German Traffic Sign Images Classification

281 Final Project - Fall 2023

Lisa Wu **Ed Brown** **Alejandro Reskala**

{lisa wu, teddybrown, a_reskala}@ischool.berkeley.edu

Abstract

Traffic sign recognition is a real problem and applies to a wide range of use cases, such as low-vision impaired drivers, intelligent transportation (self drive vehicles) systems, bad weather conditions, blocked visibility by vehicles, and other obstruction conditions or objects. Our project motivation is to explore computer vision methods to recognize traffic signs from the images accurately.

We used the German Traffic Sign Recognition Benchmark image dataset from Kaggle[1] (Figure 2 shows an example of the traffic sign images). We started with the basic image feature extraction methods, such as Histogram of Oriented Gradients (HOG), Hue Saturation Value (HSV) Histograms and Local Binary Pattern (LBP). We further investigated complex feature extractions such as Template Pattern and neural network models like RestNet101 and VGG16. With these extracted feature vectors, we applied principal component analysis (PCA) and T-distributed Stochastic Neighbor Embedding (t-SNE) Visualization method to achieve dimensionality reduction and avoid overfitting. We leveraged the key features identified and our judgment to develop classifier models, including Support Vector Machine (SVM), linear regression model and 1-Dimension Convolution model (Conv1D) to identify and determine the top features that represent the images effectively and achieve generalizability. Finally, we compared and contrasted our classification models to evaluate model performance in terms of model training efficiency and prediction accuracy.

1. Introduction

The German Traffic Sign Recognition Benchmark image dataset consists of 43 sign classes, including warning signs, speed limit and information signs, in almost 52,000 images. While the official German traffic signs have a consistent shape and color which comply with the European standards for traffic signs, the images in this dataset come with different sizes and resolutions. This data has an uneven distribution of images for each class (see Figure 1 below).

The 43 traffic sign classes in our dataset can be split into four main categories (see Figure 2 an example of traffic sign images):

- Warning signs - triangles with a thick red border and white background (e.g. pedestrian crossing)
- Regulation signs - these are compliance regulations, such as speed limits
- Guide signs - provide general information to drivers, such as slippery roads ahead
- Supplemental signs - these are meant to provide information for services such as gas stations.

[1] German Traffic Sign Recognition Benchmark

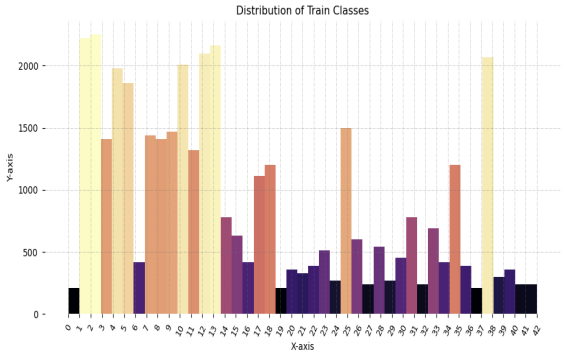


Figure 1: Distribution of train classes

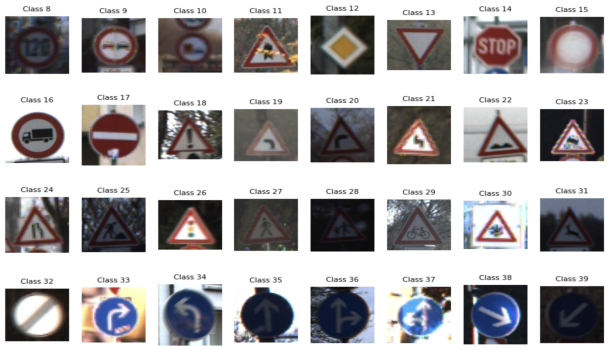


Figure 2: Images sample

We evaluated different feature extraction techniques, starting from basic feature extraction methods such as HOG, HSV, LBP, and Edge Detection and then applying advanced techniques such as Template Pattern Matching and pre-trained deep learning models such as ResNet101 and VGG16. To reduce feature dimensionality, select the key features and avoid overfitting, we applied PCA analysis and t-SNE visualization method and identified HOG, Template Pattern Matching and Hue as the top features that explain variances effectively. With the key features identified, we explored Logistic Regression, SVM and the fully connected layers using Conv1D models.

In our initial exploration, we got 98% of validation accuracy for the SVM Model by using HOG feature vectors. We tried different combinations of HOG, Template Pattern and other key features (e.g. Hue) and achieved an accuracy of 93% on the test data for the SVM model (using HOG features). We also explored the Logistic Regression model and Conv1D model and achieved 92% and 88% accuracy on the test data respectively (using HOG features).

This paper depicts our feature extraction and modeling framework used for German Traffic Signs image classification. Our approach and findings can apply to a variety of real world problems, potentially generalizable for other countries's traffic sign recognition.

We reviewed the CNN model work, a complex modeling framework in which the previous research on this topic was done by Stallkamp, Schlipsing, Salmen and Igel has been published in the paper "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition[2]".

2. Feature Extraction

Our framework started with performing exploratory data analysis to ground our initial understanding of the dataset. We then moved into feature extraction techniques, findings and feature selection.
Feature extraction is a part of the dimensionality reduction process. This process divides and reduces the raw data images to more manageable groups. Each pixel is a piece of data and the feature extraction

[2] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks, Available online 20 February 2012, ISSN 0893-6080, 10.1016/j.neunet.2012.02.016. (http://www.sciencedirect.com/science/article/pii/S0893608012000457) Keywords: Traffic sign recognition; Machine learning; Convolutional neural networks; Benchmarking.

process extracts only useful information from the image, which reduces the data amount but retains the pixels that describe the image characteristics.

We explored various image feature extraction techniques and algorithms on this dataset, with the goal to identify the key features for each traffic sign and use these features to develop multi-categorical classification models to recognize the signs and achieve generalizability.

2.1 Data Exploratory Analysis

As discussed in the Introduction section, this dataset is composed of 51,830 images of German Traffic Sign Recognition Benchmark from Kagle, which was originally published at the German Neuro-Informatic Institute[3] (Reliable ground-truth data due to semi-automatic annotation in CSV files). Physical traffic sign instances are unique within the dataset.

Data Structure Overview:

- 3 datasets: meta (43 sign images and labels), training (39209 real world images for 43 classes) and testing (12630 real world images for 43 classes)
- The training set is organized with one directory per class, each directory contains one CSV file with annotations ("GT-<ClassID>.csv") and the training images
- The training images are grouped by tracks; each track contains 30 images of one single physical traffic sign
- The number of images per class in the training set is unbalanced. For instance, there are less than 300 images for class 0 and more than 2,000 for classes 1 and 2 (to mention some)

Images:

- In PNG format with sizes ranging from 15x15 to 250x250 pixels. Not all images are squared.
- The traffic sign is not necessarily centered within the image
- Images contain a border of 10 % around the actual traffic sign (at least 5 pixels)
- Annotations are provided in CSV files containing the filename of the image, width, height, and the traffic sign bounding box coordinates

The only annotation fields used for our project were filename and class id.

2.2 Data Processing

From our initial data exploration, we normalized the new dataset by cropping and resizing to 64 x 64 using anti_aliasing=True and order=5 which was Bi-quintic and very slow. We noticed notable improvement after scaling to small images. All the new images were saved into parquet files.

After we extracted different features, the outputs were also moved into parquet files, which enabled efficiency in processing time, memory consumption and storage.

Some features had additional pre-processing performed on them. For VGG16 and RESNET101, the original images were scaled to 256x256 and center cropped back to 224x224 for those feature

[3] The Institut Für Neuroinformatik

calculations. For HOG and LBP, the images were converted to grayscale images and then had automatic histogram equalization applied before being processed by using HOG or LBP algorithm. The same processing steps are applied consistently for all images.
In our subsequent steps, based on our histogram bin count, we decided to randomly select 200 images for each of the 43 classes for a total of 8,600 images for balancing the images for each class in the PCA and t-SNE analysis. We split these images into train, validation and test dataset in the model development.

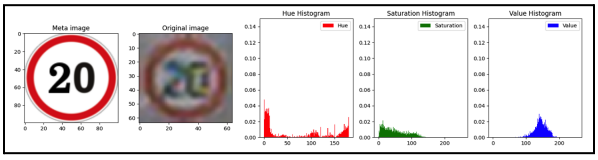
2.3 Hue, Saturation and Value (HSV) in Color Space

HSV depicts how colors appear under light and the properties are defined by expressing colors relative to white using polar coordinates.

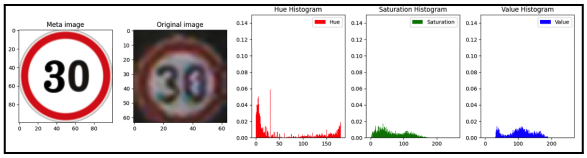
Based on images categories, color seems intuitively an important feature to be extracted. For instance, the stop sign is primarily red, while other signs are white, yellow, green or blue.

We converted the scaled 64x64 images from a range of 0 to 1 to 0 to 255 and then converted to HSV using cv2.cvtColor and the histograms were stored for later. Each class of street signs has distinct color makeup. While the colors present between classes can be similar, there are several examples of classes where color can contribute to differentiation between classes.

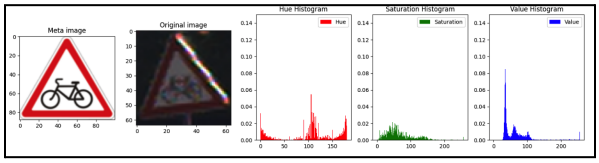
We extracted HSV histograms for the dataset and from the distribution we found similar distributions for groups of the same categories (see 20 and 30 speed limit HSV distribution for example). We also noted some outliers in which Hue histogram distribution was similar to other categories (for example, Class 0, Class 1 and Class 29). Figure 3 shows some samples of HSV results.



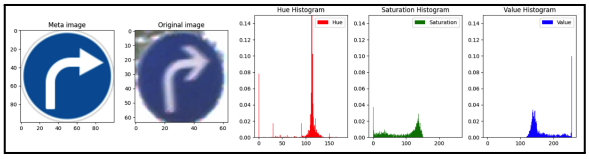
Class 0: 20 km/h Max limit Sign



Class 1: 30 km/h Max limit Sign



Class 29: Bicycle crossing Sign



Class 33: Turn right ahead Sign

Figure 3: HSV histograms

2.4 Histogram of Oriented Gradients (HOG)

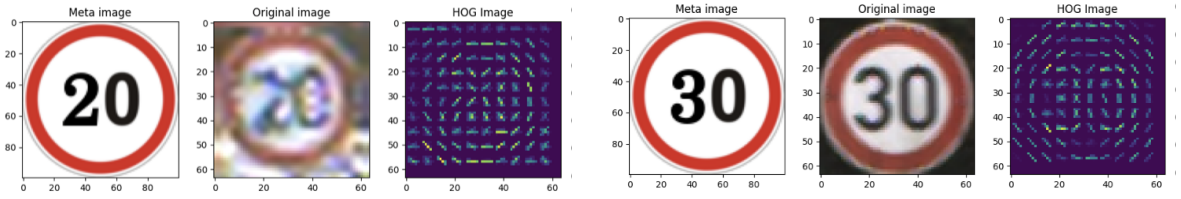
The HOG technique counts occurrences of gradient orientation in localized portions of an image and the features describe the structure or the shape of an object.

Traffic signs are planar objects with hard edges and regular patterns. Intuitively, the HOG feature seems to be a good fit for our different shapes in traffic sign images. HOG is later proven out to be one of the top features in our models section.

The HOG features were generated from the scaled 64x64 input image. After many iterations we chose the following parameters: Orientations were 9, block_norm='L2-Hys', Pixels_per_cell (6,6) cells per block of (2,2).

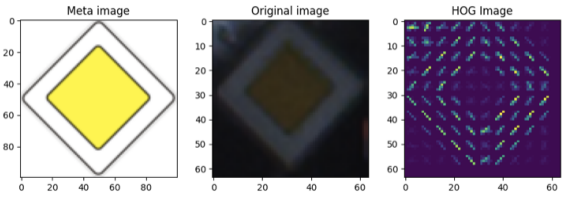
We found HOG as a more effective descriptor than the canny edge because HOG uses magnitude and angle of the gradient to compute the features. HOG not only detected the edges, but was also helpful to add a distinction where HSV features returned some confusion between different traffic signs.

Figure 4 shows the HOG representation of two classes where HSV histograms shared some similarities. Figure 5 shows a different traffic sign where the shape is clearly identified



Class 0: 20 km/h Max limit Sign Class 1: 30 km/h Max limit Sign

Figure 4: HOG Features (Class 0 and Class 1)



Class 12: Right of way at all intersections

Figure 5: HOG for Class 12

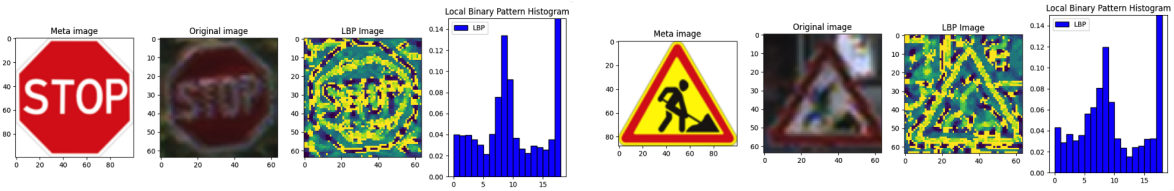
2.5 Local Binary Patterns (LBP)

The Local Binary Pattern (LBP) method differentiates tiny nuances in texture and topography and identifies key features with which we can then distinguish between images of the same type. For this dataset, we are not interested in topography or even texture, however the power of this simplistic algorithm was worth our exploration. The prior research work also found that when LBP is combined with a HOG descriptor, it further improves the image detection performance".[4]

[4] Local Binary Patterns

We also started the LBP feature extraction process with the 64x64 scaled images. We then converted them from float in range 0 to 1 to 0 to 255, followed by converting to grayscale using cv2.cvtColor function. The histogram of the gray scale image was then stretched and the local binary pattern feature applied to the resulting input image. The final LBP feature settings were radius=1, n_points=16, method='default'.

The objective of LBP is to encode geometric features of an image by detecting edges, corners, raised or flat areas (which in our case, some images may be partially blocked by branch trees) and hard lines. This allows us to generate a feature vector representation of an image. Figure 6 shows samples extracting LBP images and histograms.



Class 14: Stop Sign Class 25: Roadworks Sign

Figure 6: LBP Image and Histogram for Classes 14 and 25

2.6 Template Pattern Matching

One simple concept, yet advanced feature extraction method is Template Pattern Matching, where a template image (usually a subimage) contains the shape of the target object. Because we have meta images, we decided to take this as an advantage and use them as templates and evaluate this feature.

This method centers the template on an image point and counts how many points in the template match those in the image. The process is repeated for the entire image and the point which has the best match (maximum count) is selected to be the point where the shape from the template lies within the image.

To perform template pattern matching, we applied several transformations. We converted each target image to grayscale using cv2.cvtColor function and then padded with 4 rows of black pixels on all sides, to ensure that the template can be shifted around the target and find an optimal match. This results in a 9x9 feature vector for each meta image, which are all then flattened and concatenated and returned as a single feature vector for each target image.

Figures 7 and 8 provide an example of Template Pattern Matching in grayscale.

