicts a plot showing changes in explained variance as the number of components increases. Little improvement for additional components is observed beyond analyzing the first 5 principal components. Moreover, Figure 1 also shows minute differences in explained variance among the three different dataset transformations applied, namely Unscaled Data (US), Standard Scaled Data (SS), and MinMax Scaled Data (MM).

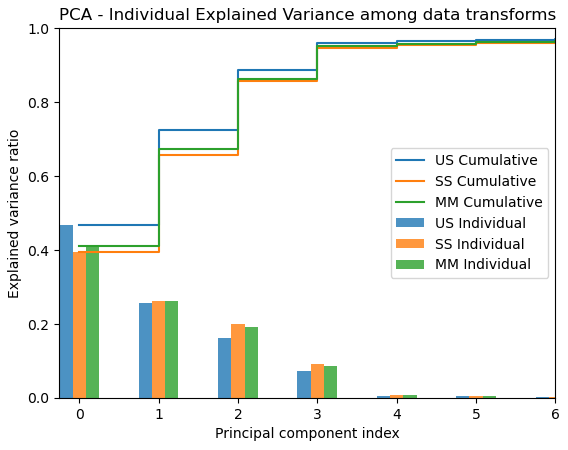


Figure 1: Principal Component Analysis chart showing differences in individual and cumulative explained variances for all three scaled models using between 1 and 7 principal components.

Regardless of the selected number of components, Figure 1 indicates that over 60% of the explained variance exists within the first two principal components. Thus, visualization of the effects of such dimensionality reduction is shown in Figure 2, where the plot on the left represents all training samples from the US, the center is SS, and the rightmost plot is MM. Colors are determined by the corresponding samples’ output class.

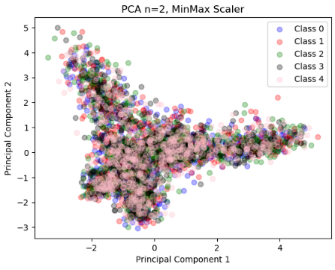
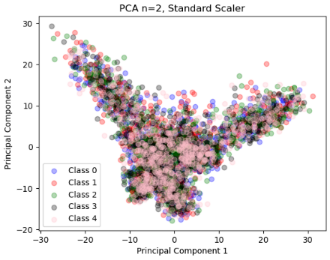
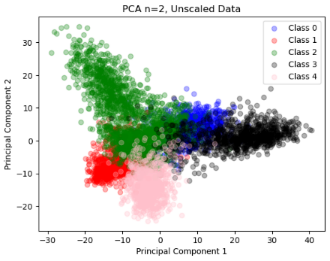


Figure 2: Visualization of PCA with dataset labels for all three scaled datasets.

TSNE was also used for visualization of potential clusters within the transformed datasets (Figure 3), as PCA showed little useful information in scaled configurations.

A diagram of a dataset

Description automatically generated with medium confidenceA diagram of a number of datasets

Description automatically generatedA diagram of a number of data set

Description automatically generated with medium confidence

Figure 3: TSNE visualization of all scaled datasets.

Regrettably, TSNE suffered a similar fate to the 2-dimensional PCA; excessive overlap of all classes prevents human-intelligible patterns from being recognized. It appears that usage of only two components does not provide ample information for the data to be clearly discriminable. Even for Figure 3 (left), which appears to have some clear class clusters, a not-negligible amount of samples reside beneath each clump, harming the dimensionally-reduced dataset’s efficacy for clustering.

Straying from transformations, unsupervised learning by way of K-Means Clustering was done. Using the Sum of Squared Error (SSE) plots from Figure 4 indicates a “knee point” about *k=8*. Furthermore, the difference plot (Figure 4, right) shows a leveling-off of the gradient about *k=8*, reinforcing the notion that K-Means with 8 clusters is likely a point-of-interest. Thus, silhouette plots for all data scaling are produced in Figure 5.

