**Traffic Sign Classification for Driver Assistance Systems**

Final Report

**Nicholas Butta, Cassie Xia, Johanna Rivera**

Machine Learning for Signal Processing

Spring 2020



**Table of Contents**

[**INTRODUCTION**](#_heading=h.30j0zll) **3**

[**PROJECT OBJECTIVE**](#_heading=h.1fob9te) **5**

[**PROJECT TASKS**](#_heading=h.3znysh7) **6**

[Sensor](#_heading=h.2et92p0) 6

[Signal Capture](#_heading=h.tyjcwt) 8

[Data Conditioning](#_heading=h.3dy6vkm) 8

[Feature Extraction](#_heading=h.1t3h5sf) 12

[Modeling/Prediction](#_heading=h.4d34og8) 16

[**RESULTS**](#_heading=h.2s8eyo1) **19**

[**CONCLUSIONS**](#_heading=h.17dp8vu) **22**

[**SUMMARY**](#_heading=h.ifw9nyikg76q) **23**

[**REFERENCES**](#_heading=h.3rdcrjn) **24**

**Traffic Sign Classification for Driver Assistance Systems**

# INTRODUCTION

Traffic sign classification is the process of automatically recognizing traffic signs along the road, including speed limit signs, warning signs, turn signs, etc. As humans, we are able to easily recognize a large number of road signs at various distances and speeds and under different lighting conditions with very little effort at close to 100% accuracy. For computers, however, the task of recognizing and classifying traffic signs is a challenging, real-world problem with high industrial relevance.

Each year, more and more money and research are invested into the field of autonomous or self-driving vehicles. This is because this technology has the potential to reduce fuel use and carbon emissions, improve driver safety, reduce road congestion, and increase operator productivity and convenience. In order to achieve these goals, self-driving cars need to implement some form of traffic sign recognition and classification in order to properly react to the ever-changing rules of modern roadways. This is just one task a self-driving vehicle must perform as a part of a larger sensor-driven control system. Similarly, driver assistance/alert systems can leverage a similar capability in order to make vehicle operators more aware of their surroundings and alert them of changes in the roadway (see figures below). Ultimately, being able to automatically recognize traffic signs efficiently and accurately is one of many requirements necessary to build truly autonomous/self-driving vehicles.

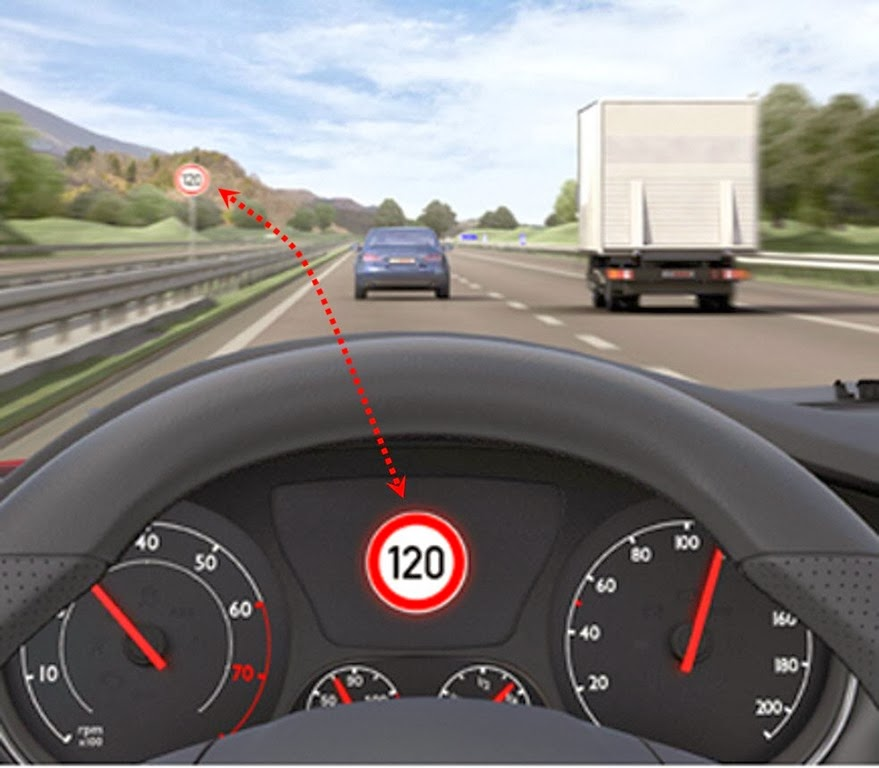
As an example, the 2020 Honda Accord includes a front camera sensor that is mounted to the interior of the windshield behind the rearview mirror. The camera captures light radiation (video) from the environment ahead, and a combination of computer hardware and software interprets the captured video in real-time, detecting and classifying traffic signs along the road. Based on these signs, the software could theoretically make decisions to control the vehicle (speed, direction, etc.), or at the very least, alert the driver on a heads-up display (HUD). This is an example of a highly complex computer vision and image recognition task.

The reason we are interested in this problem is because traffic sign classification, and image recognition in general, is a problem that machine learning algorithms (supervised and unsupervised) that we’ve learned in this course can solve. Our motivation for executing this project successfully lies in the fact that the problem of accurately classifying traffic signs automatically is an example of a real-time, safety-critical system where the health and well-being of humans is at stake. It is very important to research and develop efficient and highly-accurate models in order for this technology to be widely adapted by the consumer automobile industry. If our project is successful, it means progress will be made towards the goals of improving driver situational awareness and implementing fully autonomous and trustworthy vehicles that abide by the ever-changing rules of the road.

Given the motivations stated above, our team was able to make significant progress towards studying and implementing various machine learning models to solve the problem of traffic sign recognition. We successfully implemented each stage of the Machine Learning for Signal Processing (MLSP) paradigm using MATLAB. The repository containing our MATLAB scripts can be found at the following link:

<https://github.com/nbutta/MachineLearning670Spring2020>.

Our team spent a lot of time understanding our dataset and identifying opportunities for performing data conditioning. This meant researching many image processing techniques outside the scope of this course. We then implemented feature extraction techniques learned in the course for the purposes of reducing our feature space dimensionality and increasing model efficiency. This led to the successful implementation and analysis of four different machine learning classification models with fantastic results. This was all accomplished virtually across two different countries and three different time zones, with a time zone difference of up to eight hours, while working full-time jobs and battling through the COVID19 global health pandemic. This required abundant communication between team members and careful scheduling, with a particular focus on verbose code commenting and documentation.



**Figure 1.** Autonomous Vehicle Control System **Figure 2.** Driver Assistance HUD

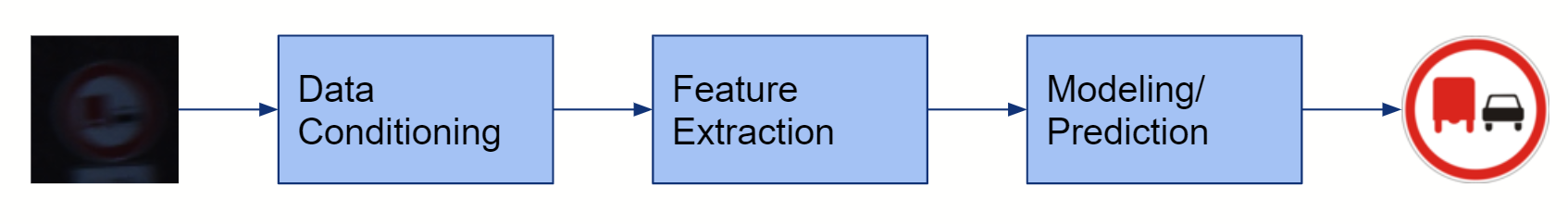
# PROJECT OBJECTIVE

The project at hand is a single-image, multi-task classification machine learning problem with more than 40 classes (sign types) and more than 50,000 images in total. The dataset used is known as the German Traffic Sign Recognition Benchmark (GTSRB). The original objective of our team’s project was to use three different classifiers trained on a set of training images to classify single test images of traffic signs. This would require a careful examination of the dataset and the implementation of each step in the MLSP chain. Given our team’s previous experience, we decided to perform these tasks using MATLAB. Once implemented and tested, the next logical step would be to examine the performance and compare the results of each classifier.

The classifiers we thought to implement were Naive Bayes, K-Nearest Neighbor (KNN), and Support Vector Machines (SVM). We chose these classifiers because they included both supervised and unsupervised learning algorithms, and we desired to compare and understand their performance given the task at hand. Finally, we were able to achieve our objective by implementing four classifiers. In addition to those classifiers listed above, we also decided to implement and study the performance of a Convolutional Neural Network (CNN). We knew that image recognition tasks lend themselves well to CNN models after some initial research on this topic, so once we learned about neural networks at the end of the course, we decided to implement this additional classifier. Sure enough, the CNN produced the best results in terms of classification accuracy, which was a nice reward for our team going above and beyond our original objectives.

# PROJECT TASKS

In figure 3 below, you can see how an example sign image progresses through the MLSP chain defined in this course. The images from our dataset are captured from real vehicle-mounted cameras and are therefore representative of actual road conditions and signal variation. The dataset was available for download from the Internet. In the data conditioning stage, we decided to reduce the large dataset size by removing low quality images. We also investigated the use of image contrast enhancement to improve classification results. Next, we used a variety of features to train our models like using the pixel intensity values, Principal Component Analysis (PCA), and we even tried designing our own set of custom features like sign shape and color histograms via image preprocessing. These different sets of features were then used to model and predict traffic signs using Naive Bayes, KNN, SVM and CNN. This section will go into greater detail on what our team did for each of the MLSP tasks.