Figure 7: Template Matching sample for Class 1 (30 km/h Max limit Sign)

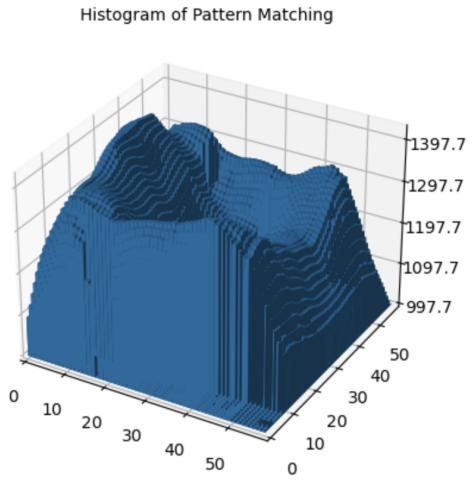


Figure 8: Template Matching Histogram sample for Class 1 (30 km/h Max limit Sign)

We believe our dataset could also benefit from multi-scale template matching. Many of our input images vary in the percentage that the sign occupies, which makes our current approach less than optimal. Multiscale pattern matching could also help us filter out the noisy backgrounds and improve our model generalizability. This will be an area for consideration of future works.

2.7 ResNet101 and VGG16

Our final set of advanced feature extractions came from the feature embeddings from pre-trained neural network models ResNet101 and VGG16.

We processed the ResNet and VGG features in the same fashion. The target images are scaled to 256x256 then center cropped to 224x224. We loaded the networks from Tensorflow with the ImageNet weights and include_top=False. The features are then extracted from the output layer of the model and stored as a 1-dimensional list for later use in our model.

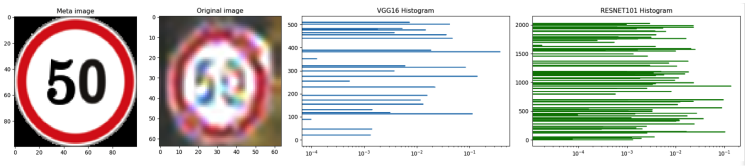
2.7.1. VGG16. From the Large Scale Image Recognition competition, VGG 16 was proposed by Simonyan and Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper Very Deep Convolutional Networks for Large-Scale Image Recognition[5].

[5] Very Deep Convolutional Networks for Large-Scale Image Recognition

VGG16 is a CNN model of 3 fully connected layers and 13 convolutional layers, (16 total). It is a simple yet powerful pre-trained model where features are extracted from the convolutional layers, flattened and passed further down to the fully connected layers to predict (classify). We further evaluated VGG16 as an additional technique for feature extraction.

2.7.2 ResNet101. From the family of the ResNets neural networks, ResNet101 solves the decay of information through time, specifically the vanishing gradient problem. When the network is as deep as the VGG16, the loss function shrinks until eventually goes to zero and there are no more learnings. ResNet101 implements a similar architecture like VGG16 with the gradients flowing directly through the skip connections backwards from later layers to initial filters as described by He, Zhang, Ren, and Sun in the research paper Deep Residual Learning for Image Recognition[6].

We used the embeddings (features) from these two models and compared with the rest of the extracted features to determine which ones will be used for our classification model. Figure 9 shows an example of histograms for image class id = 2 plotted in log10 scale.



Class 2: 50 km/h Max limit Sign
Figure 9: VGG16 and RESNET Histograms

All features extracted during this process were stored in parquet files (train_features.parquet file for all training data and test_features.parquet file for test data).

2.8 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a common statistical technique for analyzing large datasets and reducing a high number of dimensions or features while preserving the maximum amount of information. This method linearly transforms the data into a new coordinate system where most of the variation in the data can be described with fewer dimensions than the original dimensions or features.

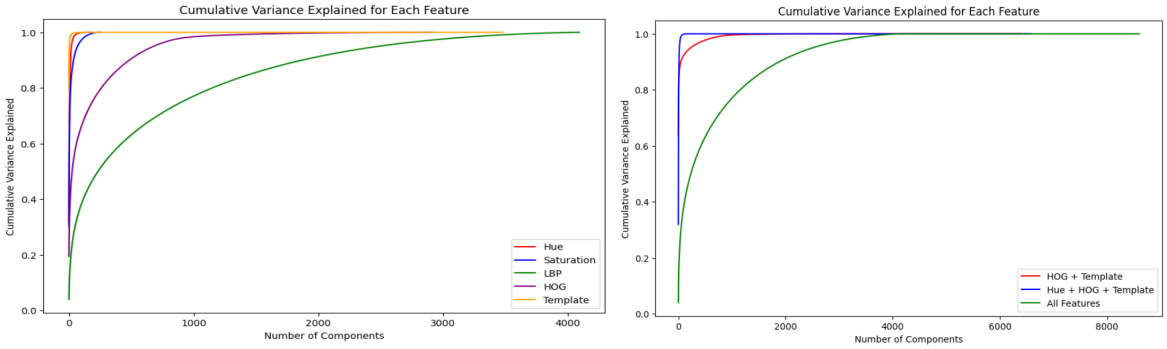
For the feature extraction methods described in the Feature Extraction section, we decided to further evaluate Hue, Saturation, HOG, LBP and Template Pattern Matching method and a subset of these combinations (see Table 1 below), by performing PCA analysis to reduce dimensionality and avoid overfitting.

[6] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in CVPR, 2016

Feature Type	Feature Name	Feature Size
Individual	Hue	180
Individual	Saturation	256
Individual	LBP	4,096
Individual	HOG	2,916
Individual	Template	3,483
Combined	HOG + Template	6,399
Combined	HOG + Template	6,579
Combined	All Features	10,931

Table 1: List of Feature Vectors for PCA analysis

We retrieved individual feature vectors (HOG, Hue, Saturation, LBP and Template Pattern Matching) from the train_features.parquet file (created in the Feature Extraction process). We combined HOG and Template to create a subset of combined features, and similarly created the subset of Hue, HOG and Template combined features. We then used sklearn PCA function to understand and plot the total variance explanatory power of each feature vector and the combined feature vectors. See Figure 9 below.



Subplot 1: PCA for Each Individual Feature Vector

Subplot 2: PCA for the subsets and total Combined Features

Figure 9: Explained variance plots of the principal components from feature vectors

Among all individual features (Figure 9 subplot 1), PCA on HOG features (the purple line) achieved the best balance of high explanatory power and efficiency.

In Figure 9 subplot 2, we compared two subsets of combined features - HOG+Template (red line) and Hue+HOG+Template (blue line) to the combination of all features (green line). We can see the subsets are more efficient in explaining the variance than using all features.

2.9 T-distributed Stochastic Neighbor Embedding (t-SNE) Visualization

Similar to PCA, T-distributed Stochastic Neighbor Embedding (t-SNE) is a powerful technique for dimensionality reduction. Unlike PCA which is a linear method and preserves the global structure, t-SNE is a nonlinear dimensionality reduction method and tries to preserve the local structure (cluster) of data. T-SNE is mainly designed for visualization and is less sensitive to the ordering of the data points.

We used the sklearn TSNE function for this analysis. Figure 10 shows the t-SNE visualization charts for each individual feature vector, the subsets of combined features and the total features.

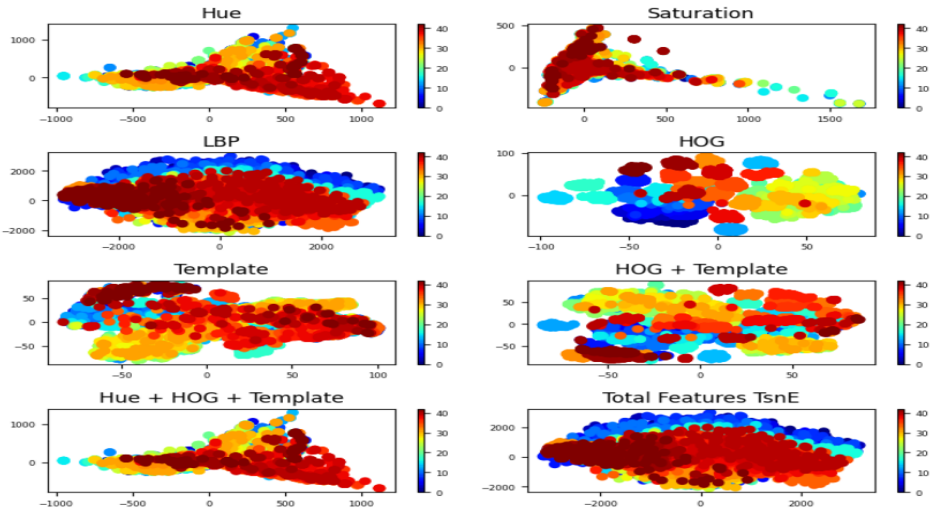


Figure 10: t-SNE visualization for the first 2 components of various features

From the t-SNE charts, among the individual feature vectors, HOG performed the best in creating clusters and differentiating among the classes. The subset combining HOG and Template also performed well.

3. Classification

Based on the PCA and T-SNE analysis described in the above section, we selected three key features (HOG, Hue and Template) to process them through different models for classification. We also included additional features (e.g. LBP, Saturation) in the initial model exploratory runs.

3.1 Results

We trained 3 classification models: Support Vector Machine (SVC), Logistic Regression, and 1-D Convolution (Conv1d) layer model.

For the model development process, we used the train_features.parquet file (created in the Feature Extraction process) to select a subset of the dataset from the 52k training data images. We randomly selected 200 images per class (200 times 43, or 8,600 images) to balance the class presentation in the training data. We splitted the 8,600 training images into train and validation data: (1) train set, consisting of 80% (6,880 images); and (2) validation set, consisting of 20% of the training set (1,720 images).

For model inference and generalization evaluation, we used the test_features.parquet file (created for the test dataset in the Feature Extraction process) and selected a subset data from the 12K test images - we selected 60 images per class for a total of 2,580 test images.

Table 2 below shows the accuracy result of our models with our initial hyperparameters (before optimizing the settings), using HOG features. For Conv1D, we used the initial hyperparameter setting as a starting point (batch size = 32, epochs = 10, kernel size = [21, 5], filter size = [6, 16), dense layers = [120, 84], learning rate = 0.001, drop out rate = 0.5, optimizer = Adam, etc.). Table 3 applied the same initial hyperparameter settings, with a subset of the combined features.

Model	Train Accuracy	Validation Accuracy	Test Accuracy
SVC	1.000	0.9862	0.9300
Logistic Regression	1.000	0.9792	0.9239
Conv1D	0.9901	0.9390	0.8729

Features Used: *HOG Features*

Table 2: Model Performance on train, validation and test dataset (no hyperparameter optimization)

Model	Train Accuracy	Validation Accuracy	Test Accuracy
SVC (HOG + Template + Hue)	1.000	0.9917	0.9429
Conv1D (HOG + Template)	0.7536	0.8097	0.6818

Features Used: *A subset of combined features*

Table 3: Model Performance on train, validation and test dataset (no hyperparameter optimization)

We then performed a hyperparameter optimization process for the SVC and Conv1d model. For the SVC model, we iterated several key settings and all 255 combinations of features (Hue, Saturation, HOG, Template and LBP, etc.) and determined the optimized parameters settings.

- "svc__kernel": "linear"
- "svc__C": 0.005
- "svc__decision_function_shape": "ovo"
- "svc__probability": True

For the Conv1D model, we iterated the hyperparameter settings and obtained the optimal settings below while keeping kernel size and learning rate the same as the initial settings (kernel size = [21, 5], learning rate = 0.001).

- Filter Size = [12, 16]
- Dense Layer = [84, 64]
- Dropout Rate = 0.3
- Epochs = 20

For all three models, we observed HOG is the most effective and efficient feature of the individual feature vectors (see more details in each model's sub section below). Table 4 and 5 below summarizes the model results after we optimized the hyperparameters for SVC and Conv1D models to balance accuracy and efficiency.

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Training Time
SVC	1.000	0.9280	0.9280	16.0s
Conv1D	0.9922	0.9500	0.8849	80.0s

Features Used: *HOG Features*

Table 4: Model Performance on train, validation and test dataset (after hyperparameter optimization)

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Training Time
SVC (HOG + Template + Hue)	1.000	0.9390	0.9360	33.6s
Conv1D (HOG + Template)	0.9887	0.9494	0.8764	361.7s

Features Used: *A subset of combined features*

Table 5: Model Performance on train, validation and test dataset (after hyperparameter optimization)

From the above results, we observed that SVCl achieves better performance than Conv1D for both efficiency and accuracy.

Support Vector Machine

Our first model is a support vector machine from sklearn called (SVC). SVC algorithm aims to find the best possible line, or decision boundary to separate different classes. The key objective of SVC is to transform the input data into a higher-dimensional feature space, which makes it easier to find a linear separation or classify the classes. Our final parameters for this model after a hyperparameter optimization were the following:

C: 0.005; Kernel: Linear Decision Function shape One vs. One, Probability: True

Our most efficient vector (HOG) produced an accuracy of 98.0% for our test data, see Table 4 below. Overall this model performed very well, indicating that our data can be accurately modeled using a linear predictor. Because we are trying to maximize the margin between our classes. Outliers in our data can have a large influence on the decision boundary. We could minimize this by further pre-processing of our input data. One such feature that was beyond our scope for this project would be homography since we are dealing with inherently 2D data for traffic signs. Another would be a contrast equalization due to the variety of lighting conditions in the original dataset.

The performance of the various feature vectors for the SVC (linear) model using the validation dataset. See the Appendix for additional results with the SVC model and confusion matrices.

Feature	Validation Accuracy
Hue	32.45%

HOG	97.95%
LBP	89.92%
Template	92.30%

Table 4: SVM Model Accuracy

Logistic Regression. Our second model, Logistic Regression is a simple but powerful approach for modeling a predictive relationship between dependent and independent variables. We decided to create this model using the extracted features as input to the model.

During our experiment, we tried different hyper-parameters (epochs, learning rate, batch size) and the model demonstrated an impressive accuracy of 97.7% for the HOG features vector; followed by the Template features vector, with an accuracy of 90.7%. See Table 5 for full Precision, Recall, and F1-Score results. See appendix for confusion matrices on HOG and Template for our Logistic Regression model.

Feature	Precision	Recall	F1-Score
Hue	0.297199	0.237209	0.214802
Saturation	0.244971	0.170349	0.162789
HOG	**0.977197**	**0.976163**	**0.976375**
Template	**0.907834**	**0.904070**	**0.904310**
VGG16	0.029873	0.035465	0.010238
ResNet101	0.716369	0.600000	0.610147

Table 5: Linear Regression Precision/Recall/F1-Score

Conv1d NN. Our third model is a 1-dimensional neural network of connected layers using the Conv1d layer (the kernel moves in one direction). Conv1D layer applies filters/weights to the 1-D array input features (multiply the weights to the input features) to create the feature map (output array). The neural network has artificial neurons which calculate the weighted sum of the inputs and return an action map.

Feature	Precision	Recall	F1-Score
HOG	0.89	0.88	0.87
Template	0.77	0.73	0.73
Hue	0.17	0.16	0.16
LBP	0.81	0.80	0.80

Table 6: Conv1D Model Precision/Recall/F1-Score

3.2 Model Performance

Support Vector Machine: The linear SVC model performed well with our data. To better understand the performance of the model. We chose to evaluate the time to train and predict times of the model

without probability enabled for every combination of feature vectors to understand the performance of the model with different sized feature vectors.