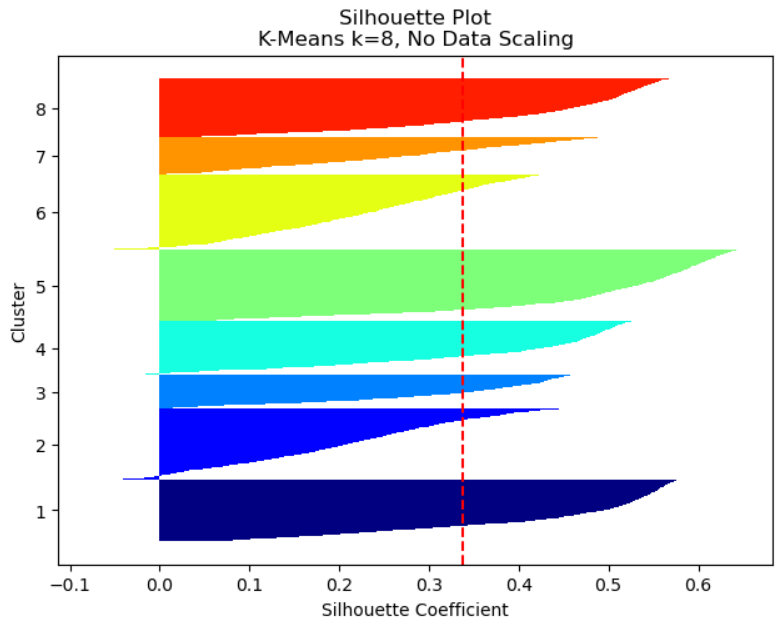
A graph with a line

Description automatically generatedA graph with a red line

Description automatically generated

Figure 4: SSE vs. # of K-Means Clusters for the MinMax Scaled Dataset

Outside of pure exploratory visualization, Silhouette Plots were used to identify *silhouette coefficients* corresponding with the datasets to evaluate clustering capabilities. Higher silhouette coefficients (up to +1.0) correspond with lower distances between a given sample and the location of the cluster’s centroid, whereas lower values (<0.2) indicate the opposite.

A chart of different colors

Description automatically generatedA chart of different colored lines

Description automatically generated

Figure 5: Silhouette plots of all scaled datasets for K Means Clustering with 8 neighbors each.

The silhouette plots indicate that all three scaled datasets attained similar average values. Moreover, the silhouette plot of the MinMax scaled plot shows particularly strong samples about cluster 4 and 8. Additional analysis of the clusters is done by viewing core samples (samples with the greatest silhouette coefficients) as in Figure 6. Clusters 1, 3, 4, and 5 are particularly interesting; the first exhibits a clustering of vehicles (bikes, buses, and cars) for the prevalence of a wheel pointing axially towards the camera. Nevertheless, cluster 3 shows vehicles at an angle, cluster 4 appears to group images up by presence of white colors (indicating heat, for the thermal camera), and finally, the fifth cluster shows vehicles from the back. Since multiple samples showcased similar image features in clusters, such features may lead to similar classification of such samples from the model.