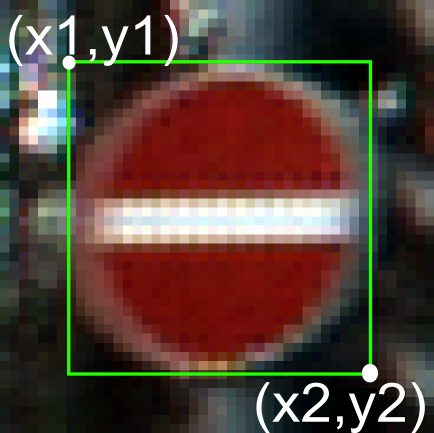
**Figure 3.** Machine Learning for Signal Processing (MLSP) paradigm

## Sensor

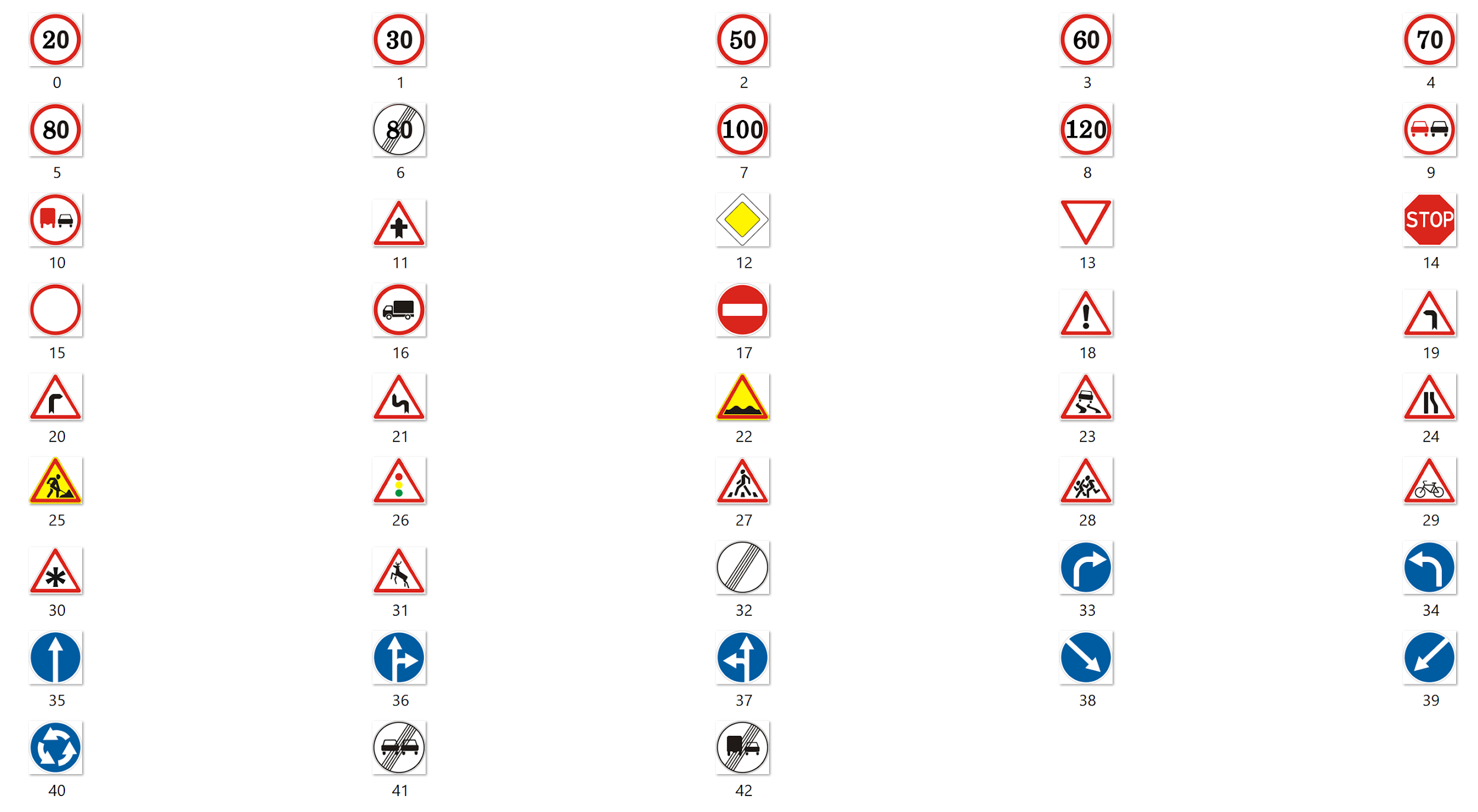
The sensor data used as input to the MLSP chain for this project are 3-channel RGB images of traffic signs. As previously mentioned, this dataset is known as the German Traffic Sign Recognition Benchmark (GTSRB) dataset. A sample of the GTSRB dataset is pictured in figure 4.

There are a total of 43 object classes in the dataset, shown in figure 5. The “meta” signs and their respective labels can be found in the image below. Additionally, some example dataset training and test instances can be seen in the figure below. Physical sign instances are unique within the dataset, meaning each real-world traffic sign only occurs once. Each image contains a single traffic sign with a border of 10% around the sign (at least 5 pixels). This border is to allow for edge-based image processing approaches. The dataset also contains the (x, y) coordinates for a bounding box around each sign as seen below on the right. The images are stored in PPM format (Portable Pixmap, P6) and have all different sizes. The images can be any size between 15x15 to 250x250 pixels and are not necessarily square. The dataset contains the dimensions of each image.

The dataset is already divided into labeled training and test images. There are a total of 39,209 training images and 12,630 test images. Each image is represented by an NxMx3 (RGB) matrix of intensity values. Some images are more clear than others. There is significant variation in the quality of the images, as to be expected, when captured from a moving vehicle in outdoor conditions.



**Figure 4**. GTSRB Dataset Sample



**Figure 5**. GTSRB Dataset with 43 Sign Object Classes

## Signal Capture

The GTSRB dataset used for this project was acquired via Internet download from the following location:

[GTSRB - German Traffic Sign Recognition Benchmark](https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign)

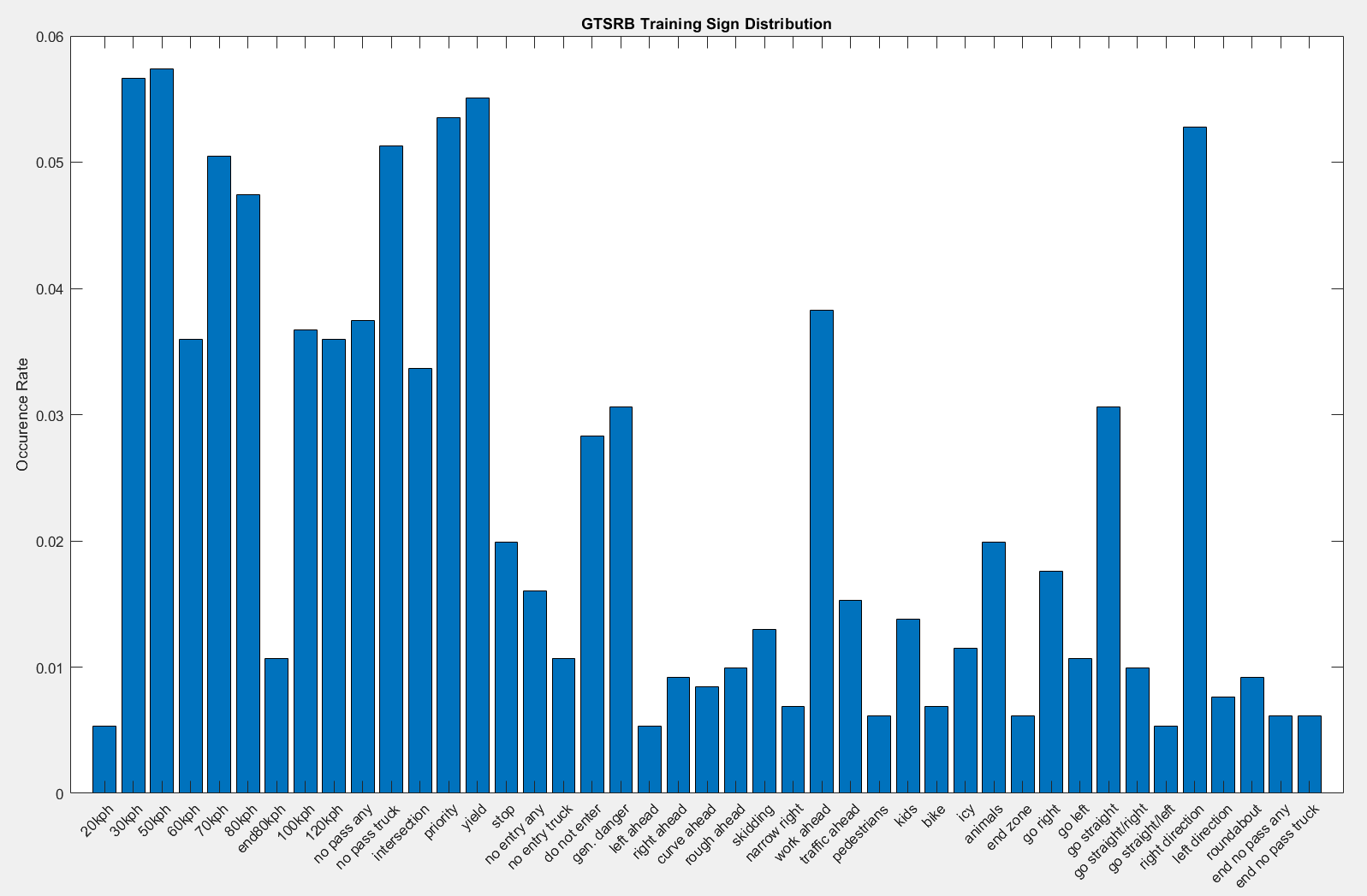
A detailed description of the dataset can be found here:

[German Traffic Sign Benchmarks](http://benchmark.ini.rub.de/?section=gtsrb&subsection=news)