For our most accurate feature vectors the total training and prediction time for the test data were between 33.6s for 6579 features. The second most accurate vector required 97.6s for 10675 features.

The most efficient vectors required 14.6s for training and prediction of 2916 features. The second most efficient vector required 16.6 seconds for 2096 features. See the appendix for charts.

Logistic Regression: Our Logistic Regression model performed consistently on HOG and Template when compared to the SVM model. As seen in table 3, It performed extremely well on validation data, however the performance dropped significantly on test data. For the HOG features vector, our accuracy for validation is **97.7%**; however it dropped to **75%** with test data; followed by the Template features, with an accuracy of **90.7%** and **68%** for validation and test data respectively.

Feature	Precision		Recall		F1-Score	
	Validation	Test	Validation	Test	Validation	Test
Hue	0.297199	0.123820	0.237209	0.107522	0.214802	0.096491
Saturation	0.244971	0.051136	0.170349	0.063737	0.162789	0.044100
HOG	**0.977197**	**0.750167**	**0.976163**	**0.803800**	**0.976375**	**0.769758**
Template	**0.907834**	**0.680754**	**0.904070**	**0.725020**	**0.904310**	**0.692439**
VGG16	0.029873	0.038994	0.035465	0.038717	0.010238	0.013093
ResNet101	0.716369	0.563943	0.600000	0.473476	0.610147	0.459978

Table 7: Logistic Regression Precision/Recall/F1-Score

Conv1d Model: We iterated the key hyperparameters using different combinations of the settings below, for HOG and the combined HOG and Template features to evaluate the optimal hyperparameter settings.

- Filter Size = [[6, 16], [12, 16]]
- Kernel Size = [21, 5]
- Dense Layer = [[120, 84], [84, 64]]
- Dropout Rate = [0.3, 0.5]
- Learning Rate = 0.001

Table 8 reflects the most efficient and accurate results using the optimal settings. HOG feature outperforms the combination of HOG and Template features in efficiency and accuracy. As discussed in Section 2.6 Template Matching feature extraction above, it's likely that our template features are not optimal and could benefit from multi-scale template matching, and hence further improve the model performance. This is a consideration for future works.

Feature	Epochs	Filter Size	Kernel Size	Dense Layer	Dropout	Train Accuracy	Val Accuracy	Test Accuracy	Train Time	Train Time
HOG	10	[6, 16]	[21, 5]	[84, 64]	0.3	0.9923	0.9500	0.8849	80	80s
HOG	20	[12, 16]	[21, 5]	[120, 84]	0.3	0.9820	0.9465	0.8841	74	74s
HOG+Template	20	[12, 16]	[21, 5]	[120, 84]	0.3	0.9744	0.9384	0.8601	182	182s
HOG+Template	20	[12, 16]	[21, 5]	[84, 64]	0.3	0.9887	0.9494	0.8764	362	362s

Table 8: Conv1D Model Performance - using optimized hyperparameter settings

3.3 Confusion Matrix

Overall, our three models performed consistently well using the HOG features. Template features also performed reasonably well, especially for SVC and Logistic Regression models. The confusion matrices are fairly large for the 43 classes for our models. We included the HOG Feature Confusion Matrix for the three models in Figure 11-13 below. The remaining confusion matrices are located in the Appendix section and are full sized for clarity.

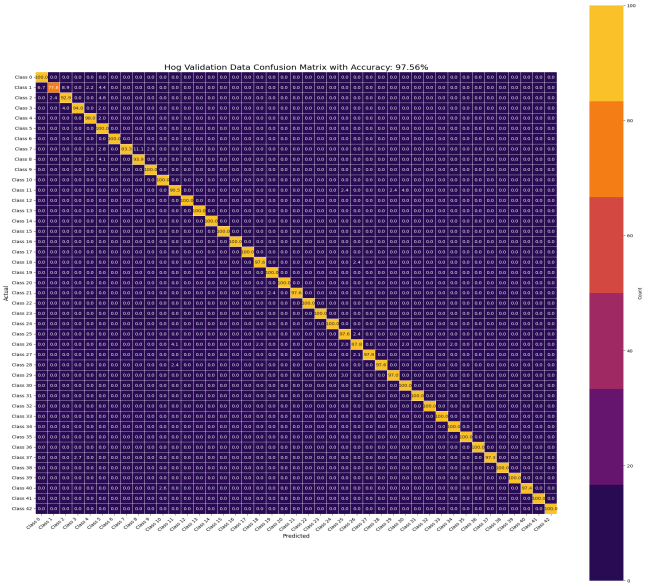


Figure 11: SVC Validation Confusion Matrix (*HOG*)

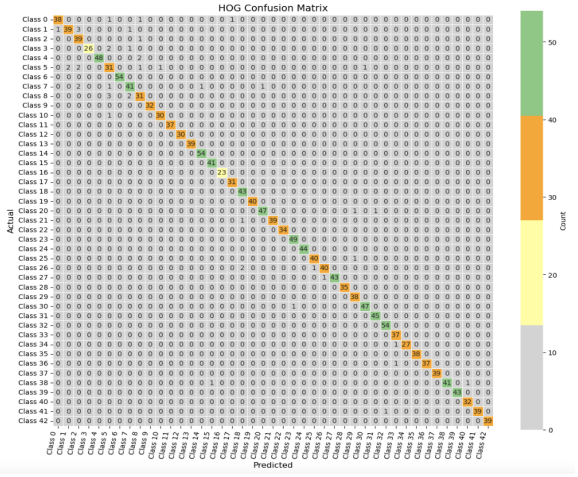


Figure 12: Logistic Regression Confusion Matrix (*HOG*)

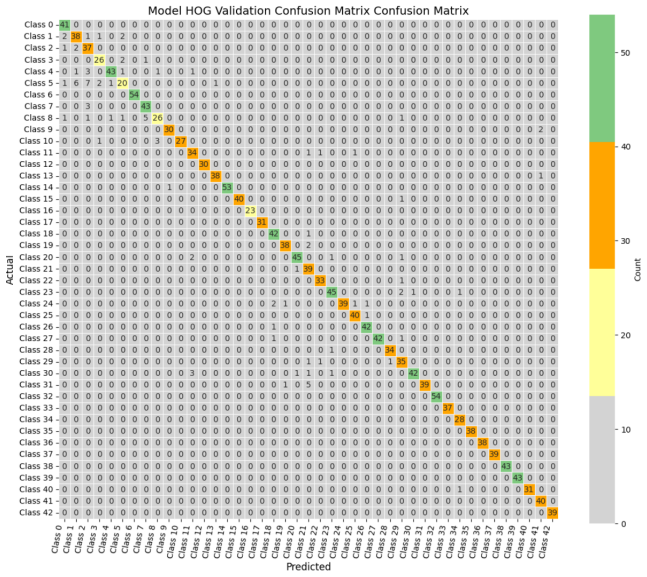


Figure 13: Conv1D Validation Confusion Matrix (*HOG*)

4. Generalizability

We followed common practices splitting our data into train, validation and test. As discussed in the Result section, we used three datasets: (1) train set, consisting of 80% (6,880 images); and (2) validation set, consisting of 20% of the training set (1,720 images).

We used the train set for model training and used the validation set to evaluate performance during the model development stage. We used the test set only to attest the generalizability of our model. Our models generalized well based on consistent performance between the validation and test datasets.

Overfitting:

As discussed in Section 2.8 PCA and 2.9 t-SNE, we used these two methods to identify key features that explain the maximum amount (95%) of variance and reduce the number of dimensions and features, to avoid model overfitting. We identified HOG and Template as the top two features for this dataset. For the Conv1D model, we used dropout rate and early stopping function (based on validation accuracy) to reduce overfitting.

SVC and Linear Regression models (validation accuracy at 98% and test accuracy 93%) generalized better than the Conv1D model (validation accuracy 95% and test accuracy 88%), using HOG features. One potential reason could be neural networks require a much larger dataset and more advanced features for model training than what are needed by the linear models like SVC and Logistic Regression. Improving complex Template feature extraction by using multi-scale template matching could help Conv1D to improve model performance and generalization.

Hyperparameter tuning: We used iterative grid search to refine our hyperparameters to obtain the most efficient, accurate and generalizable models. This search process uses various combinations of parameters and evaluates the model performance for every model version to optimize the parameter settings.

5. Efficiency vs. Accuracy

We trained and tested all three models and found that the Logistic Regression model took the longest. See below for details.

Support Vector Machine. For the SVC linear model we passed all 255 possible feature vector combinations to understand the features which produced the best accuracy and the most efficient vector. Our most efficient model used the HOG feature vector with 2916 Features and produced an accuracy of 97.56% for validation data. Our most accurate feature vector consisted of Hue, Value, HOG, LBP and Template which resulted in 1091 features and an accuracy of 98.8% for validation data. The HOG feature has proven to be our most efficient and predictive feature for this dataset.

Logistic Regression. For the Logistic Regression, we recorded the training and prediction times across all features with all 43 classes and full training and testing data.

Table 9 shows the elapsed time recorded for training data using a Macbook Pro 1.4 GHz Quad-Core Intel Core i5 and 8 GB 2133 MHz.

Our top feature HOG took only 52 seconds to complete training for an accuracy of 97.7% and took less than 1 second for predicting all test data. The Template feature took 18 minutes to get an accuracy of 90.8%; while our worst feature under Logistic Regression was VGG16 with 119 minutes and 3% of accuracy.

Feature	Accuracy	Efficiency
Hue	29.7%	72 min, 15 sec
Saturation	24.5%	39 min, 17 sec
HOG	97.7%	0 min, 52 sec
Template	90.8%	18 min, 26 sec
VGG16	3%	119 min, 16 sec
ResNet101	71.6%	129 min, 30 sec

Table 9. Efficiency on Logistic Regression

Conv1d Model: See discussion in Section 3.2 Model Performance under **Conv1d Model.** See the full details of the efficiency vs. accuracy results in Appendix 3 Conv1D Model Performance.

6. Conclusion

In summary, our evaluated models with the selected features (HOG, Template, and Hue) presented a consistent behavior and an impressive accuracy with a good efficiency. We decided to select the SVC model due to accuracy, efficacy and efficiency.

Compared to the CNN models presented by Stallkamp, Schlipsing, Salmen and Igel in "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition [1]", where most of the machine learning algorithms achieved a correct recognition rate of more than 95%, with the committee of CNNs reaching near-perfect accuracy, outperforming the human test persons; our SVC model reached an accuracy of 98% on validation data and 93% on test data just using HOG and slightly higher using the aggregated features (HOG + Template + Hue). Therefore, with a simpler model, we surpassed most of the machine learning algorithms.

7. References

[1] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks, Available online 20 February 2012, ISSN 0893-6080, 10.1016/j.neunet.2012.02.016.
(http://www.sciencedirect.com/science/article/pii/S0893608012000457) Keywords: Traffic sign recognition; Machine learning; Convolutional neural networks; Benchmarking.

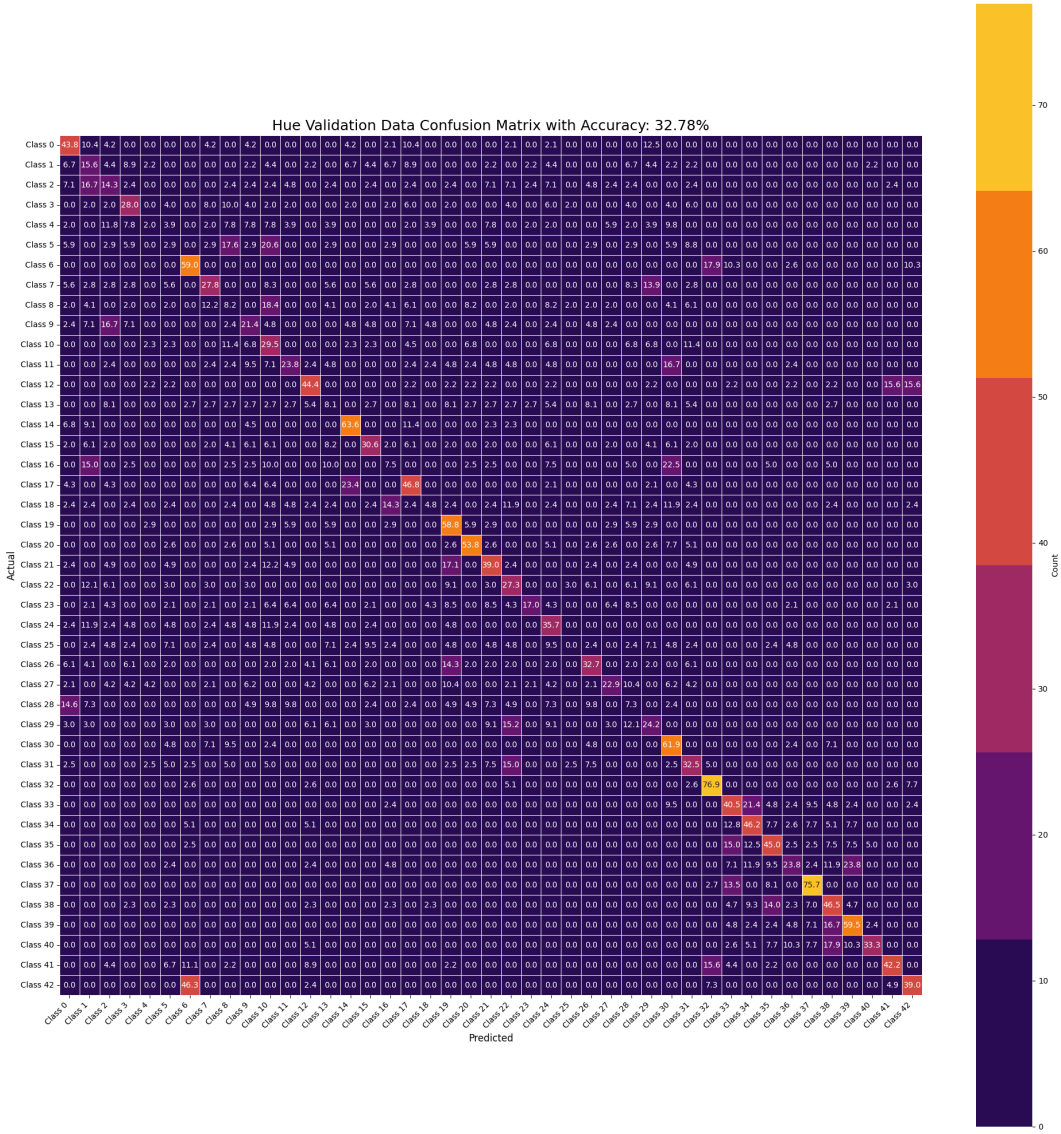
[2] Richard Szeliski
Computer Vision
Algorithms and Applications 2nd Edition
Springer AG 2022