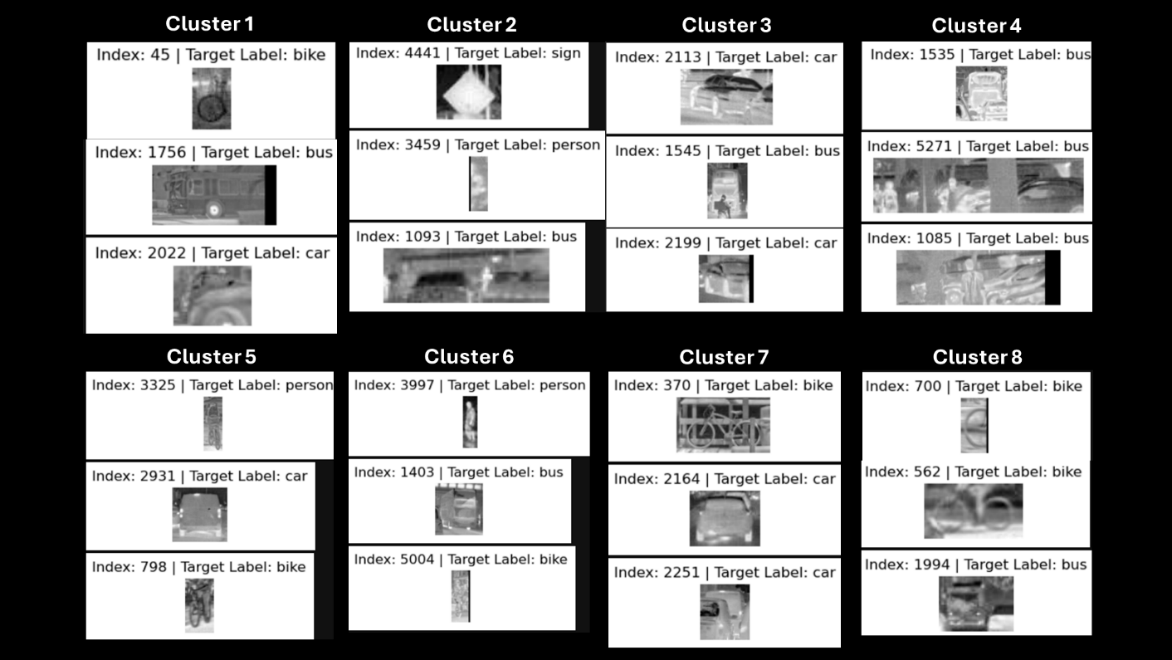


Figure 6: Silhouette plot cluster core samples for MinMax Scaled Data. Such samples agree most with KMeans clusters.

*Supervised Learning*

Sample results from select Decision Tree pipeline experiments are shown in Figure 7. This highlights differences in accuracy, mean fit time, AUC-ROC, and the f1 score.

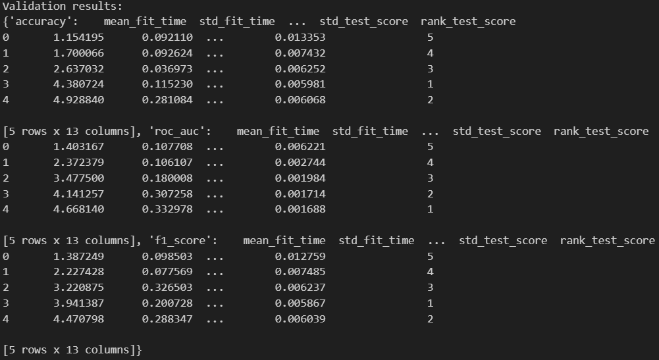
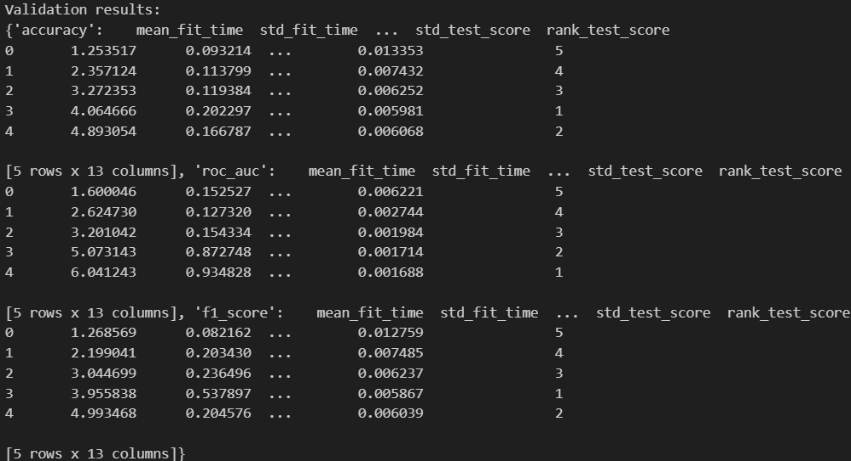


Figure 7: Standard Scaling (left), Random Forest Ensemble (right)Results

Scores for the Gradient-Boosted model generally fared worse than the random forest models, however many of the averaged scores across the three different scoring methods were close.