## Data Conditioning

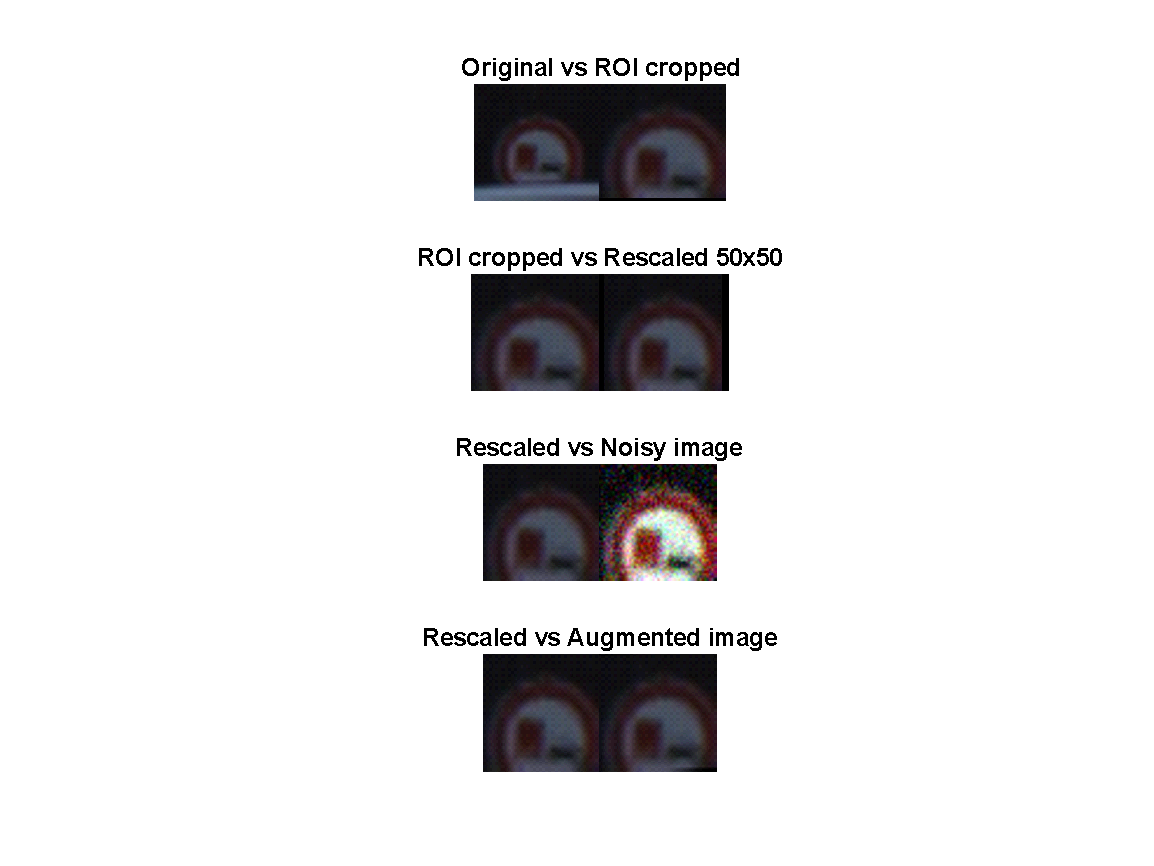
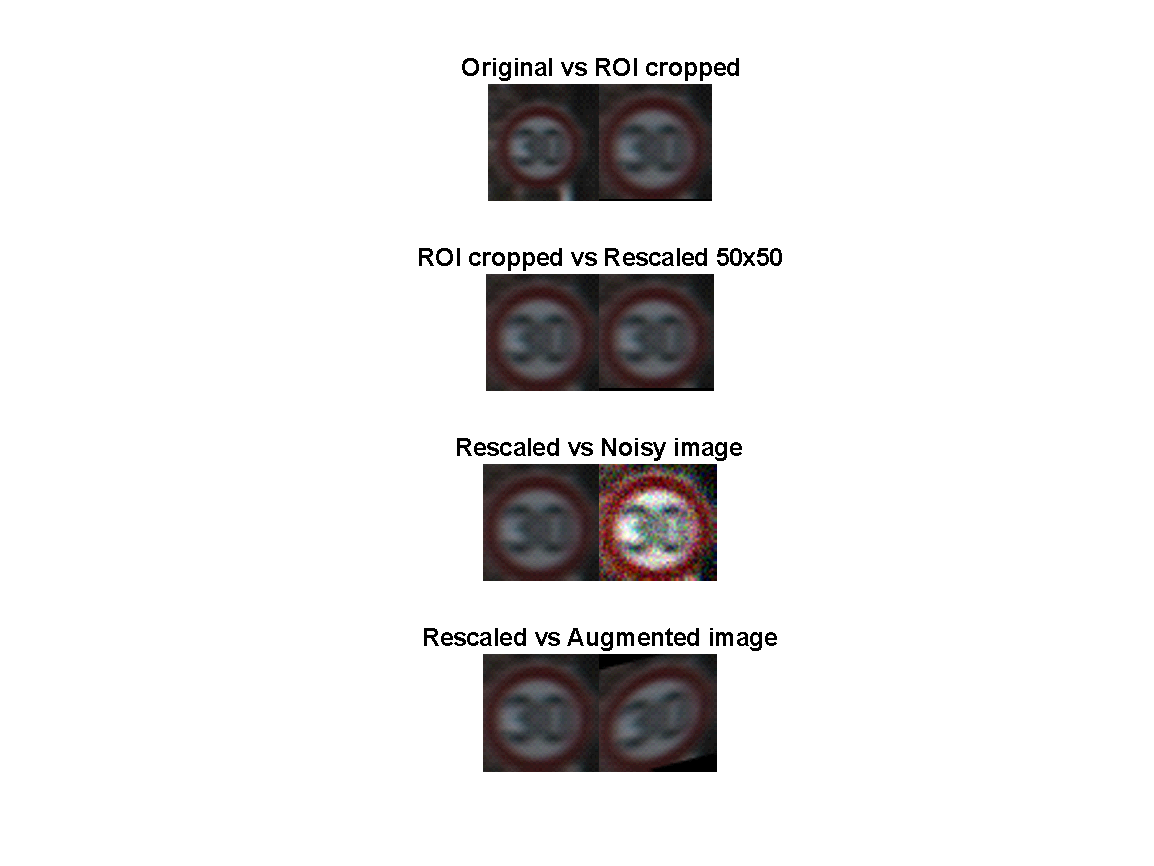
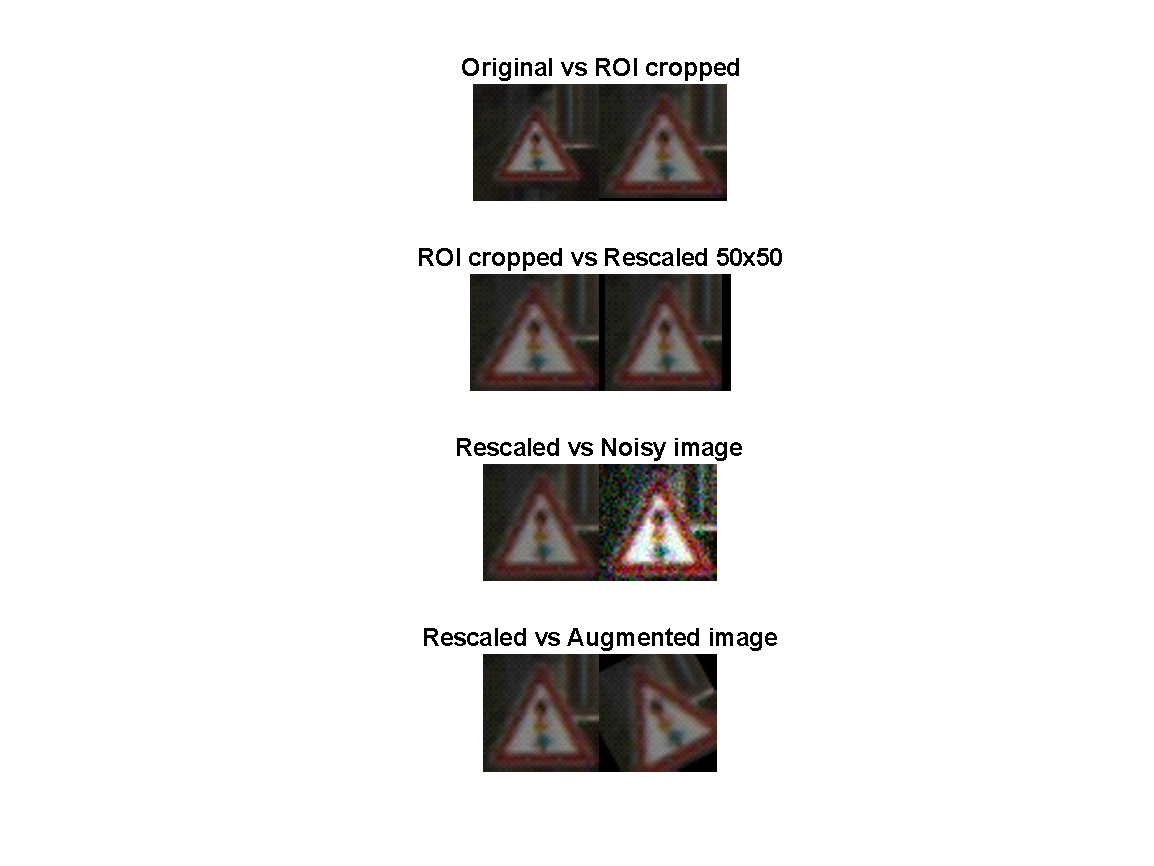
After acquisition of the GTSRB dataset from the Internet, our team conducted an in-depth analysis of the image data using MATLAB. This included studying the distribution of the training data by class, analyzing the quality of the images, and researching image processing techniques for improving the dataset for the purposes of model training and prediction accuracy. After an initial analysis of our dataset, it became apparent that our classifiers would have to be capable of coping with a variety of signal variation due to illumination changes, partial occlusions, rotations, weather conditions, etc if we neglected to condition the data.

First, we noted that among the training data, there was considerable class skew. This is evidenced by the bar plot (figure 6) below of the relative frequencies of each sign in the 39,209-training image dataset.



**Figure 6**. GTSRB Training Sign Distribution

We first thought to even the distribution of training data, as class imbalance can often cause problems in machine learning classification models. To this end, the team implemented image processing techniques in MATLAB to perform image data augmentation in an attempt to even the distribution of sign images in the dataset and avoid biased classifiers. The goal of data augmentation in machine learning is to generate new data from existing data by adding noise or performing a variety of augmentations like rotations and translations. Some examples of the results of this process can be seen below. Derivations of the existing images with low frequencies in the training dataset could theoretically be added, while some images with high frequency could be removed to even the distribution of the data. Ultimately, our team decided against this form of data conditioning as the class skew may actually lead to better performance, given that the skew is representative of the actual frequencies of signs on the roadway.



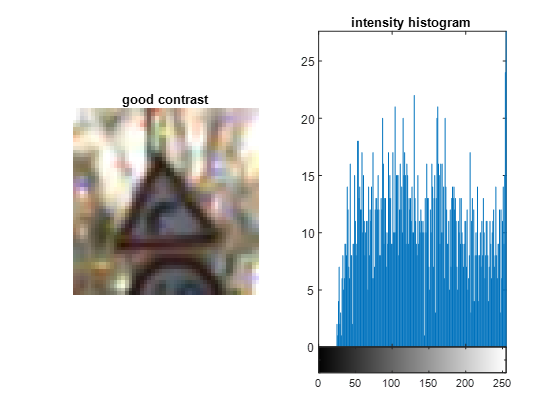
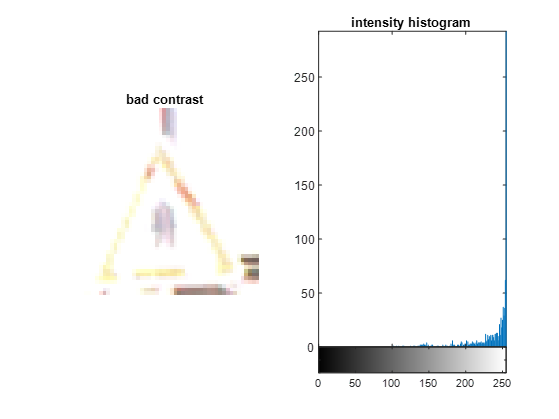
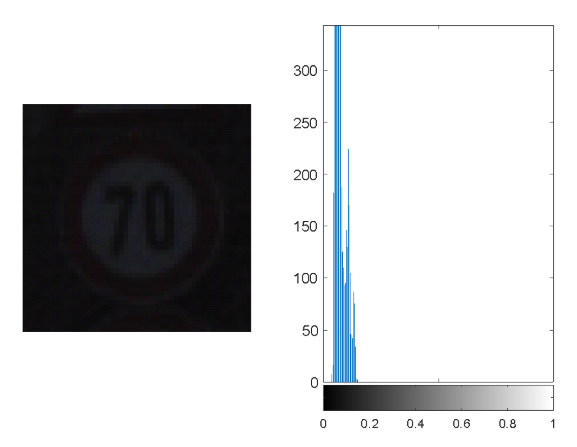
**Figure 7**. Image processing comparisons.