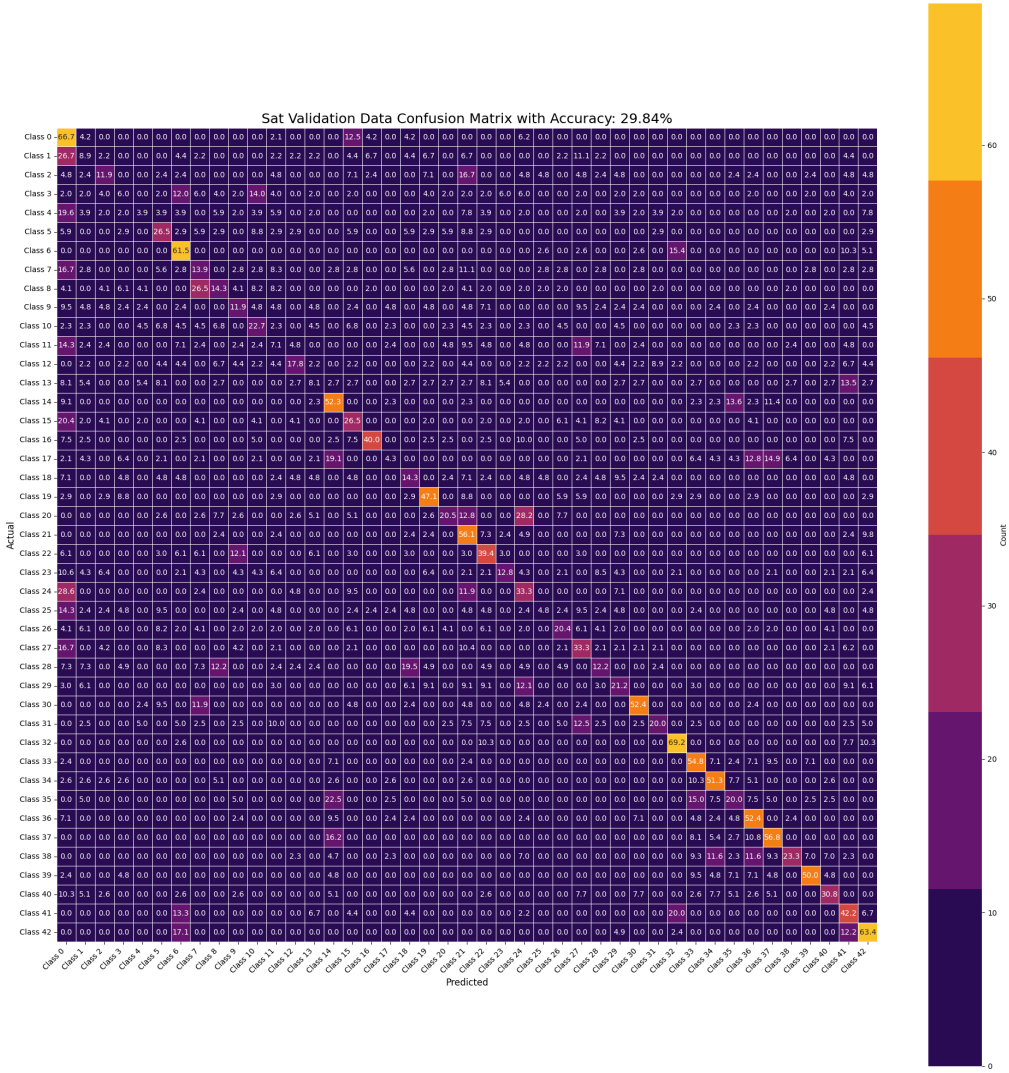
[3] Mark S. Nixon, Alberto S. Aguado
Feature Extraction and Image Processing for Computer Vision 4th Edition
Academic Press 2020

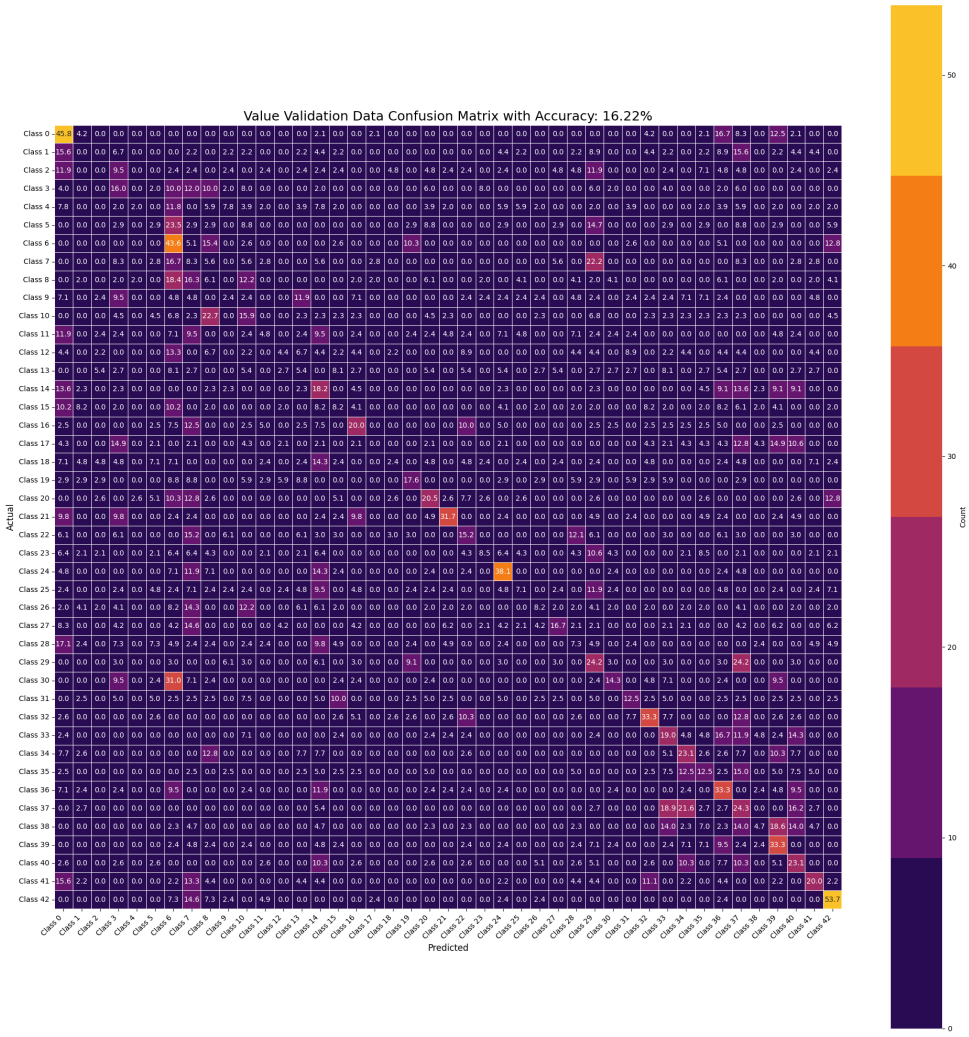
Appendix 1: Confusion Matrices for Individual Feature Vectors

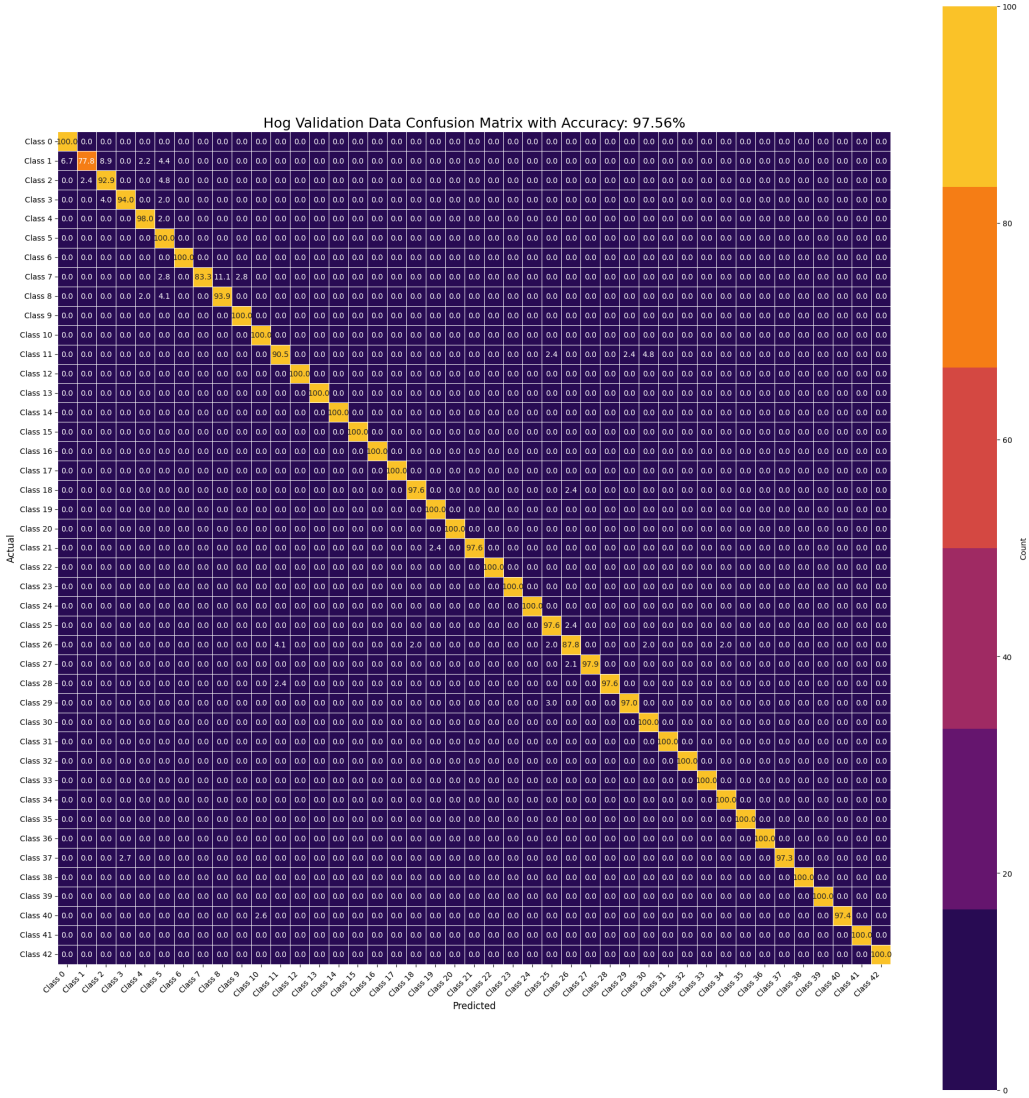
Confusion Matrices for SVC model

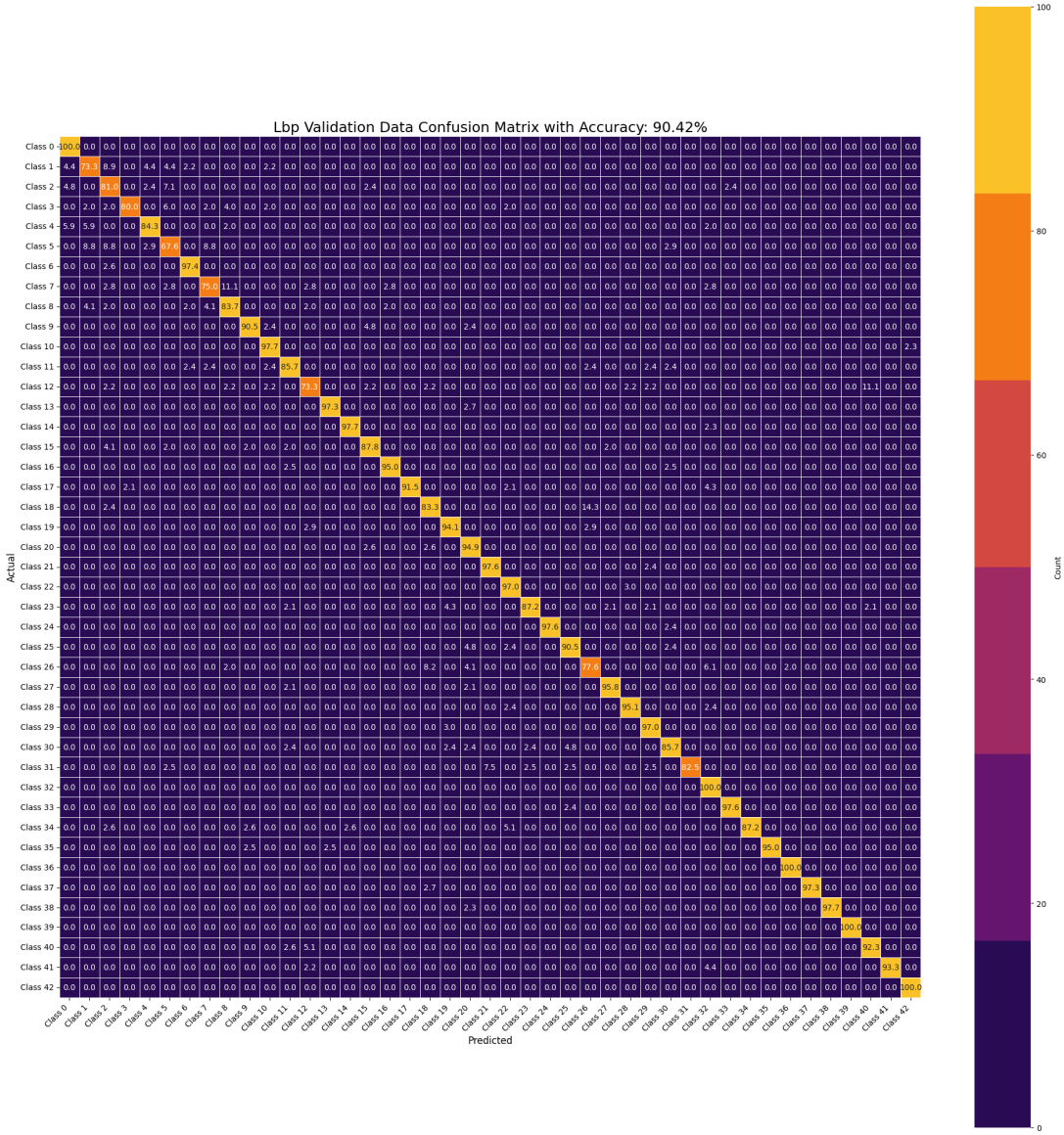
These plots are quite large. We've done the best we can to make them visible within the report. Please see the individual images on GitHub for a more clear representation of the information.