*Supervised Learning: MLP Classifier*

Overall, MLP performed well for most all combinations of parameters, the lowest average score of all three scoring methods was 88.50%. Because of the volume of data (128 combinations), only the top 10 average scoring models are shown in the Table 6 below. The best accuracy achieved was 92.97% with a low standard deviation of 0.0035, which is consistent.



Table 6: MLP Classifier – Top 10 Performing Model Options, Avg. Mean Scores, and Avg. Std. Of Scores

*Supervised Learning: Naive Bayes*

The Naive Bayes classifier with cluster labels for KMeans(n=5) performed about as well as could be expected for reasonably complex data (see Table 7). Various preprocessing techniques were evaluated. The best performing can be seen below in Table 7, where each scoring method is shown. The parameters compound what has been seen in other experiments. Namely, that StandardScaler data normalization is extremely well-suited to the data, that a large amount of the important information and variance is present in few of the features. This is very convenient for training speed and thus the variety of configurations that can be tested. There is little deviation in the scores between splits so we can be fairly sure that these results will remain consistent. The best-performing configuration was identified as using a variance threshold of 0.4, PCA with 32 components, and a standard scaler. These parameters were chosen based on their ability to optimize metrics such as accuracy, F1-score, and ROC-AUC across cross-validation folds.

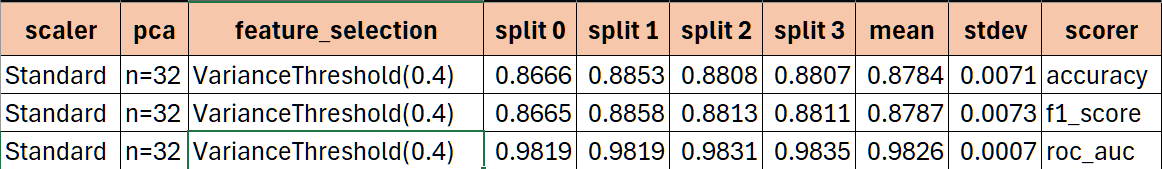


Table 7: Naive Bayes Classifier – Top Parameters Performance

*Supervised Learning: SVM*

Grid search with SVM rapidly found a set of optimal hyperparameters, namely: a regularization parameter C of 10, a Radial-Basis Function kernel, and a kernel coefficient of 0.001. Other aspects of the pipeline (and the model’s accuracy on the validation dataset) are found in Table 8. Using the Standard Scaler with a small PCA (encompassing little more than the first couple PCs that provide >95% of the cumulative explained variability) and a slight movement to the Variance Threshold yields a classification accuracy greater than 90%. Furthermore, a high F1 Score, indicates that few false positive/negative classifications are made when compared to true positives. Finally, the ROC AUC resides at greater than 90% as well, indicating that the model could correctly distinguish between classes with a high accuracy. That said, this is still a few percentage points shy of being the best model attained.



Table 8: SVM tuned hyperparameters and associated test-set performance

Beyond purely quantitative metrics of measuring this model, qualitative hypotheses can be made about the model’s general capabilities by observing sample images from its correct and incorrect classifications. Beginning with the “impossible” difficulty samples, Figure 8 shows an image with little to no comprehensible information to the human eye. The model unsurprisingly failed to correctly classify this image, likely because the absence of patterns and the heavy prevalence of noise made this unintelligible.

A close-up of a sign

Description automatically generated

Figure 8: "Impossible"-difficulty sample

Next, sample images that are readily understandable by the human eye are observed in Figure 9. Such samples highlight the model’s failings and can give insight into which tangible features the model seeks to classify an image. Here, both signs and vans are misclassified as busses. This indicates a potential bias in the model towards considering large rectangular objects busses. Additionally, the model commonly misclassifies images of singular bicycle wheels, taken from an angle, as cars. From these select samples, it is hypothesized that the CNN is picking up primarily on image shapes and orientations; the rectangular shapes of buses are likely found prevalently thorughout the dataset, and thin images of wheeled vehicles could have commonly shown cars as well. Recall, clusters 1 and 8 from Figure 6; thiey showed that the dataset possesses several clusters of images of wheeled vehicles from particular orientations, possibly explaing these misclassifications.