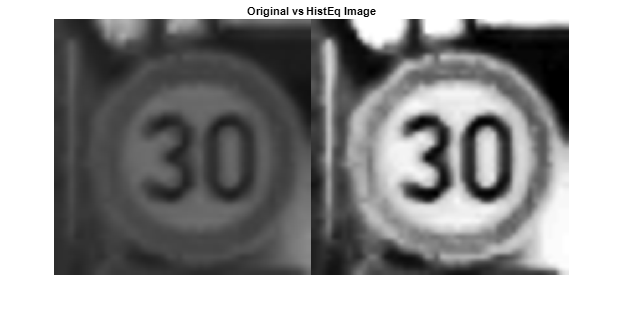
We then studied the distribution of image sizes within the dataset. It was found that there was a large variation in image size within the dataset, shown in figure 8, and that the average image size was around 50x50 pixels. Some sample images and their disparity in size can be viewed below. It is very difficult for the human eye to make out the small image. Thus, we conditioned the data by resizing all images to 50x50 after capturing the image Region of Interest (ROI) for training and prediction purposes. The ROI was used to remove the sign background, as the intensity values from the background are not relevant to the sign intensities, and may mislead the model.



**Figure 8**. Comparison between different image sizes within dataset

In addition to variations in image size, we identified image contrast as a major source of signal variation. Two sample images from the training set and an histogram of their pixel intensities can be found below. Even though the image size is reasonable, the poor distribution of image contrast makes the image on the left very difficult to recognize, while the wider distribution of the image on the right makes the sign much easier to see. Because of this, our team decided to enhance the contrast of all training and test images using grayscale histogram equalization. This process removes the color information from the data, but spreads the image intensity values across the entire dynamic range of 0-255, making images much easier to identify (example below in figure 9).

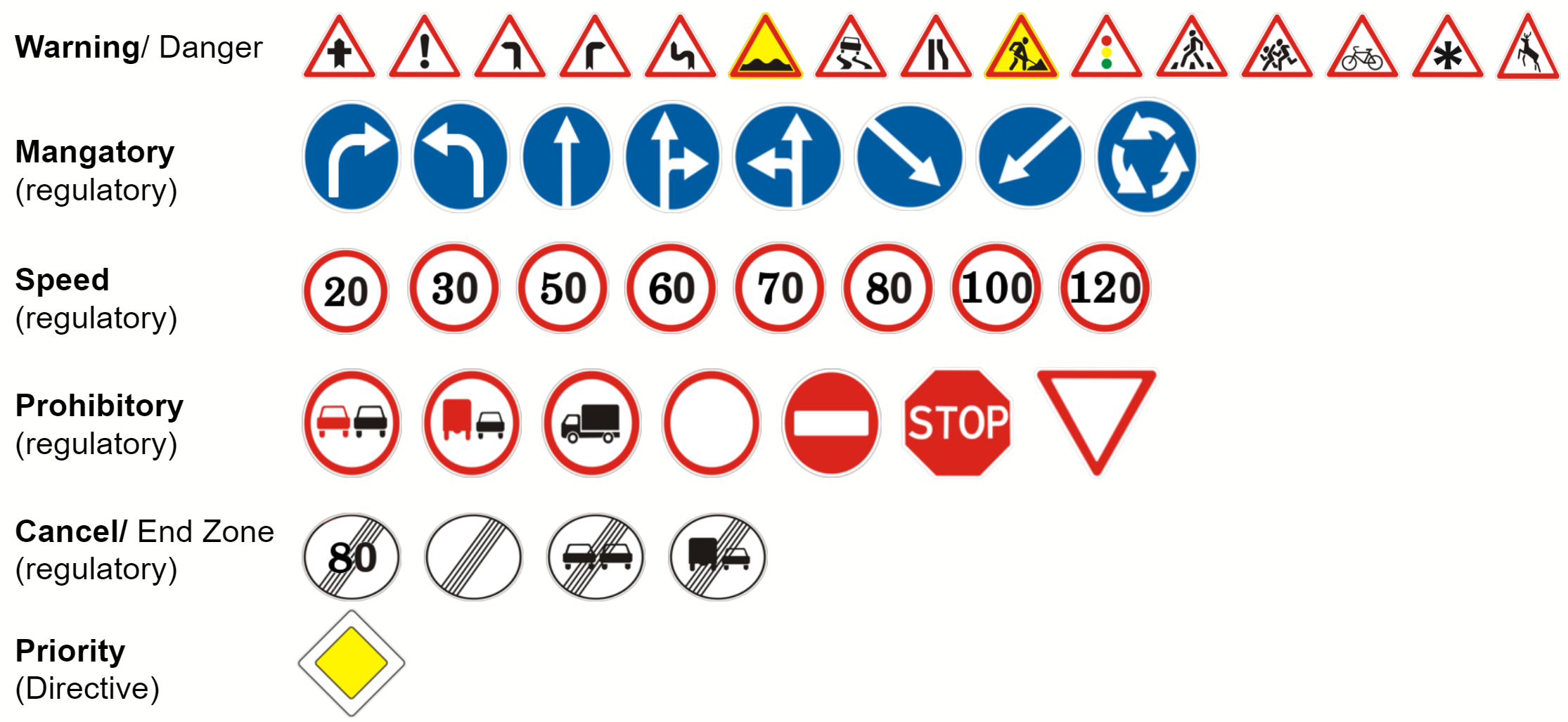




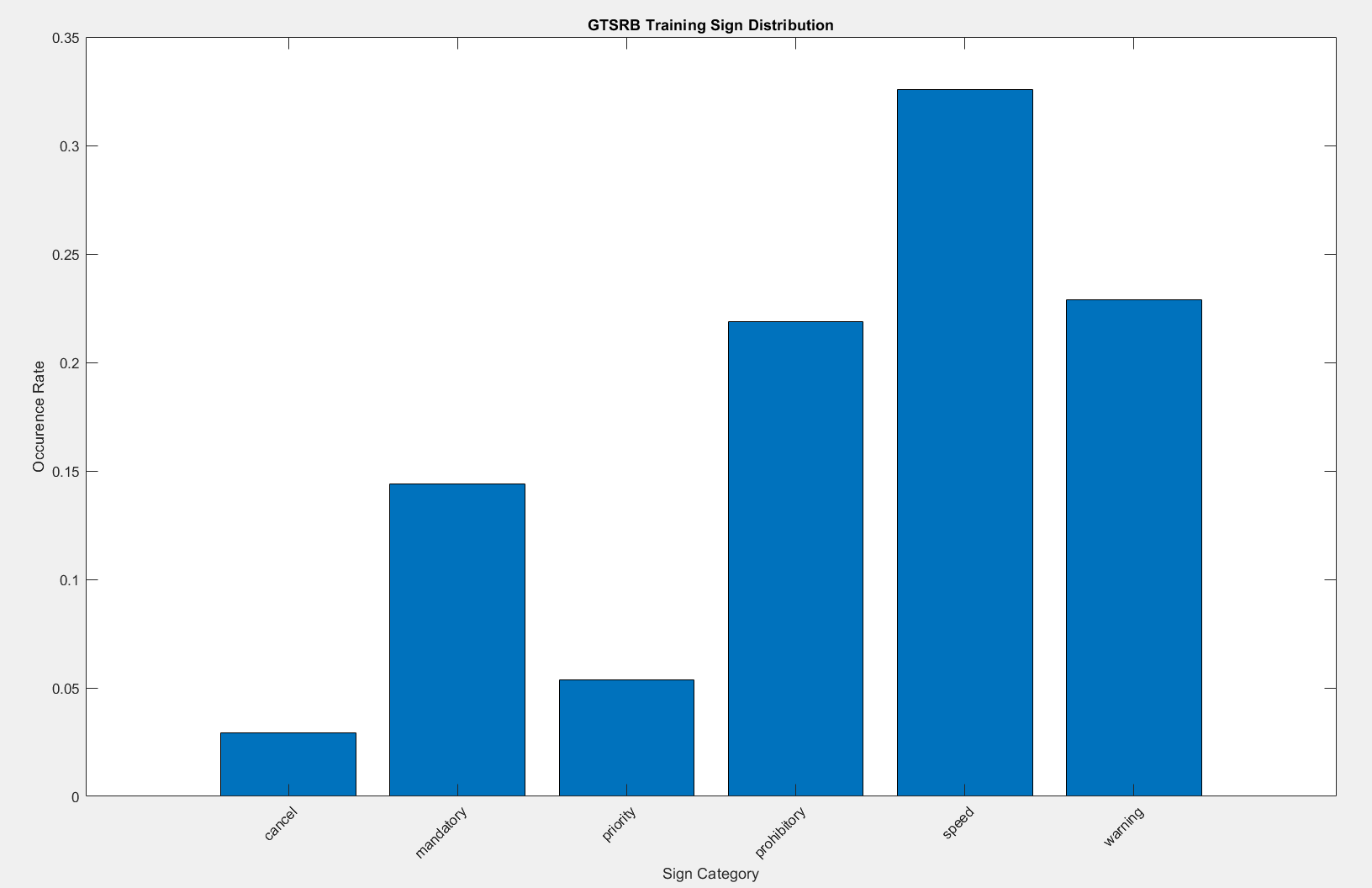
**Figure 9**. Grayscale histogram equalization

In light of these challenge with the dataset, in addition to resizing all images and performing histogram equalization, our team decided to reduce the overall size of the dataset by removing image data below a certain size (number of pixels) and image having large variance in intensity histograms. For instance, the low resolution and low contrast images seen above would both be removed from our dataset. This was done to improve the overall quality of the dataset, which we thought should lead to better classifier accuracy. We were able to achieve high performance classification results with high quality data. Future work on this project could assess the robustness of the models by reincorporating the lower quality images into training and testing.

Finally, it was noted that many of the signs in the dataset look extremely similar in terms of shape, color, etc. Thus, we predicted the classifier may have trouble distinguishing between similar signs (i.e. speed limit signs). Thus, we reduced the number of sign classes by grouping the signs based on similarity. Traffic signs of 70km/hr, 30 km/hr etc. are grouped as the speed sign, for example. By doing this, we can achieve high performance and fidelity in our model. Then, once the image is classified as a speed sign, for example, another algorithm (with different training parameters) could classify which the speed the sign is. The cascaded portion of the model has been left for future work. The groups of classes can be seen below in figure 10. These group labels are assigned to each of the training and test images and are used for performance analysis.