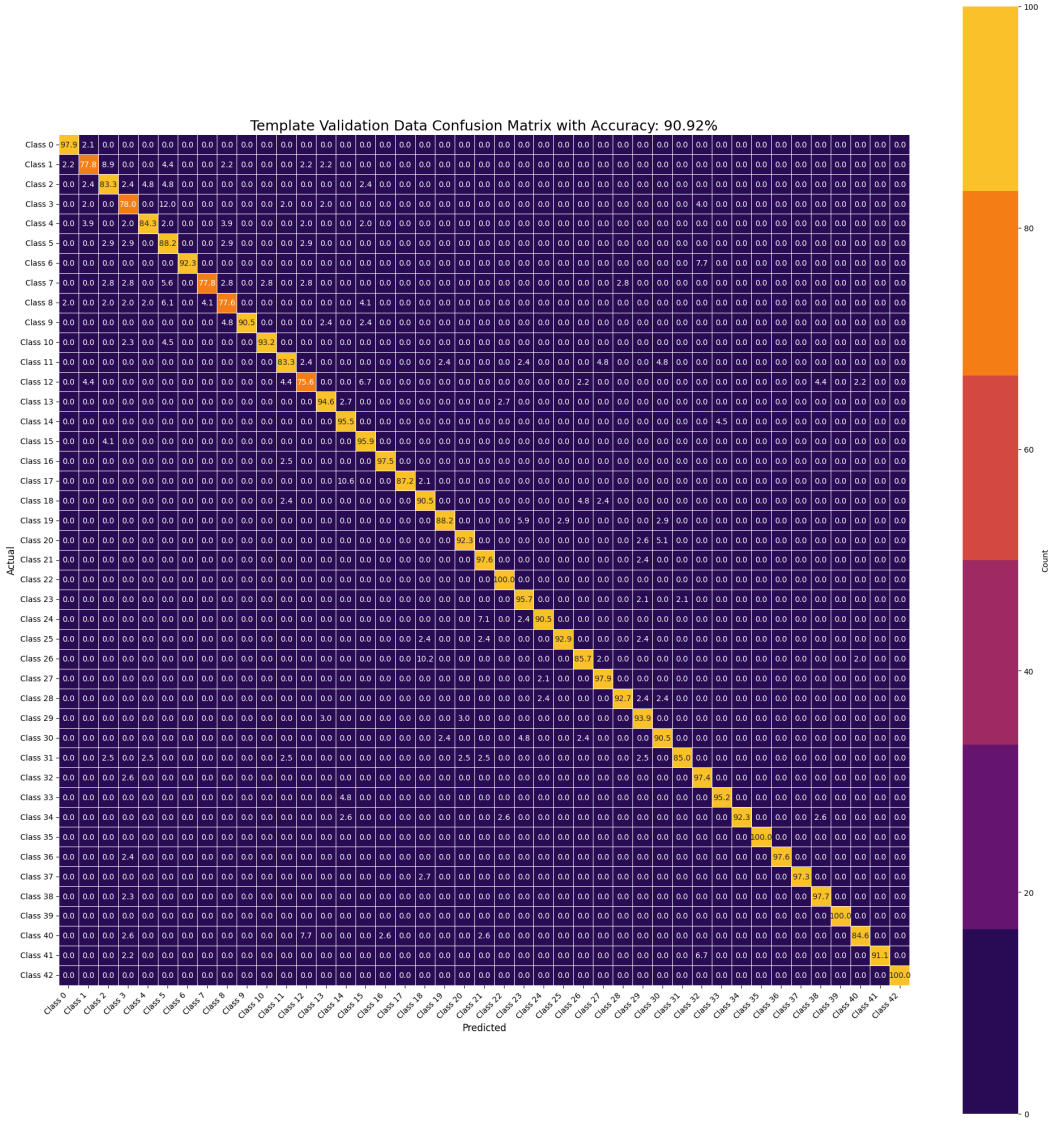


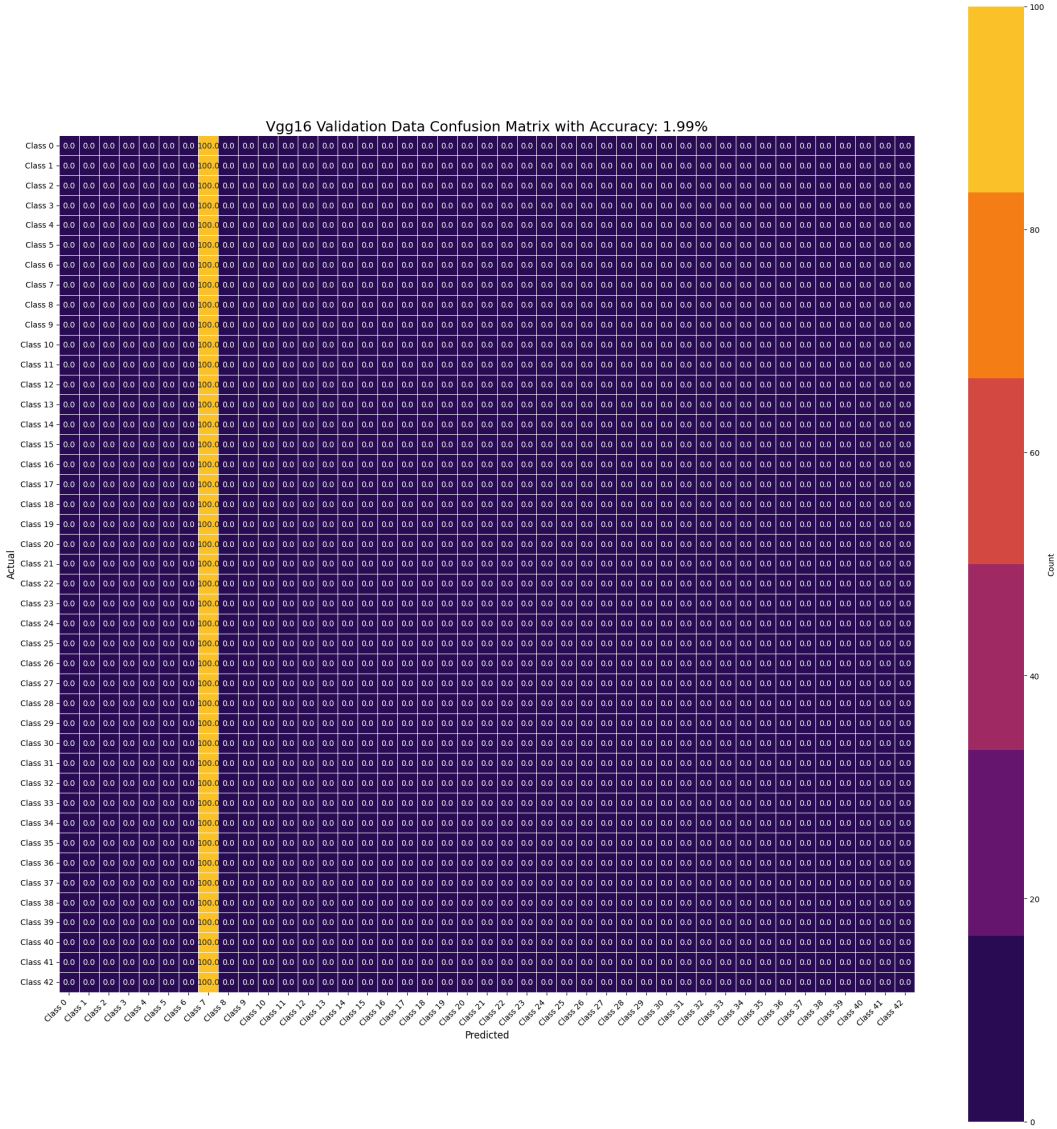

Sat Validation Data Confusion Matrix with Accuracy: 29.84%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
10
20
30
40
50
60

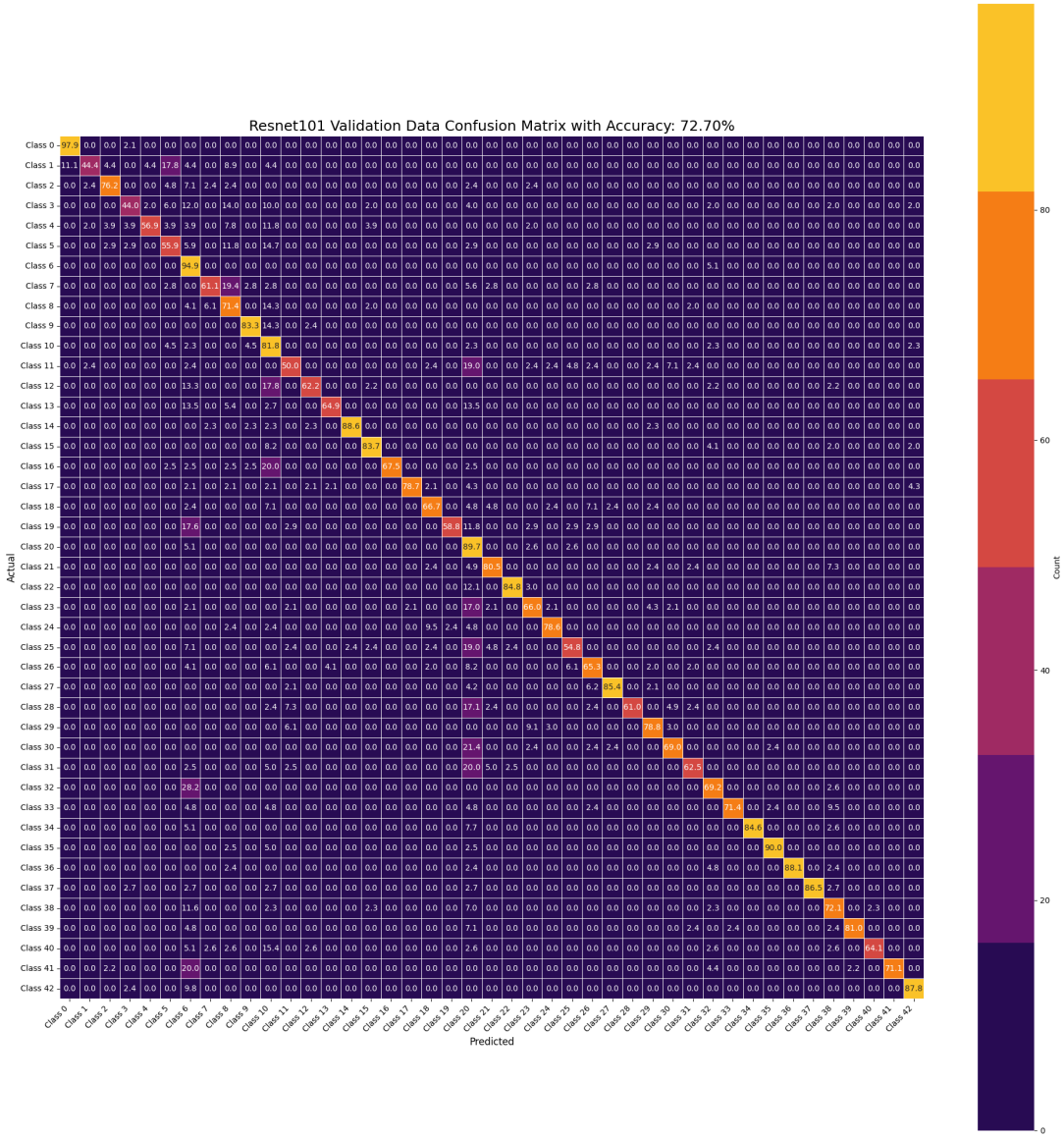

Value Validation Data Confusion Matrix with Accuracy: 16.22%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
10
20
30
40
50


Hog Validation Data Confusion Matrix with Accuracy: 97.56%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
100
80
60
40
20
0


Lbp Validation Data Confusion Matrix with Accuracy: 90.42%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
20
40
60
80
100

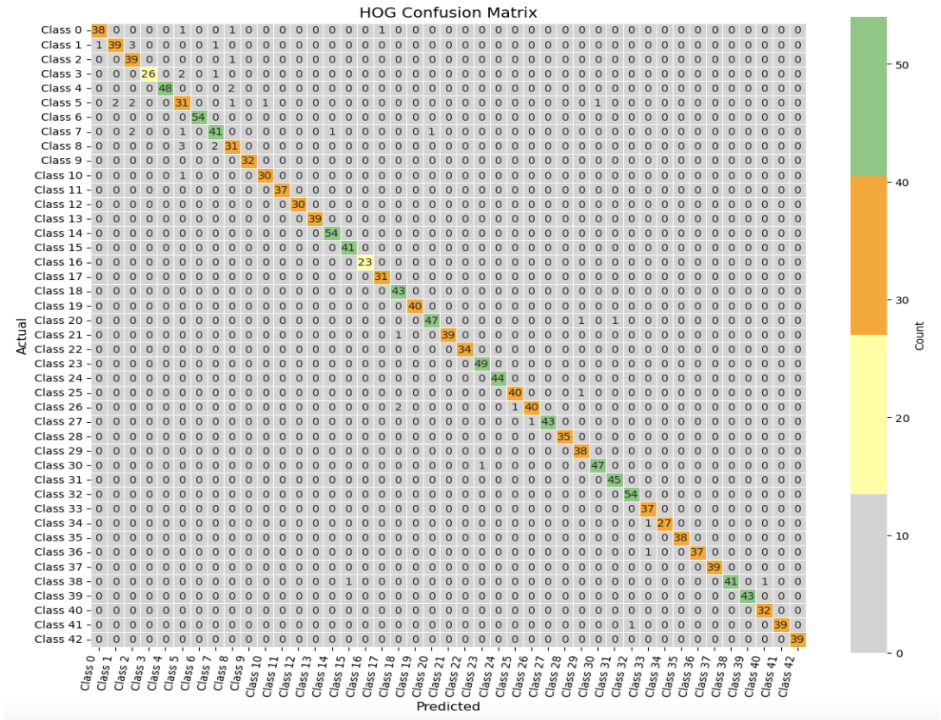

Template Validation Data Confusion Matrix with Accuracy: 90.92%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
20
40
60
80
100

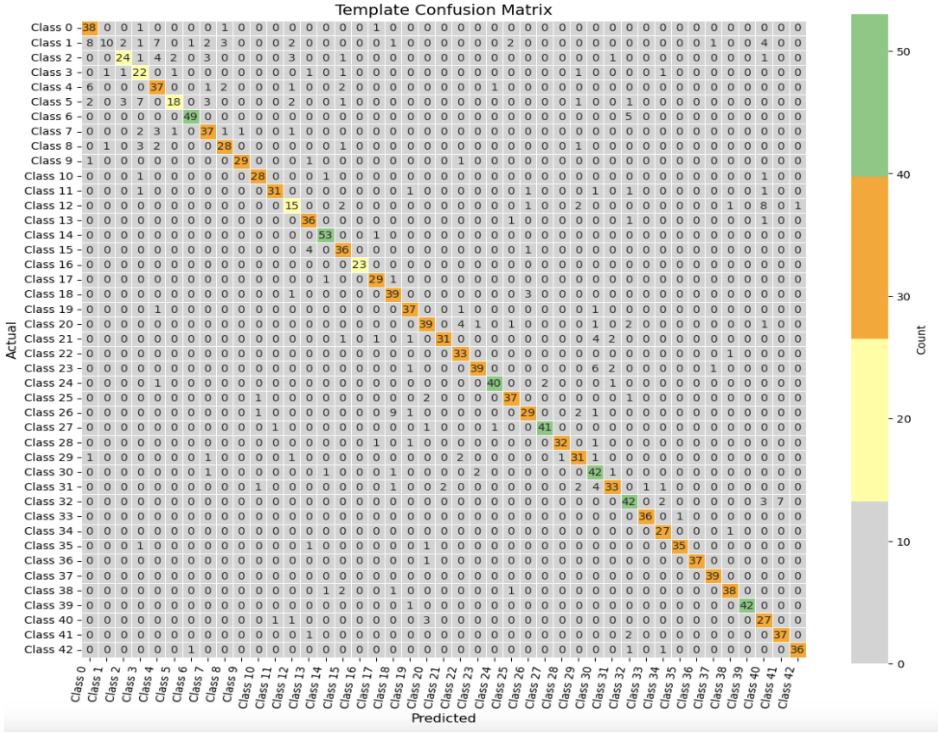

Vgg16 Validation Data Confusion Matrix with Accuracy: 1.99%
Actual
Predicted
Count
100
80
60
40
20
0
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42



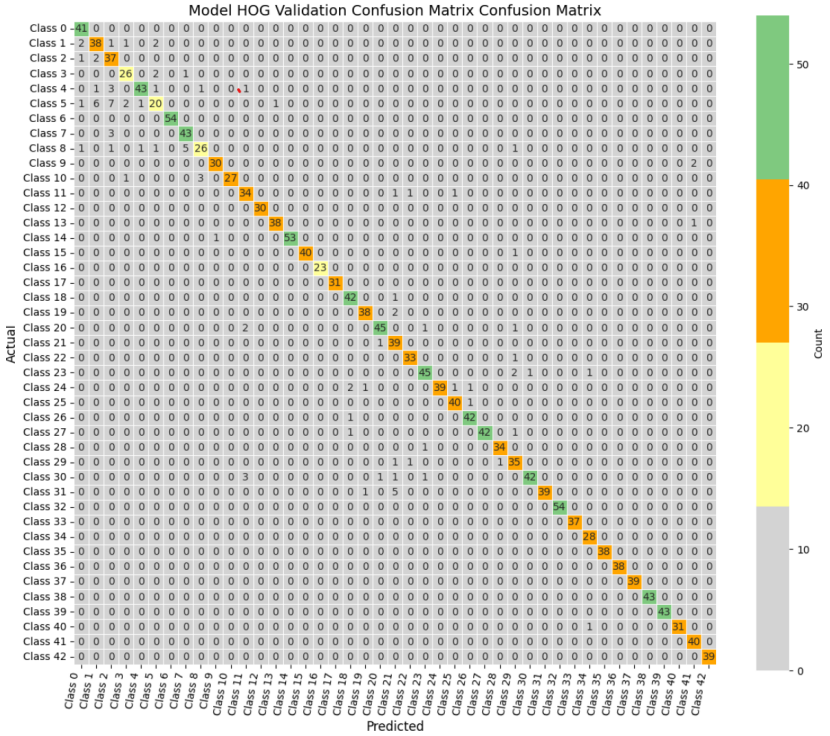
Confusion Matrices for Logistic Regression model.