A close-up of a sign

Description automatically generatedA black and white photo of a sign

Description automatically generatedA close-up of a white object

Description automatically generatedA back of a van

Description automatically generated A blurry image of a person's face

Description automatically generatedA close-up of a person on a motorcycle

Description automatically generated

Figure 9: The SVM model commonly confuses square objects (signs and vans) with busses. Moreover, it also misclassifies single-wheeled bikes as cars.

Despite highlighting the SVM model’s failings, it showed promise in several regards as well. For instance, It correctly classified over 90% of testing images, comprising all classes as well. Figure 10 shows the model correctly identifying each class from a selection of samples from the test dataset. That being said, the model was observed to particularly struggle with classification of signs, likely due to its aformentioned proclivity to consider rectangular objects as busses.

A blurry image of cars

Description automatically generatedA blurry image of a person walking

Description automatically generated A close-up of a bike

Description automatically generated A bus on the road

Description automatically generated A close-up of a sign

Description automatically generated

Figure 10: Correctly classified samples via SVM

*Supervised Learning: Ensemble Bagging and Voting Classifier*

The test scores for the ensemble bagging are shown below (see Table 9). The inclusion of clustering labels as additional features likely enriched the feature space, aiding the ensemble classifier's ability to capture patterns in the data. Overall, this combination of preprocessing and modeling demonstrates a comprehensive approach that balances feature engineering with robust classification techniques, yielding results that are both interpretable and high-performing. This ensemble classifier was then used to generate labels for the final test data for submission.

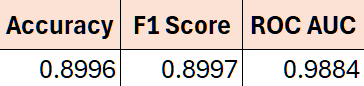


Table 9: Ensemble Bagging VotingClassifier Test Scores

*Supervised Learning: Ensemble AdaBoost and VotingClassifier*

The test scores for the AdaBoost ensemble are presented below (see Table 10). Despite the exclusion of MLP, the remaining classifiers demonstrated strong compatibility with AdaBoost's boosting mechanism, contributing to a well-balanced and effective ensemble. This combination of preprocessing, feature engineering, and ensemble learning illustrates a robust approach that has potential to yield competitive and interpretable results – even more interpretable due to the lack of MLP, but less competitive.

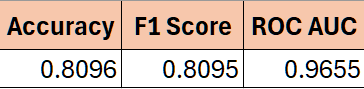


Table 10: Ensemble AdaBoost VotingClassifier Test Scores

## Analysis and Discussion

*Unsupervised Learning*

As shown in Figure 1, a significant portion of the dataset is described by the first half dozen principal components, meaning significant savings in computational time are available via dimensionality reduction. Of the dozens of experiments run, little variation in training speed and accuracy was observed when altering the number of PCs. For that reason, lower PC counts (particularly n=8 and 16) were preferred for lowering training times. Even though PCA could be pushed to provide fewer numbers of PCs, (as per Figure 2), this came results in lowered dataset intelligibility, as demonstrated by the chaotic display of the 2-dimensional PCA and TSNE data.

Nevertheless, K-Means clustering was used to identify patterns among samples within the dataset. From the SSE and silhouette plots of Figures 5 and 6, to the cluster core samples shown in Figure 7, there exist several similar image features among the samples. Some included shapes, colors, and patterns, each of which could have contributed to particular biases or misclassifications for models.