**Figure 10**. Groups of sign classes



**Figure 11**. Occurrence rate of each sign category

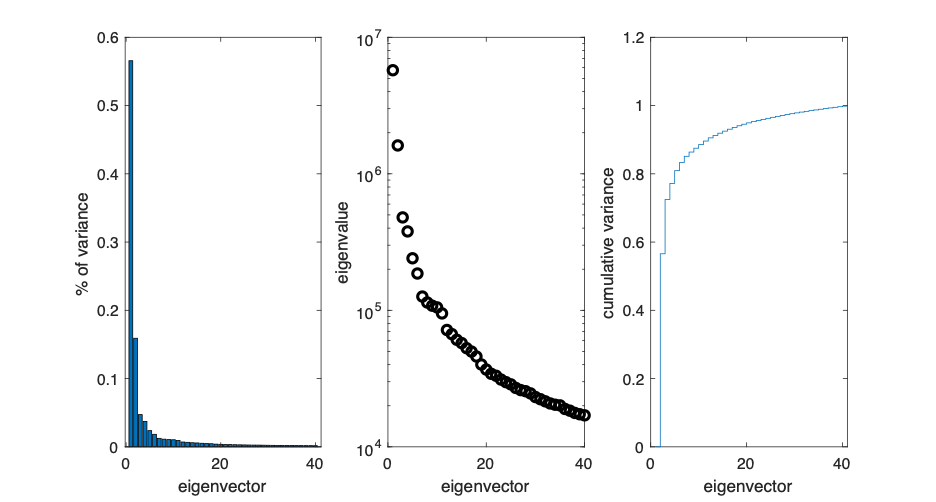
## Feature Extraction

Feature extraction techniques can be used to drastically reduce the dimensionality of a dataset with a large number of features or attributes. The reduction of data dimensionality means that one is able to represent the most important/interesting parts of a dataset using a much more compact, representative vector of features. This approach is extremely useful in image processing applications where image sizes can be rather large, so a reduced feature representation is required to efficiently complete machine learning tasks.

The GTSRB dataset is a large, lifelike dataset that contains around 50,000 total images. The images of the GTSRB dataset vary in size from 15 x 15 to 250 x 250 pixels. This means that for the largest images in the dataset, we have 62,500 observation features, if we were to use each image pixel as its own feature. When the number of features is similar to the number of observations, it is desirable to use feature extraction (dimensionality reduction) techniques to speed up model training, reduce the risk of overfitting, improve data visualization and ultimately improve classification accuracy.

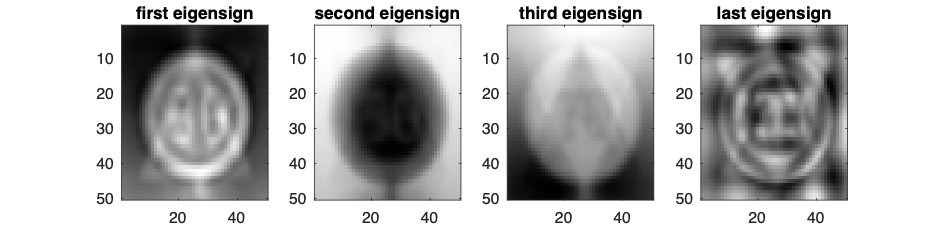
Thus, our aim was to summarize most of the information contained in the dataset of traffic signs images by creating new features from the original features (pixel intensities). One of the most used linear dimensionality reduction techniques is called Principal Component Analysis (PCA). PCA is an algorithm for finding a subspace with less dimensionality of the original data that maximizes the variance and minimizes the reconstruction error. In PCA, we compute the eigenvectors and eigenvalues of the covariance matrix of the original data to extract the eigenvectors with maximum eigenvalues. These vectors represent the direction of maximum variance in the data and become our principal components. For the purposes of this project, we can consider these vectors as “eigensigns.” The original data is projected onto the set of orthogonal, principal components to produce a set of weights or scores. We can then use the weights to take a linear combination of the principal components to reconstruct the original image if desired.

For the purposes of this project, we reduced the dimensionality of the GTSRB dataset using PCA to 40. This value was determined experimentally. It can be seen below (figure 12) that with 40 eigensigns, we can represent close to 100% of the entire dataset variance.



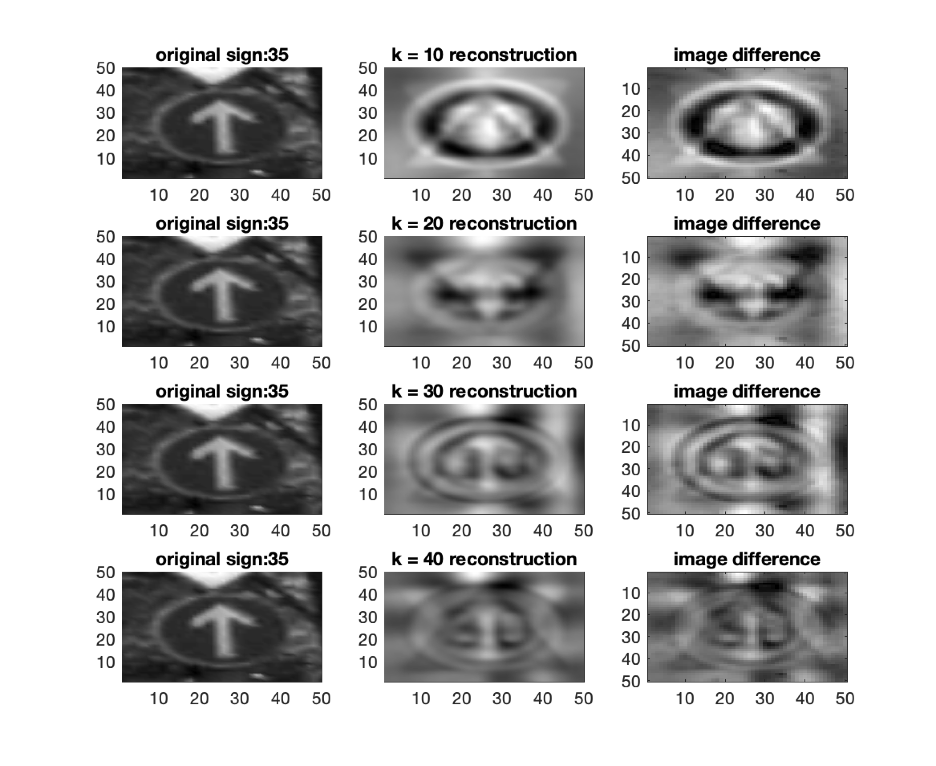
**Figure 12**. PCA dataset variance based on 40 eigensigns

Thus, after performing the data conditioning described in the previous section, we applied PCA to each of the training and test images to reduce the feature space to a total of 40 values per image. Below you can see the first three and the last eigensign:



**Figure 13**. First three and last eigensign

It’s interesting to note that the first eigen site contains the skeletons/borders of both circular and triangular signs, with what appear to be digits in the center. The second eigensign reverses those intensities in the first eigensign to account for darker signs. Below, you can see how one can reconstruct an example test sign using a linear combination of eignensigns. With more bases, the sign reconstruction begins to look more circular and contains the upward pointing arrow.



**Figure 14**. PCA reconstructed images compared with original signs

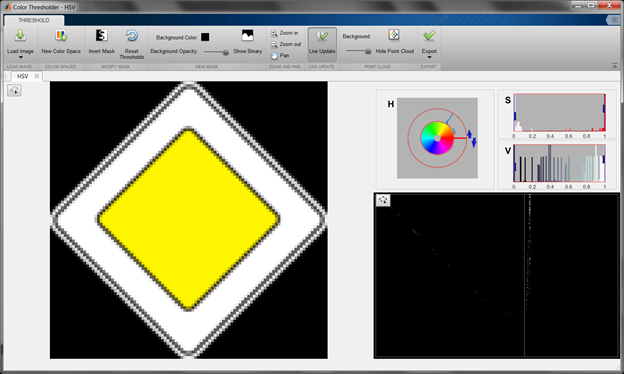
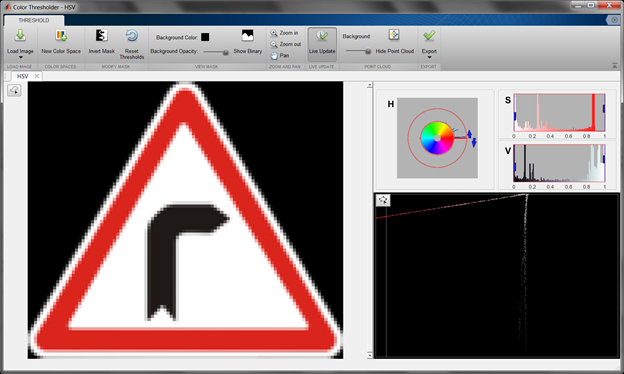
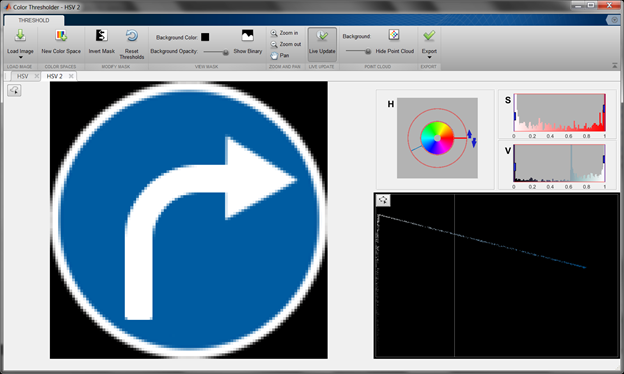
In the end, we used the PCA features to train and test each of our four models. Another idea our team discussed and began to implement was to reduce the feature dimensionality by designing a set effective features to distinguish various sign types using image processing techniques, rather than take all pixels as features or rely on PCA for the feature dimension reduction. Designing 20 or so features would have been ideal. The derived features discussed were sign shape, boundary color, presence of digits, presence of pictograms, histogram of hues and histogram of gradients. These derived features coincide with physical sign features that help humans to group and interpret different traffic signs. Below are some results of attempting to determine whether a traffic sign is a circle or not:



**Figure 15**. Results of attempting to determine circularity of traffic sign

Ultimately, shape, digit and pictogram detection didn't work very consistently for test images, given that the images are cluttered with background content and variations in illumination/ partial occlusions, etc., so we decided to just use pixel intensities and PCA for our data features. More advanced image processing techniques may be employed to perform these tasks on autonomous vehicles. This is another opportunity for future work on this topic.