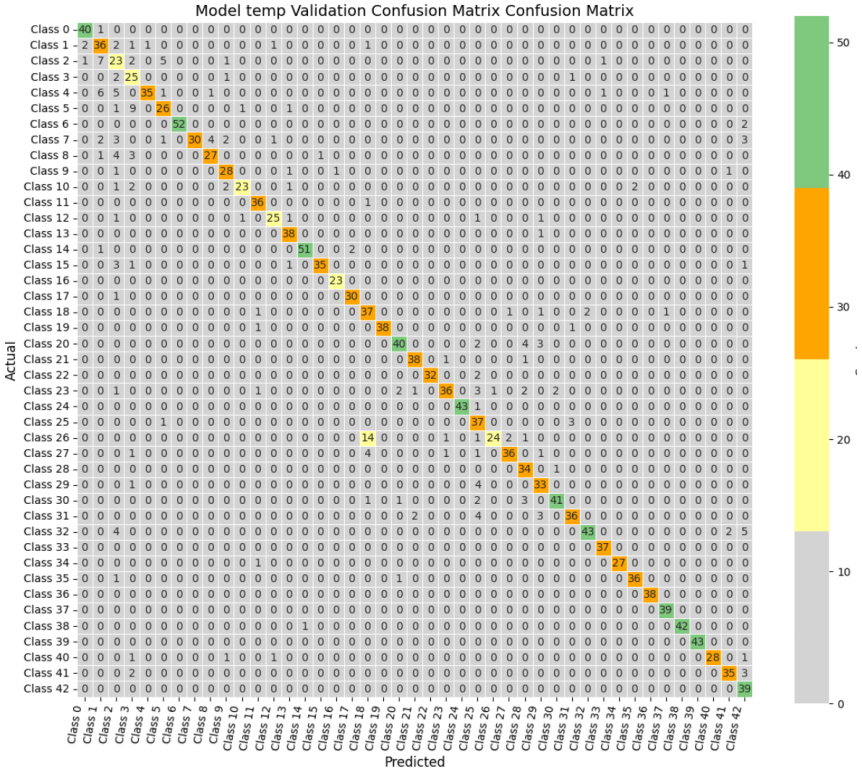
For illustration purposes, we are presenting in this paper the HOG and Template confusion matrices. For other features, please refer to our git repository.

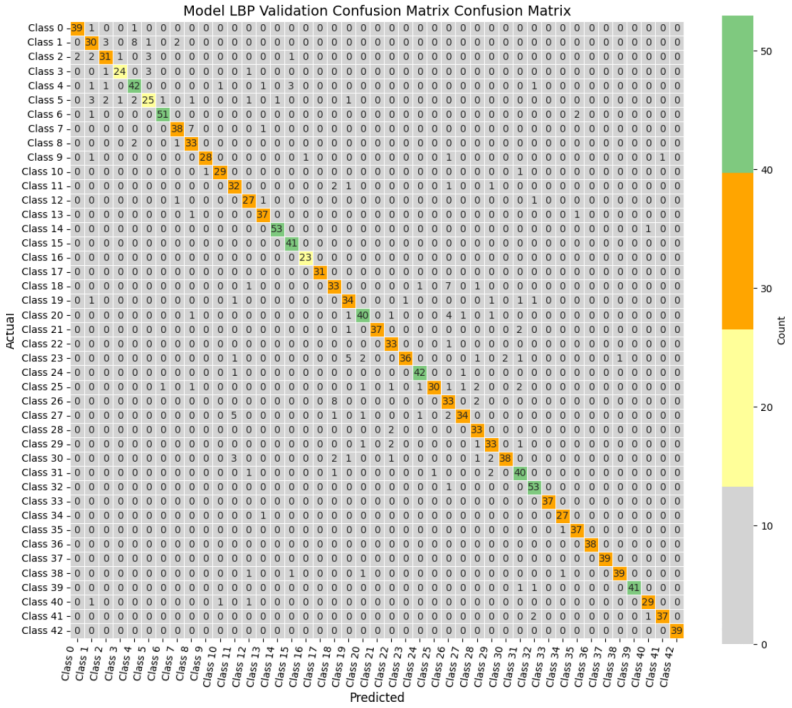




Confusion matrix for Conv1D Model


Model HOG Validation Confusion Matrix Confusion Matrix
Actual
Predicted
Count

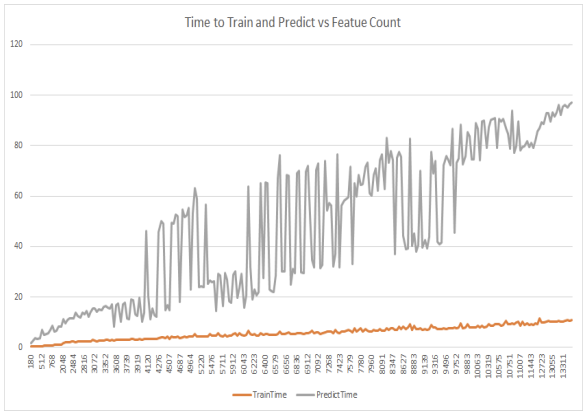

Model temp Validation Confusion Matrix Confusion Matrix
Actual
Predicted

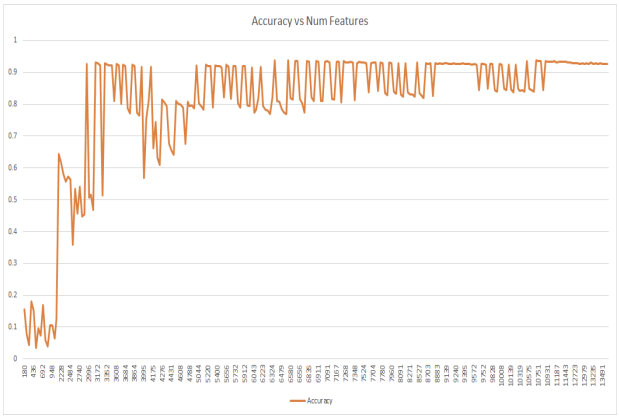


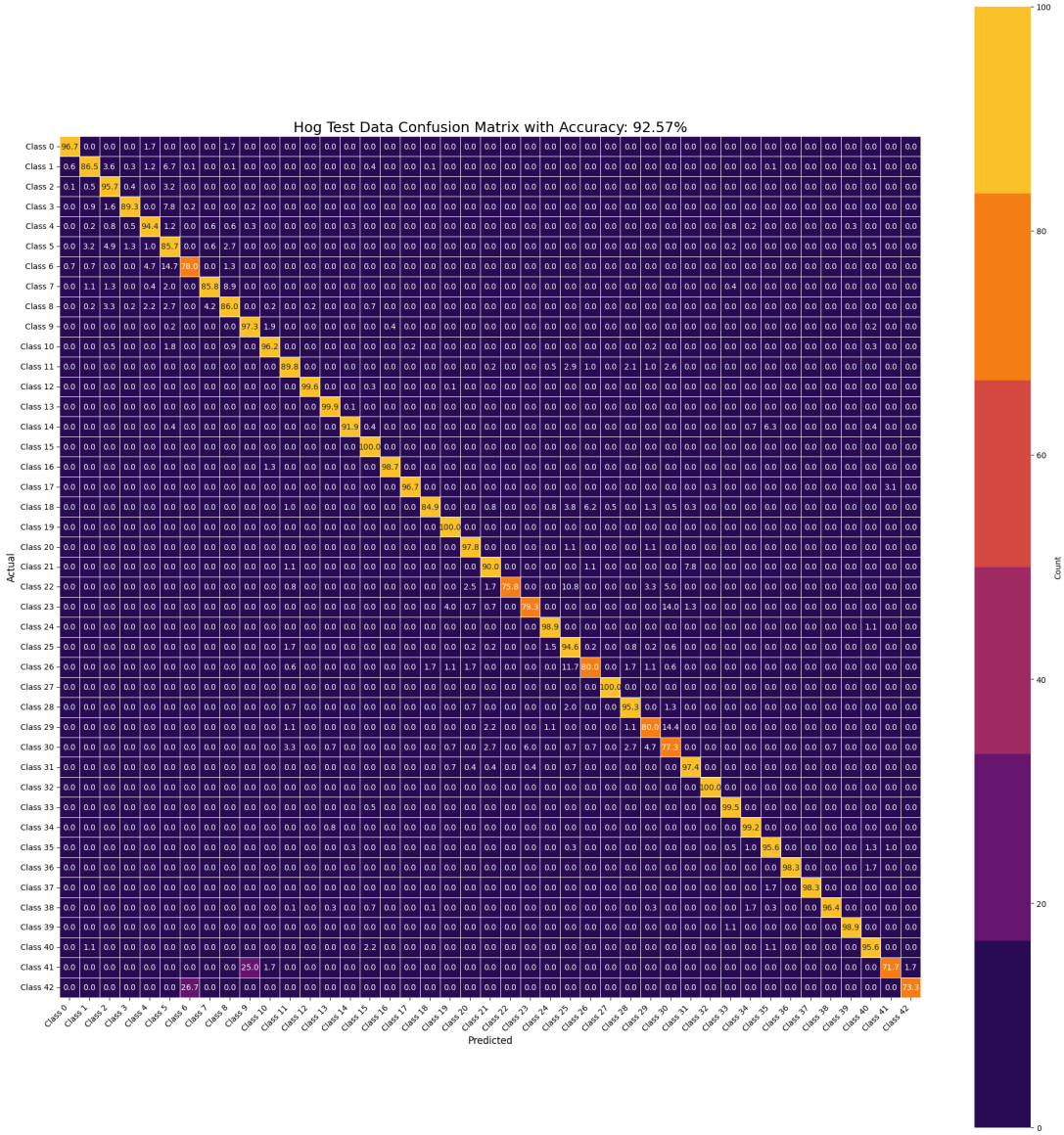
Appendix 2: SVM Model Performance

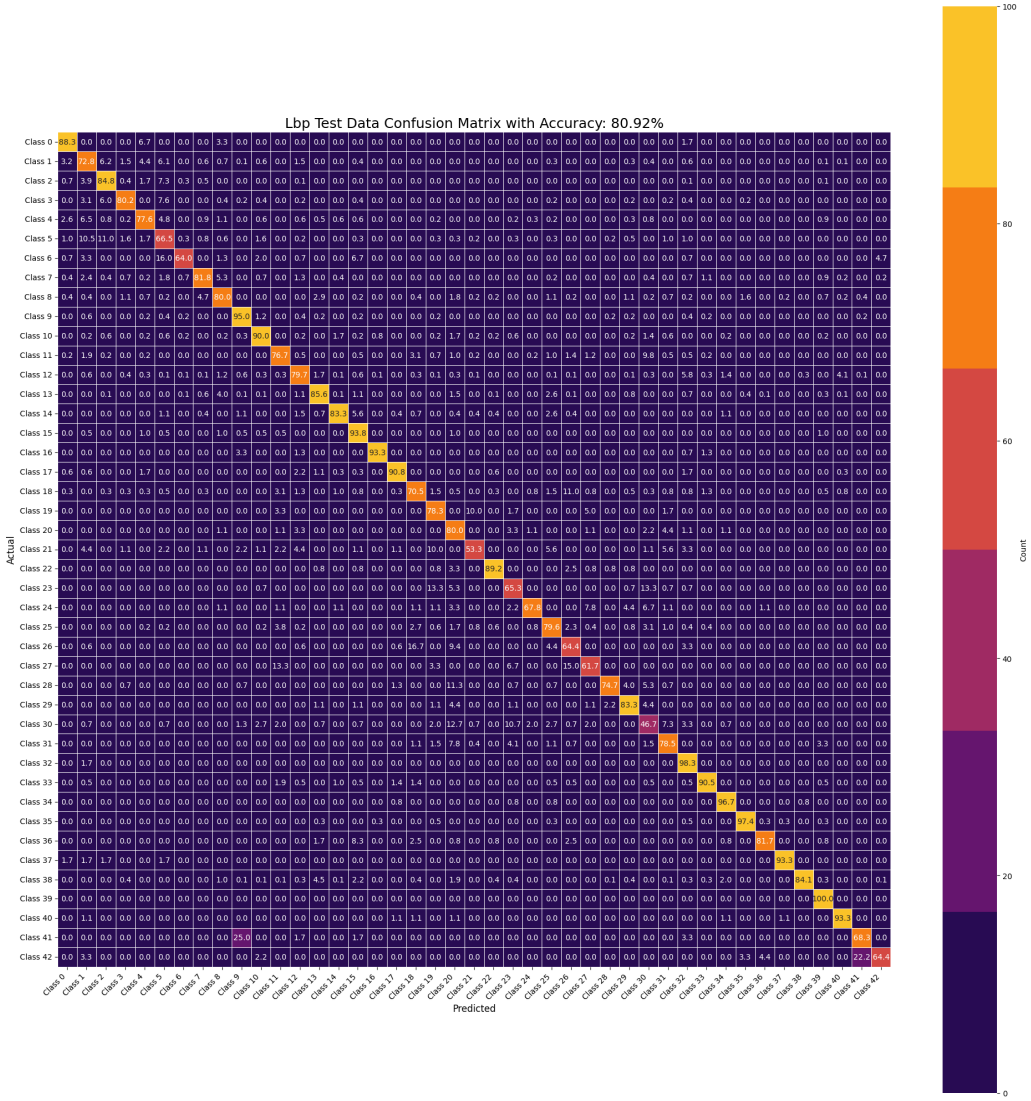
Performance of Most Accurate Feature Vectors					
Features	**NumFeatures**	**TrainTime**	**PredictTime**	**TotalTime**	**Accuracy**
["hue", "hog", "template"]	6579	5.107984	28.448629	33.556613	0.939113
["hue", "hog", … "template"]	10675	10.667384	86.932375	97.599759	0.938084
["hog", "template"]	6399	4.944094	27.446252	32.390346	0.938084
["hog", "lbp", "template"]	10495	9.180042	79.244453	88.424495	0.937134
["hue", "sat", … "template"]	6835	5.367003	31.019355	36.386358	0.93665