*Supervised Learning*

For our decision tree models, they were able to run on the entire dataset with medium speed, both taking around three and a half minutes. All three scoring methods (accuracy, F1, and AUC of ROC) follow the same trend of scoring better with more depth but leveling off. We can also see as max depth increases that so does the mean fit time. The standard deviation of test scores also lowered as the max depth increased, as expected because deeper trees capture more data and patterns at the cost of time. Both F1 and accuracy scores were highest at max depth of 4 and for AUC of ROC, a max depth of 4 was second only to 5 by hundredths of a percent, thus this seemed to be our best depth for sure. The scores did not seem to differ much by normalizing the data. The best average scores were 90.20%, 98.58%, and 90.23% for accuracy, roc\_auc, and f1 respectively.

Generally, the gradient-boosted models fared worse than their random forest counterparts. Additionally, they ran a lot slower, with cross validation taking about 30 minutes to run for each model despite searching up to a depth of 3. Similar to random forests, these models weren’t affected much by normalizing the data. The best average scores were 89.78%, 98.60%, and 89.80% for accuracy, roc\_auc, and f1 respectively, all achieved with a max depth of 2.

*Supervised Learning: SVM*

Support Vector machines attained intermediate performance. With optimal hyperparameters of C=10, gamma=0.001, and an RBF kernel, the model achieved an accuracy of 90.6%, an F1 score of 98.4%, and an ROC AUC of 90.6%. Thus, SVM exhibited slightly better performance than random forests and gradient-boosted decision trees in terms of accuracy and ROC AUC, but falters considerably compared to the MLP classifier. Overall, tuning the L2 regularization hyperparameter proved most important, as it shows the most performance gain per increment. Varying data scaling from standardization to normalization showed little benefit, however, by their nature SVMs can suffer from unscaled data. Furthermore, the SVM had struggled with discriminating samples akin to those found in the K-Means core clusters, indicating that the model may have been learning abstract image features like shapes.

*Supervised Learning: MLP Classifier*

The top results from the grouped averages of scores highlight the optimal parameter combinations that yield the highest performance across different scoring metrics. For example, the parameter set using *SelectKBest(k=4)*, *relu* activation, *adam* solver, *PCA(n\_components=8)*, and *StandardScaler()* consistently achieves high average test scores across all splits, with an average mean test score exceeding 0.92. This suggests that this combination balances feature selection, dimensionality reduction, and scaling effectively to enhance the neural network's performance. The low standard deviation in test scores for these top results also indicates consistency and robustness across different data splits.

## Conclusion & Future Work

Following the preliminary results of phase one experimentation, the path was cleared for new model experimentation and refinement. Further evaluation of decision trees with Gradient-Boosted ensemble methods were unfortunately unfruitful. New classifiers Naive Bayes and SVM offered new insight and opportunities. Unfortunately, they were not able to add substantial benefits to accuracy, but likely did add robustness through ensemble methods.

Continued refinement of Multi-Layer Perceptron (MLP) classifier was marginal in its benefit. The addition of clustering labels via KMeans did improve accuracy but only by a tiny amount. It seems that MLP was already quite optimal and possibly already picking up on qualities of the features that clustering was revealing. Nonetheless, some benefit was offered. Impressively, and somewhat surprisingly, MLP revealed itself to be the most accurate of all models, including ensemble methods of which it was a part. The highest accuracy achieved by MLP and any model a part of these experiments was 92.97%.

Finally, the true ensemble methods of Bagging and AdaBoost were used to hybridize the models in the hopes of achieving increased generalization and performance. However, from the results it can be seen that the Multi-Layer Perceptron outperformed both of the ensemble methods. Thus, it seems that the other models including the ensemble has a diminishing effect on the performance of MLP in the case of Bagging. This is supported by the AdaBoost results as it achieved a much lower accuracy Bagging and did not include MLP due to a compatibility issue.

Additional experimentation with other classifiers or a resolution to the compatibility issue between MLP and AdaBoost would very likely allow for even greater model performance to be achieved. It could be fruitful to experiment further with new classifiers such as K-Nearest Neighbors (KNN) if resources permit. Also, additional data normalization methods could be explored beyond the two relied on here. There is certainly room for additional fine refinement between the various parameter values explored here as well, though with diminishing returns.