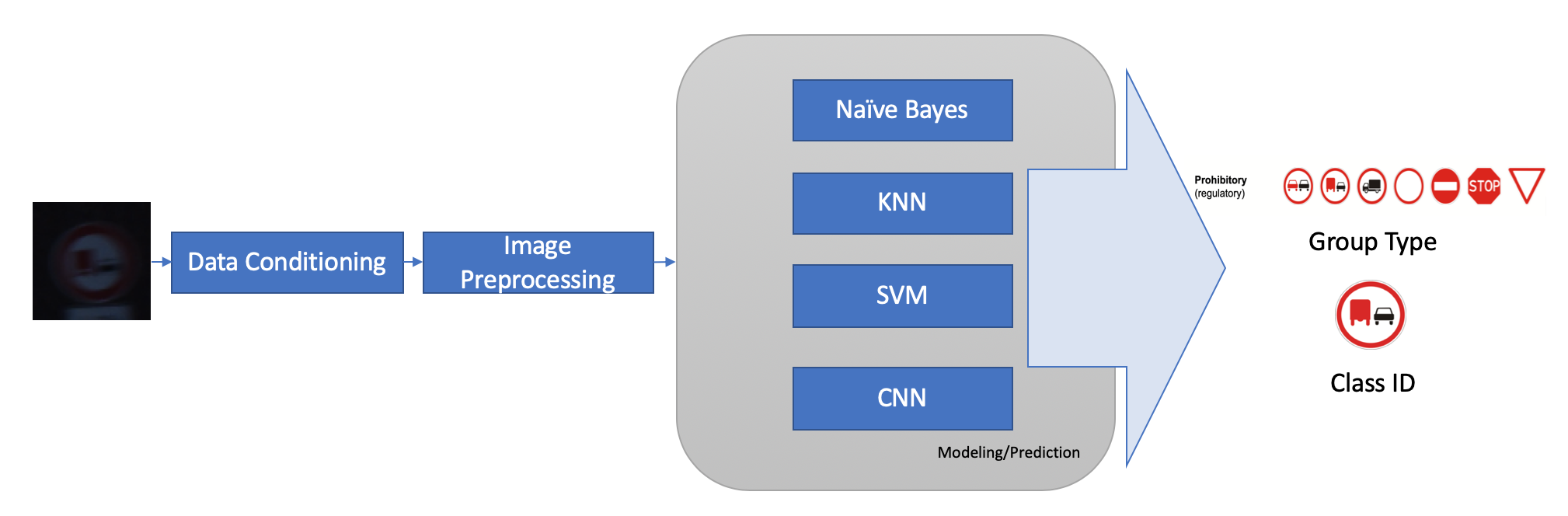
Given that we used grayscale histogram equalization and PCA, which removes the color information from the images, we thought it would be worth investigating the performance of our classifiers using color features, as these features are specifically designed to help humans interpret traffic signs. To this end, we extracted a histogram of hues for the test and training images for further analysis of the incorporation of color as features for our models. This works by binning the Hue values of images after converting the RGB image to the HSV color space. Given that the speed and warning signs have similar color patterns, we didn’t expect these features to work extremely well in differentiating between those signs classes.



**Figure 16**. Histogram of hues

## Modeling/Prediction

The GTSRB data set provides a wide range of variations between classes, hence our modeling approach takes into consideration multiple classifiers to find the optimal one. The four classifiers that were implemented are Naïve Bayes, K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Convolutional Neural Networks (CNN), pictured in figure 17. For each approach, histogram of hues or histogram equalization was applied to the data before running them through each classifier.



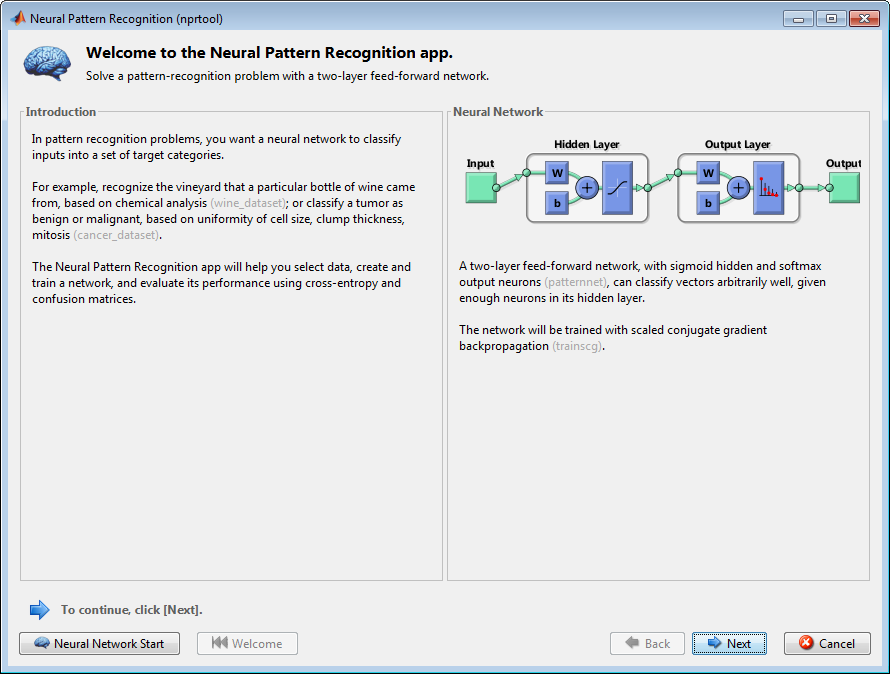
**Figure 17**. Modeling/prediction part of MLSP paradigm

In the Naïve Bayes approach, we apply the Bayes theorem in order to find the conditional probabilities among classes. The assumption in this case is that each feature in the data set is independent of each other, as opposed to the KNN and SVM classifiers. This classifier was chosen because it is a simple technique that is highly scalable. For the Naive Bayes classifier algorithm, the following input arguments are used: means of each class, the covariance matrices of each class, the a priori probability of each class, and the data vectors to be classified. The output would be predicted labels of the class corresponding to the data vectors given. The input arguments are calculated beforehand, based on the data vectors to be classified, before putting them into the algorithm.

The general idea in the KNN and SVM classifiers is that close elements are strongly related. In the KNN approach, we measured the distance between the test and the training data points in order to find the closest elements in the feature space. The final classification is based on a majority class of the k nearest data points. In this case, the number of neighbors selected was 5, given that it showed the best performance.

In the SVM classifier, we find a hyperplane that distinctly separates the data points. On each side of the plane are the classes used for classification. The general objective is to maximize the margin (distance between both classes) to allow more confidence when classifying future data points. The SVM classifier was implemented using the MATLAB function ‘fitcecoc’ using standardized predictors, a linear kernel function and a box constraint of 1 as parameters. This classifier uses the ‘one vs one’ classification technique with a total of 903 binary learners.

For CNN, a MATLAB GUI called ‘nprtool’, shown in figure 18, is used for training our neural network. The MATLAB GUI allows the user to set the number of hidden layers, and what training data and labels we want to put. From there, the user can set the percentages for dividing the training data up into training, validation, and testing. The standard 70%, 15%, and 15% is used for all trained CNNs. Validation is to validate that the network is generalizing and to stop training before overfitting, while testing is used as a completely independent test of network generalization. Training was done through the GUI using a two-layer feed-forward network with sigmoid output neurons, and training uses scaled conjugate gradient backpropagation. The number of output neurons is set to the number of categories. After training, testing data from the GTSRB test.csv file is used to test the network, and then performance is determined. The GUI also allows additional modifications to the network after training the network once. There are options to plot the confusion matrix, receiver operating characteristic, performance, training state, and error histogram from the GUI. Different hidden layer sizes are used for each CNN trained network, and they were either 10, 40, or 70. To clear things up from the presentation, the CNN classifier’s performances are evaluated through testing both the testing data sequestered from the original data set and the test data from GTSRB. The former is used to evaluate the performance of the classifier before applying the test data. Therefore, the transfer learning approach was employed.