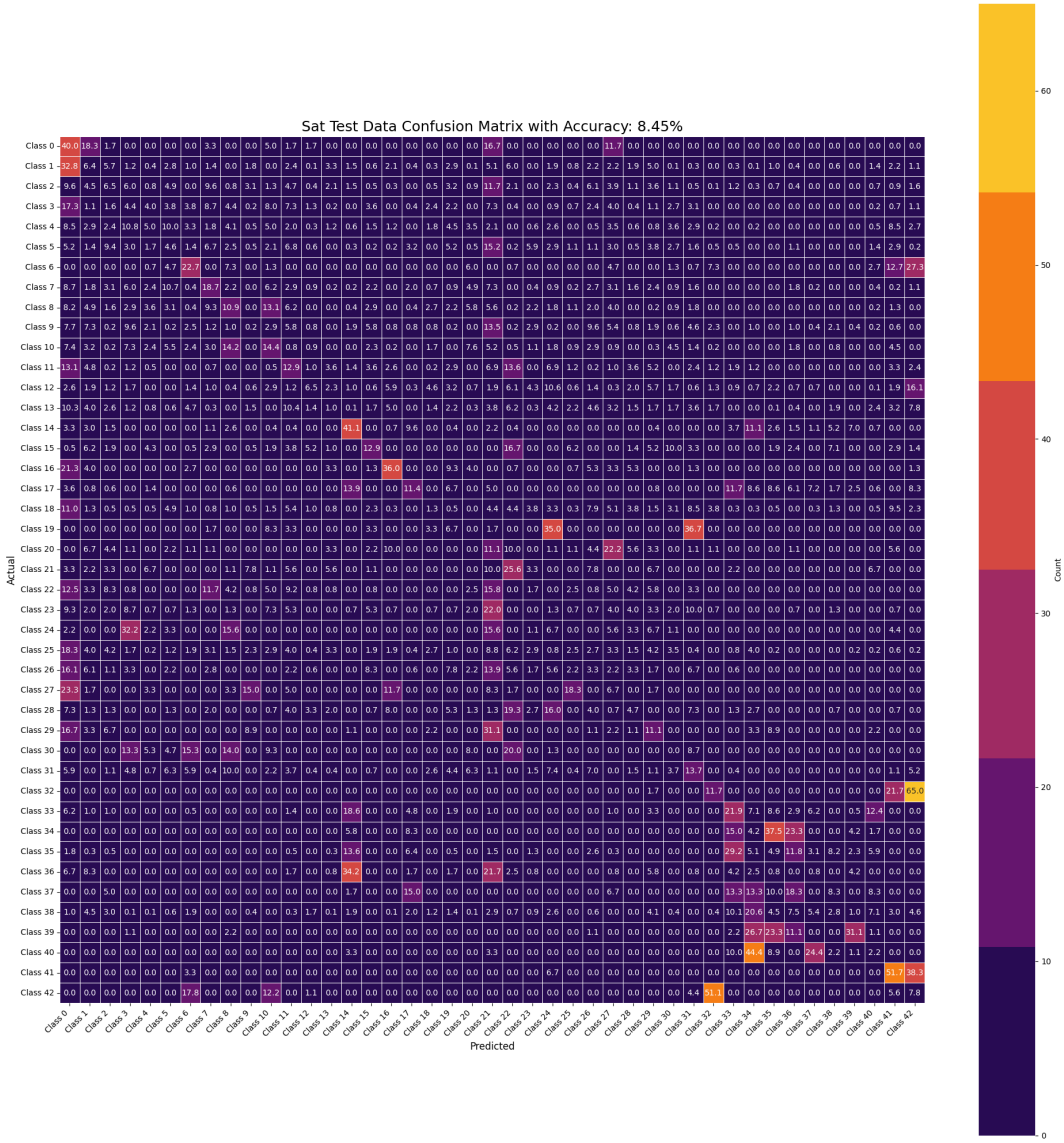
Performance of Most Efficient Feature Vectors					
Features	**NumFeatures**	**TrainTime**	**PredictTime**	**TotalTime**	**Accuracy**
["hog"]	2916	2.298981	12.276515	14.575496	0.927712
["hue", "hog"]	3096	2.533071	14.051537	16.584608	0.930641
["value", "hog"]	3172	2.812045	14.721348	17.533393	0.922803
["sat", "hog"]	3172	2.868986	15.072888	17.941874	0.928583
["hue", "value", "hog"]	3352	2.915004	15.744933	18.659937	0.92399

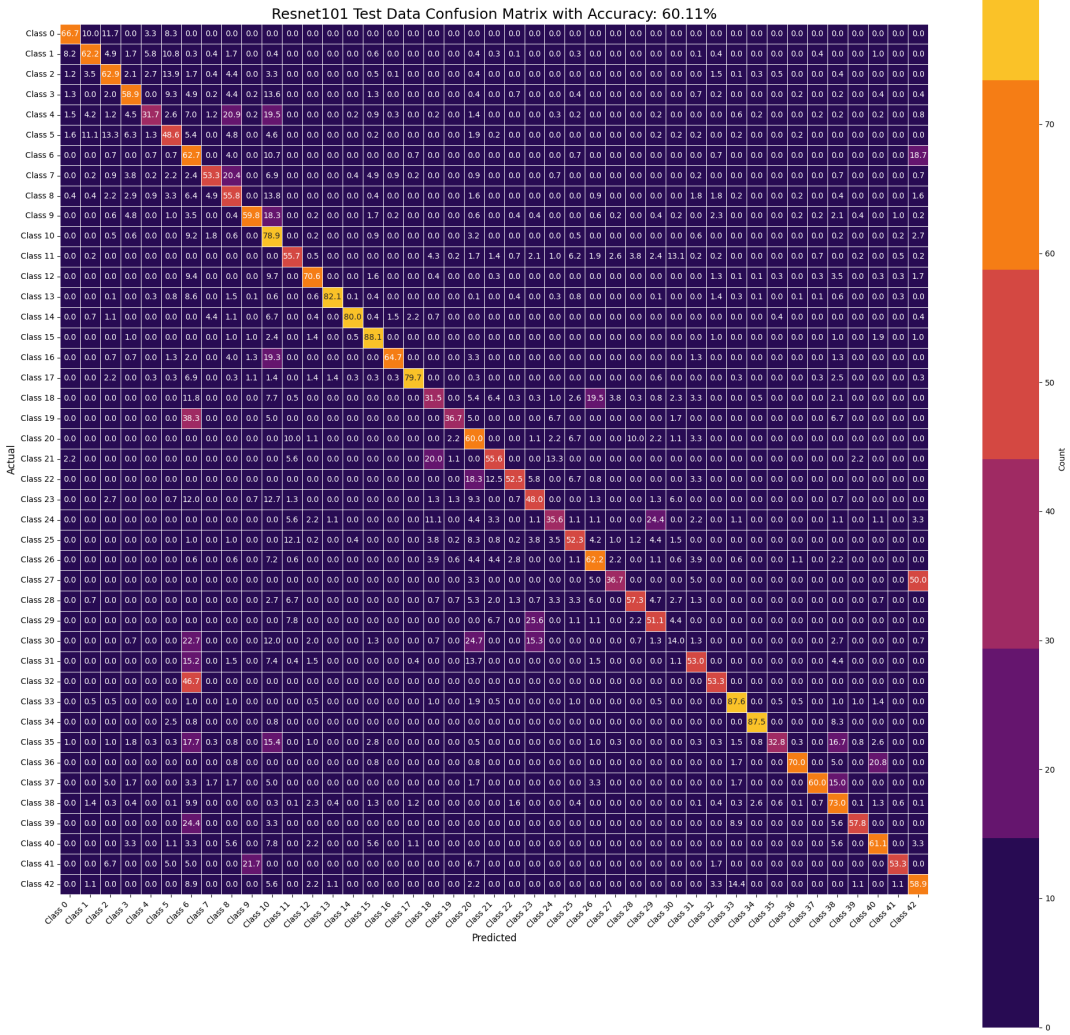

Time to Train and Predict vs Featue Count
120
100
80
60
40
20
0
TrainTime
PredictTime

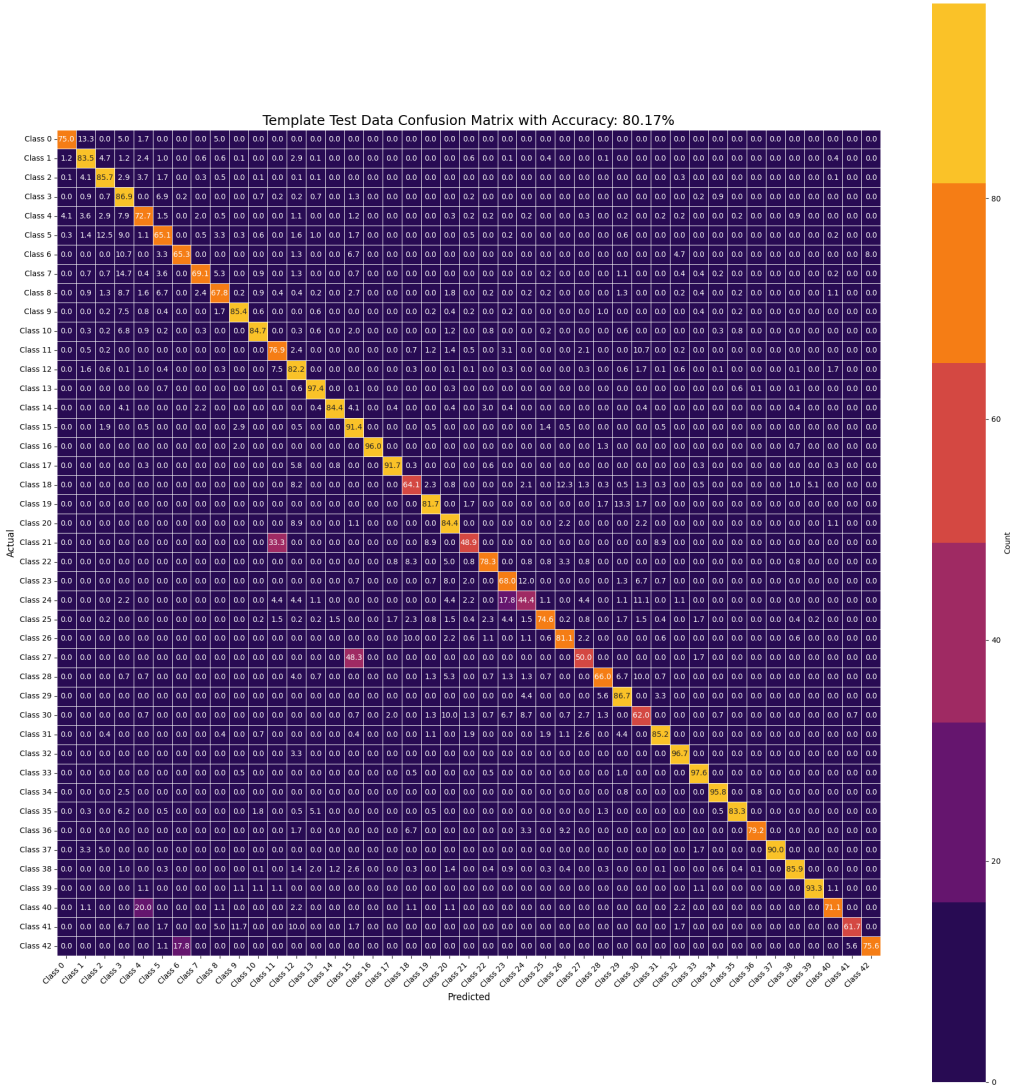

Accuracy vs Num Features
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Accuracy

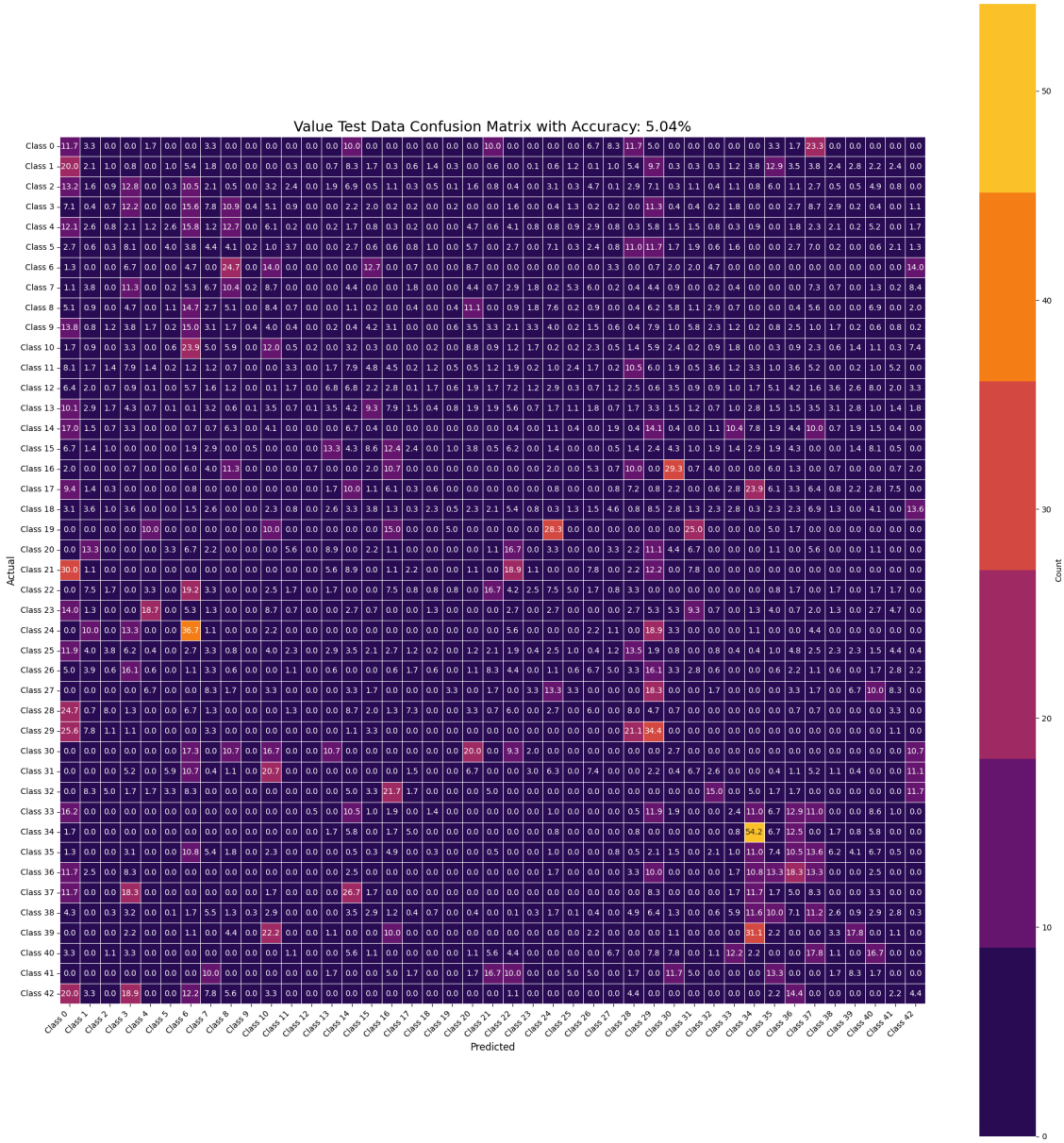

Hog Test Data Confusion Matrix with Accuracy: 92.57%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
20
40
60
80
100