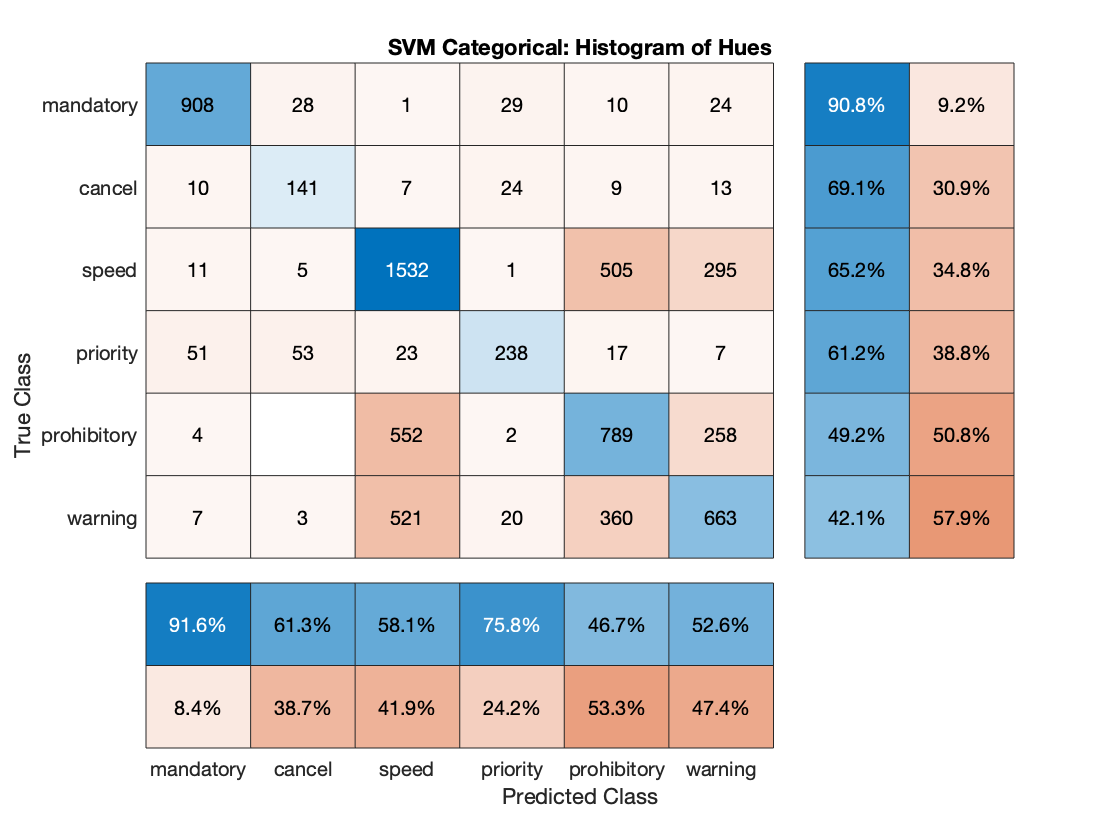
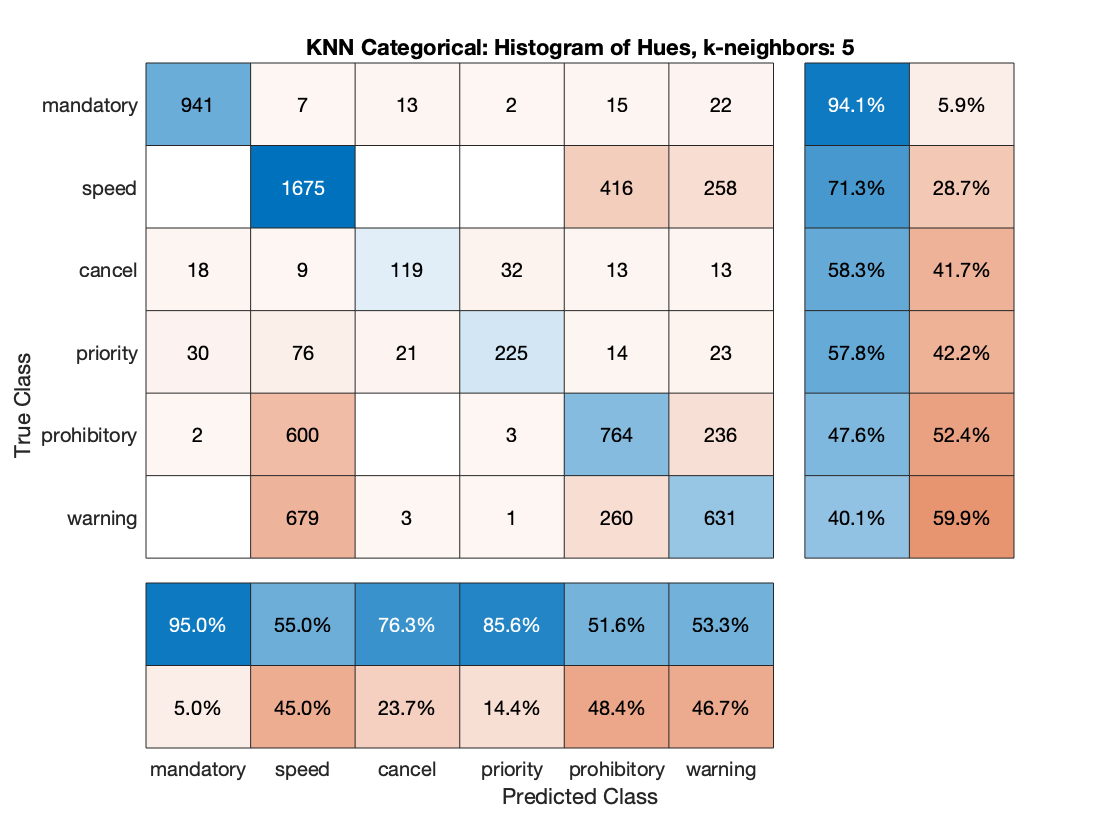
**Figure 18**. CNN ‘nprtool’ GUI training a neural network

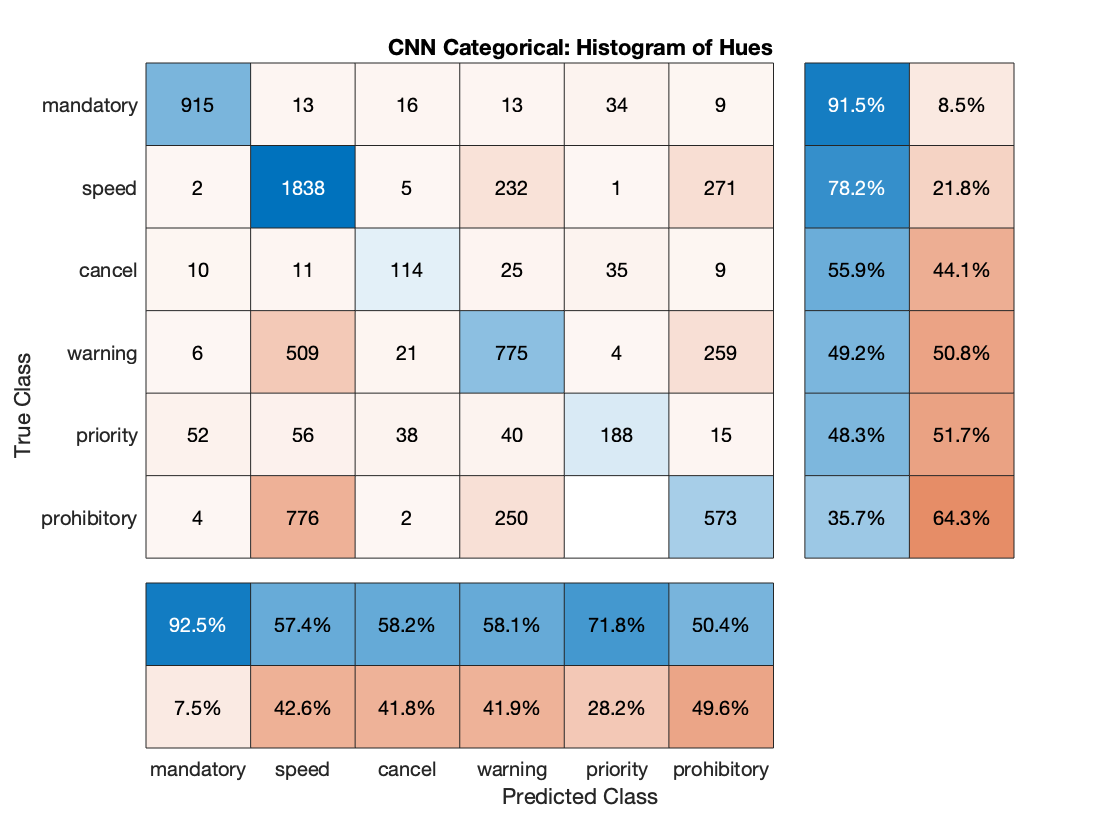
In terms of the dataset, the GTSRB data is split into a training and a test set. The training set includes over 39,000 images in a group of directories. Each directory corresponds to a single class and a list of images related to that traffic sign within the directory. This data is pre-processed as described in Section 3.3, and later loaded by the training algorithm to train the model.

In order to test the performance of the models, we use the test data set provided by GTSRB. The test set has over 12,000 images with varying sizes, lighting conditions, occlusions and other attributes found in real-life data. This data is particularly a smaller sample set compared to the training data, but has enough attributes to provide an unbiased evaluation of the model. The overall accuracy of a classifier is driven by the confusion matrix. We analyze the confusion matrix to evaluate the correct classification of the traffic signs.

# RESULTS

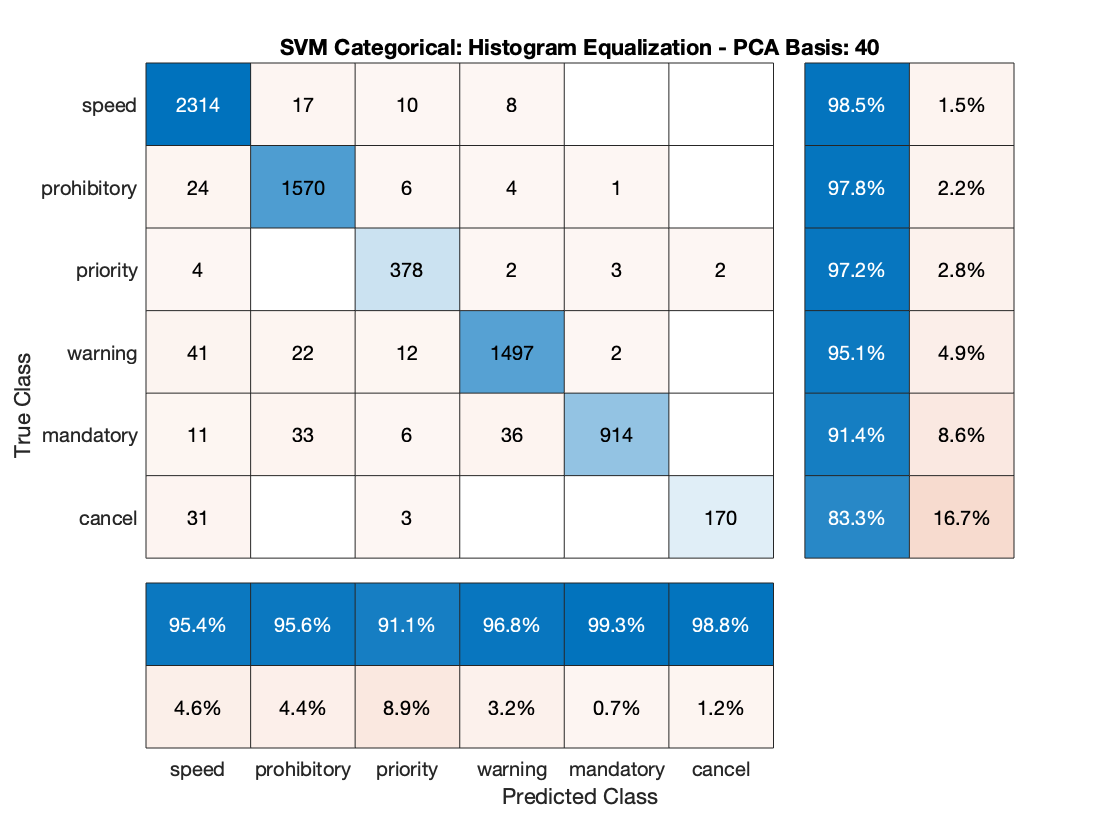
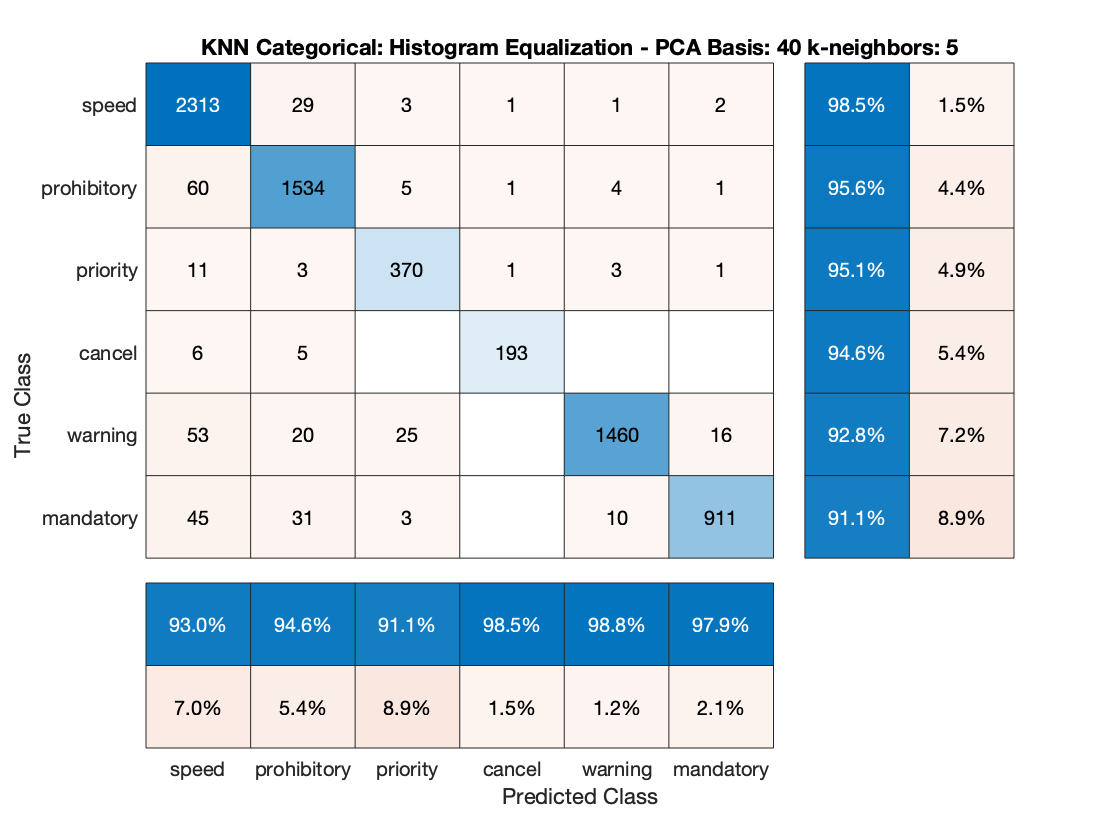
All the classifiers were trained and tested using two different input methods for comparison (Histogram of Hues vs Histogram Equalization). The dataset was conditioned and pre-processed as described in Section 3. Using the histogram of hues data, the classifiers showed an overall 60% categorical accuracy and a 22% class classification accuracy. The ‘mandatory’ group category has the highest score across all the classifiers. All the signs in this group category are white and blue which is a unique color compared to the signs in other group categories. The classifiers perform really well in this group category only because of the inherent limitation of considering color features only to classify a sign.

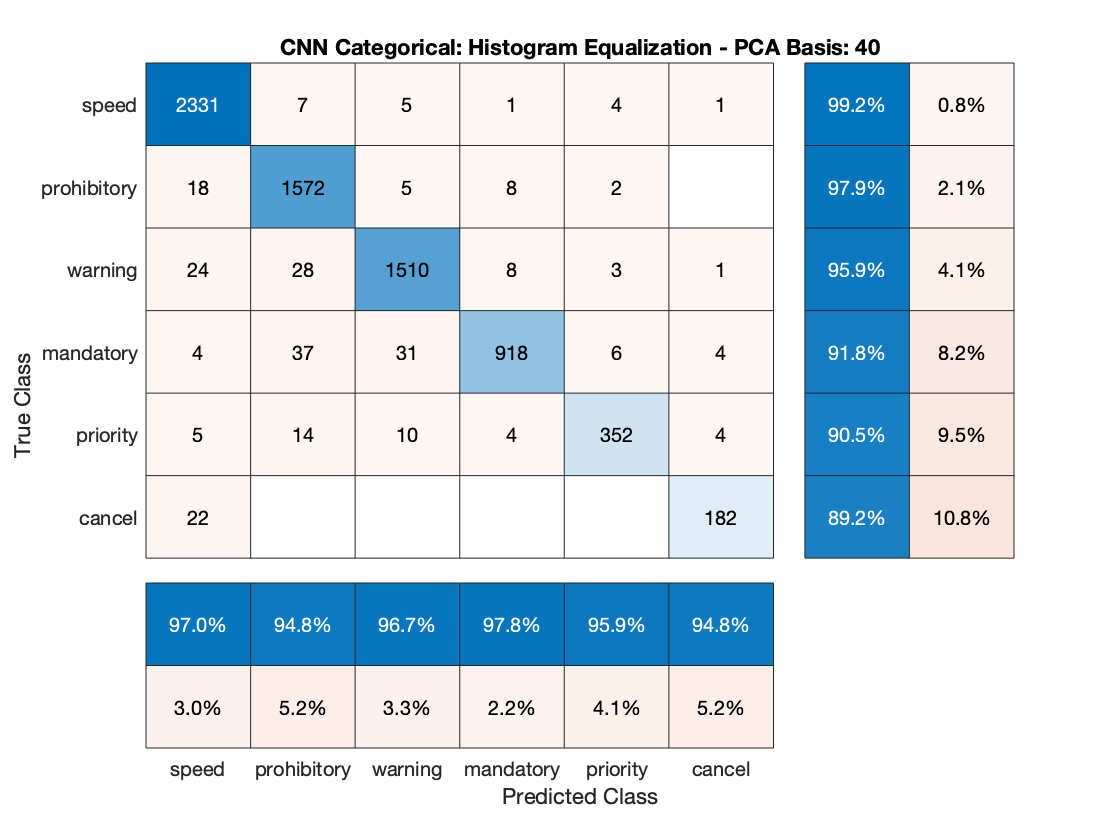




**Figure 19.** Confusion Matrices: Categorical Histogram of Hues

On the other hand, the histogram equalization data showed the best results in both categorical and class classification as shown in the images below. In this case, the features considered are the pixel intensities after enhancing contrast and reducing the dimensionality using 40 PCA basis. The CNN classifier had the best accuracy, with a 97% for categorical and 89% for class classification. The SVM classifier slightly underperformed (compared to CNN) with a 96% accuracy for categorical and 87% for class classification. Lastly, the KNN classifier showed a 95% accuracy for categorical and 72% for class classification.





**Figure 20.** Confusion Matrices: Categorical Histogram Equalization

For Naive Bayes, the team only performed class classification using 40 PCA basis with the histogram equalization data. This classifier was not suited for this data set as it had an error rate of 95%. Therefore, any further testing with other data conditioning and feature extraction methods were not considered.

Below in Table 1 is a summary of each classifier’s performance results.

**Table 1. Performance Results of each Classifier**

|  |  |  |
| --- | --- | --- |
| **Classifier – Input Type** | **Categorical Accuracy** | **Class Accuracy** |
| KNN – Histogram Equalization | 0.95 | 0.72 |
| KNN – Histogram of Hues | 0.61 | 0.22 |
| SVM – Histogram Equalization | 0.96 | 0.87 |
| SVM – Histogram of Hues | 0.60 | 0.22 |
| Naïve Bayes – Histogram Equalization | - | 0.05 |
| CNN – Histogram Equalization | 0.97 | 0.89 |
| CNN – Histogram of Hues | 0.60 | 0.22 |

# CONCLUSIONS

The team has exceeded expectations and completed all four proposed classifiers, including CNN which was originally going to be done if time permitted. We successfully implemented each stage of the MLSP paradigm using MATLAB. For sensor and data capture, we were able to obtain a real-world dataset from GTSRB and apply multiple conditioning techniques. The classification performance results were high for SVM, KNN, and CNN, and we decided to not proceed with Naive Bayes after seeing the suboptimal performance using the histogram equalization data.

For lessons learned, we learned that hue histograms do not work as well because some signs have very similar color distributions, so it’s not a good defining trait. We also learned that Naive Bayes doesn’t work well here compared to other algorithms, most likely because it is too simple. Another thing learned is that filtering out poor images and applying gray contrast boost before PCA greatly improves the results of the classifiers. It is also good to classify the signs based on categorical accuracy, to see if the inaccuracy from our class classification is at least somewhat close.

In the future, we could use HOG and Haar-like features in the GTSRB database to attempt different forms of classification not covered in class. There are also other feature extraction methods that we can try using with the HOG and Haar-like features for classification. With different feature extraction methods, we could look into classification algorithms that are better suited for those types of data. For example, Haar-feature classifiers are very similar to CNN, except a Haar-feature is manually determined, instead of where the values of the kernel are determined by training in CNN.

Our high classification rate is after excluding lower quality images in the dataset. The team could try to make lower quality images also classifiable with improved classifiers. The code we currently have is also not very organized, so we could refactor and reorganize our code for better readability and reusability.

# SUMMARY

The main objective of this project was to implement a multi-class classification model in order to classify traffic sign images provided by the German Traffic Sign Recognition Benchmark (GTSRB) data set. The team was able to implement a multi-class classification model with different classifiers (Naïve Bayes, KNN, SVM and CNN). The CNN classifier had the best results across all the classifiers with a 97% categorical and 89% class classification accuracy. The overall performance of the model was improved after several iterations of image pre-processing, data conditioning and feature extraction. The poor-quality images were also excluded from the data set and the rest of the images were resized to a common average size as part of the image pre-processing step. The team considered and tested multiple feature extraction techniques ranging from pixel intensity, principal component analysis, histogram of hues to sign shape recognition. Ultimately, the pixel intensities and PCA data features with enhanced contrast showed the best results. In terms of classifiers, Naive Bayes was concluded to not be suited for this data set, and SVM slightly underperformed compared to CNN, therefore supervised learning seems to be a better approach in this scenario.

The main challenge of classifying poor quality image in the dataset remains. The team has considered using other image features (HOG and Haar-like) for future work, but overall the main objective of this project was successfully completed.

# REFERENCES

[1] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1453–1460. 2011.

[2] “Preparing Your Dataset For Machine Learning: 8 Basic Techniques That Make Your Data Better.” *AltexSoft Blog*, Altexsoft, 16 June 2017, <https://www.altexsoft.com/blog/datascience/preparing-your-dataset-for-machine-learning-8-basic-techniques-that-make-your-data-better/>

[3] D. Levinger. “Six Steps to Master Machine Learning with Data Preparation.” *KDnuggets News,* KDnuggets, December 2018. <https://www.kdnuggets.com/2018/12/six-steps-master-machine-learning-data-preparation.html>

[4] “Traffic Sign Recognition (TSR).” *Honda Owners*, Honda, 2020. <https://owners.honda.com/vehicles/information/2020/Accord-Sedan/features/Traffic-Sign-Recognition>

[5] H. Neeb and C. Kurrus. “Distributed K-Nearest Neighbors.” *Stanford CME 323 Spring 2016 Project Reports,* Stanford University, 5 June 2016. <https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/neeb_kurrus.pdf>

[6] R. Gandhi. “Support Vector Machine - Introduction to Machine Learning Algorithms.” *Towards Data Science,* Medium, 7 June 2018. <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>