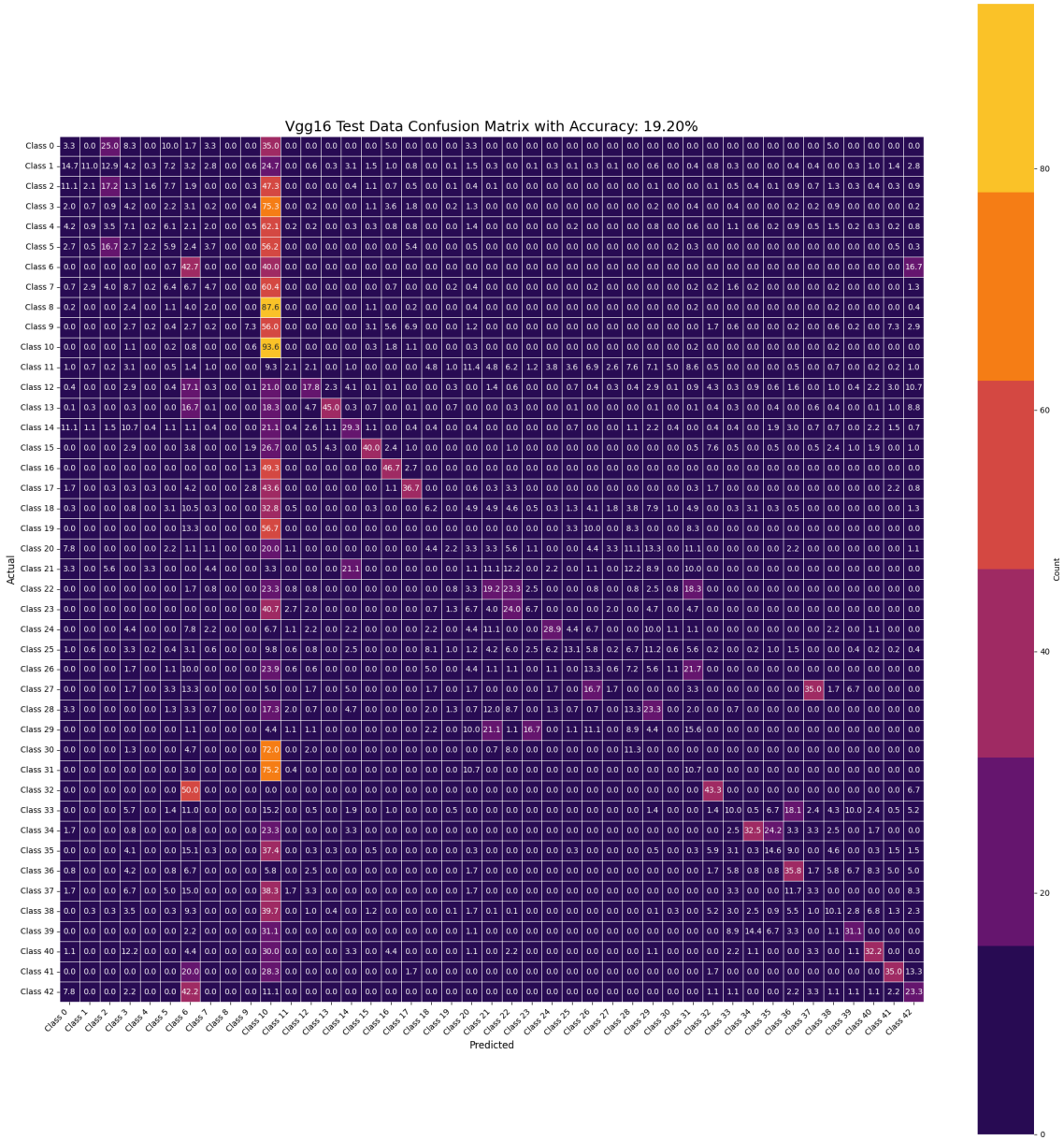

Lbp Test Data Confusion Matrix with Accuracy: 80.92%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
20
40
60
80
100


Sat Test Data Confusion Matrix with Accuracy: 8.45%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
10
20
30
40
50
60

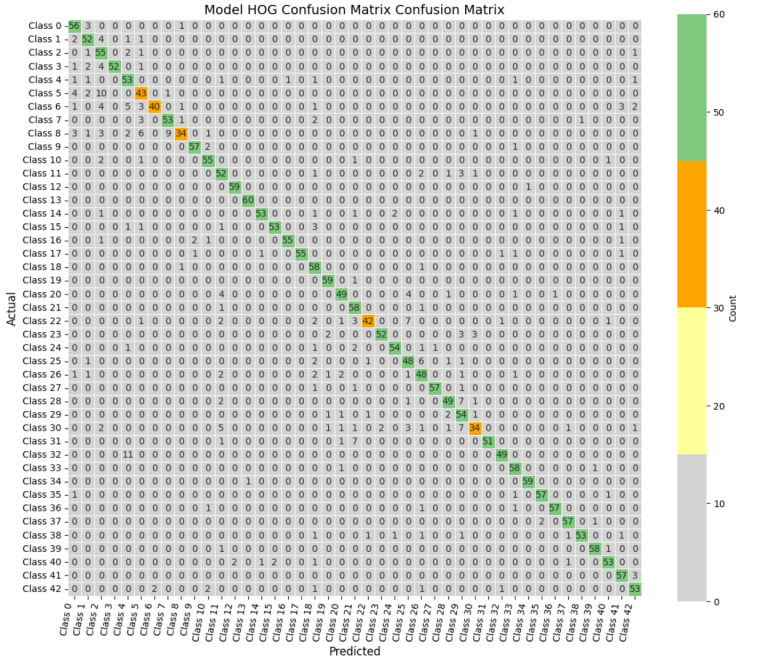

Resnet101 Test Data Confusion Matrix with Accuracy: 60.11%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
10
20
30
40
50
60
70
80

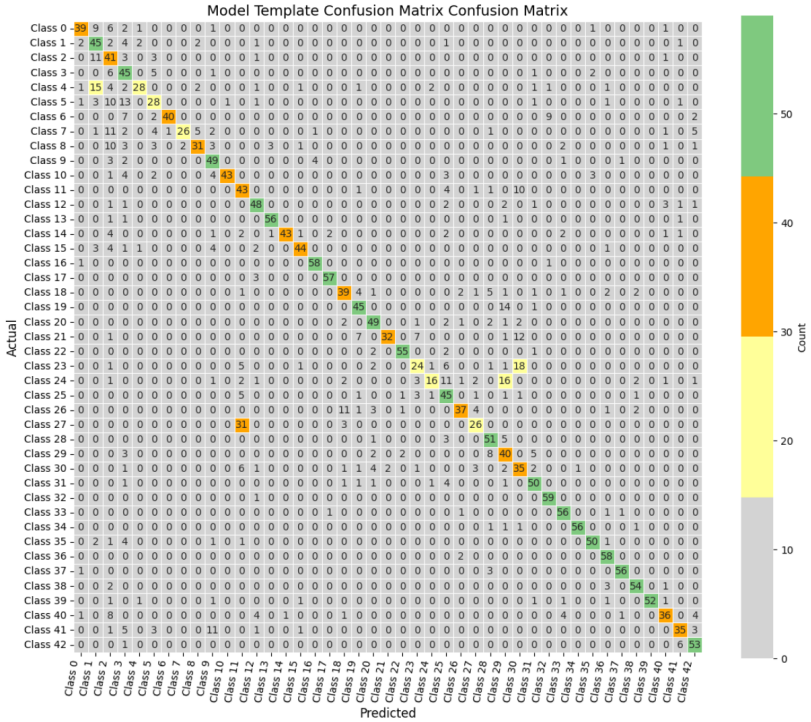

Template Test Data Confusion Matrix with Accuracy: 80.17%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
20
40
60
80

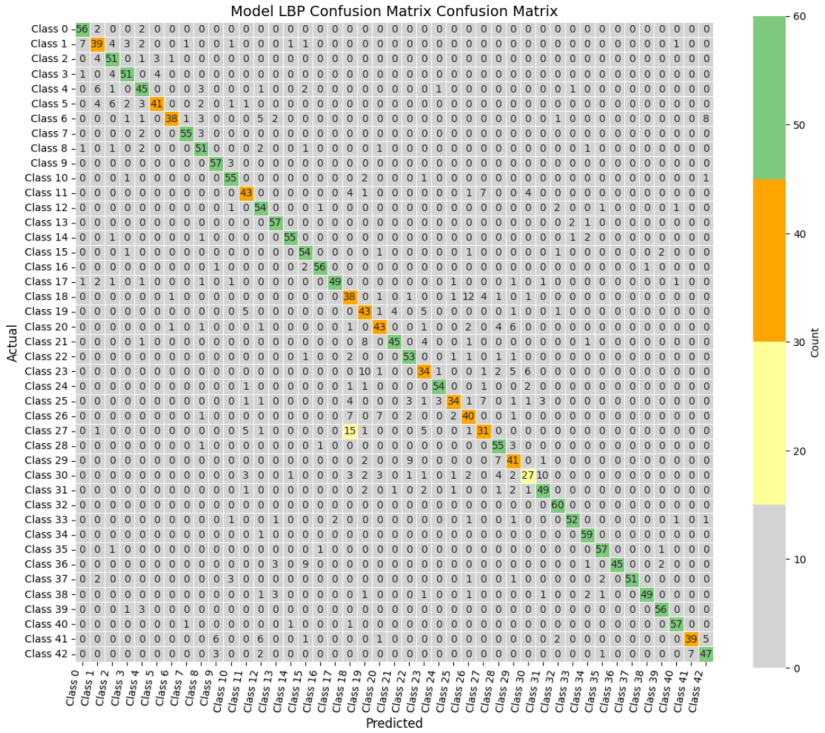

Value Test Data Confusion Matrix with Accuracy: 5.04%
Actual
Predicted
Count
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32
Class 33
Class 34
Class 35
Class 36
Class 37
Class 38
Class 39
Class 40
Class 41
Class 42
0
10
20
30
40
50



Confusion matrix for Conv1D Model - Using Test Dataset


Model HOG Confusion Matrix Confusion Matrix
Actual
Predicted
Count


Model Template Confusion Matrix Confusion Matrix
Actual
Predicted
Count



Appendix 3 Conv1D Model Performance

Efficiency vs. Accuracy Results

Feature	Epochs	Filter Size	Kernel Size	Dense Layer	Dropout	Train Accuracy	Val Accuracy	Test Accuracy	Train Time	Train Time
HOG+Template	10	[6, 16]	[21, 5]	[120, 84]	0.5	0.1187	0.1198	0.1326	87	87s
HOG+Template	20	[12, 16]	[21, 5]	[120, 84]	0.5	0.0673	0.0616	0.0647	112	112s
HOG+Template	20	[12, 16]	[21, 5]	[120, 84]	0.3	0.9744	0.9384	0.8601	182	182s
HOG+Template	10	[6, 16]	[21, 5]	[84, 64]	0.3	0.7772	0.7674	0.7198	201	201s
HOG+Template	10	[6, 16]	[21, 5]	[84, 64]	0.5	0.2517	0.2116	0.2198	206	206s
HOG+Template	10	[6, 16]	[21, 5]	[120, 84]	0.3	0.9884	0.9337	0.8578	221	221s
HOG+Template	20	[12, 16]	[21, 5]	[84, 64]	0.5	0.1664	0.1703	0.1767	235	235s
HOG+Template	20	[6, 16]	[21, 5]	[84, 64]	0.3	0.9836	0.9366	0.8481	243	243s
HOG+Template	10	[12, 16]	[21, 5]	[84, 64]	0.5	0.2538	0.2547	0.2539	252	252s
HOG+Template	10	[12, 16]	[21, 5]	[84, 64]	0.3	0.9810	0.9262	0.8178	253	253s
HOG+Template	10	[12, 16]	[21, 5]	[120, 84]	0.5	0.2554	0.2424	0.2473	260	260s
HOG+Template	10	[12, 16]	[21, 5]	[120, 84]	0.3	0.9837	0.9262	0.8422	263	263s
HOG+Template	20	[12, 16]	[21, 5]	[84, 64]	0.3	0.9887	0.9494	0.8764	362	362s
HOG+Template	20	[6, 16]	[21, 5]	[84, 64]	0.5	0.3638	0.3349	0.3481	394	394s
HOG+Template	20	[6, 16]	[21, 5]	[120, 84]	0.3	0.9895	0.9297	0.8523	425	425s
HOG+Template	20	[6, 16]	[21, 5]	[120, 84]	0.5	0.9847	0.9355	0.8612	431	431s

Feature	Epochs	Filter Size	Kernel Size	Dense Layer	Dropout	Train Accuracy	Val Accuracy	Test Accuracy	Train Time	Train Time
HOG	20	[12, 16]	[21, 5]	[120, 84]	0.3	0.9820	0.9465	0.8841	74	74s
HOG	10	[6, 16]	[21, 5]	[84, 64]	0.3	0.9923	0.9500	0.8849	80	80s
HOG	20	[6, 16]	[21, 5]	[84, 64]	0.3	0.9885	0.9419	0.8795	81	81s
HOG	20	[6, 16]	[21, 5]	[120, 84]	0.3	0.9853	0.9552	0.8771	85	85s
HOG	10	[6, 16]	[21, 5]	[84, 64]	0.5	0.9821	0.9465	0.8779	91	91s
HOG	20	[6, 16]	[21, 5]	[120, 84]	0.5	0.9892	0.9430	0.8787	106	106s
HOG	10	[6, 16]	[21, 5]	[120, 84]	0.3	0.9858	0.9273	0.864	107	107s
HOG	10	[6, 16]	[21, 5]	[120, 84]	0.5	0.9900	0.9407	0.8756	110	110s
HOG	10	[12, 16]	[21, 5]	[84, 64]	0.3	0.9849	0.9448	0.8655	111	111s
HOG	20	[12, 16]	[21, 5]	[84, 64]	0.3	0.9881	0.9483	0.8659	111	111s
HOG	10	[12, 16]	[21, 5]	[120, 84]	0.3	0.9933	0.9285	0.864	115	115s
HOG	10	[12, 16]	[21, 5]	[84, 64]	0.5	0.9868	0.9355	0.8674	125	125s
HOG	20	[12, 16]	[21, 5]	[84, 64]	0.5	0.9892	0.9448	0.8826	137	137s
HOG	20	[6, 16]	[21, 5]	[84, 64]	0.5	0.9910	0.9419	0.876	140	140s
HOG	10	[12, 16]	[21, 5]	[120, 84]	0.5	0.9908	0.9372	0.8713	145	145s
HOG	20	[12, 16]	[21, 5]	[120, 84]	0.5	0.9847	0.9384	0.8752	147	147s